Hochschule Augsburg University of Applied Sciences

Diplomarbeit

Studienrichtung Informatik

Jorge Andrés Cuartas Monroy

Vergleich von Python-Webframeworks am Beispiel einer Webschnittstelle für Mikrocontroller

Prüfer: Prof. Dr. Hubert Högl
Abgabe der Arbeit: 17.1.2011

Verfasser der Diplomarbeit:
Jorge Andrés Cuartas Monroy
Rehmstr. 7
86161 Augsburg
Telefon: 0176 21232896
jandres.cuartas@googlemail.com

Fakultät für Informatik
Telefon: +49 821 5586-3450
Fax: +49 821 5586-3499

Hochschule Augsburg
University of Applied Sciences
Baumgartnerstraße 16
D 86161 Augsburg

Telefon +49 821 5586-0
Fax +49 821 5586-3222
http://www.hs-augsburg.de
poststelle@hs-augsburg.de

Sie dürfen:

- Das Werk bzw. den Inhalt vervielfältigen, verbreiten und öffentlich zugänglich machen
- Abwandlungen und Bearbeitungen des Werkes bzw. Inhaltes anfertigen

Zu den folgenden Bedingungen:

- Namensnennung — Sie müssen den Namen des Autors/Rechteinhabers in der von ihm festgelegten Weise nennen.
- Keine kommerzielle Nutzung — Dieses Werk bzw. dieser Inhalt darf nicht für kommerzielle Zwecke verwendet werden.
- Weitergabe unter gleichen Bedingungen — Wenn Sie das lizenzierte Werk bzw. den lizenzierten Inhalt bearbeiten oder in anderer Weise erkennbar als Grundlage für eigenes Schaffen verwenden, dürfen Sie die daraufhin neu entstandenen Werke bzw. Inhalte nur unter Verwendung von Lizenzbedingungen weitergeben, die mit denen dieses Lizenzvertrages identisch oder vergleichbar sind.

Wobei gilt:

- Verzichtserklärung — Jede der vorgenannten Bedingungen kann aufgehoben werden, sofern Sie die ausdrückliche Einwilligung des Rechteinhabers dazu erhalten.
- Sonstige Rechte — Die Lizenz hat keinerlei Einfluss auf die folgenden Rechte:
  - Die gesetzlichen Schranken des Urheberrechts und sonstigen Befugnisse zur privaten Nutzung
  - Das Urheberpersönlichkeitsrecht des Rechteinhabers
  - Rechte anderer Personen, entweder am Lizenzgegenstand selber oder bezüglich seiner Verwendung, zum Beispiel Persönlichkeitsrechte abgebildeter Personen.
- Hinweis — Im Falle einer Verbreitung müssen Sie anderen alle Lizenzbedingungen mitteilen, die für dieses Werk gelten. Am einfachsten ist es, an entsprechender Stelle einen Link auf diese Seite einzubinden.

  Sie können den Lizenztext unter `http://creativecommons.org/licenses/by-nc-sa/3.0/deed.de` nachlesen.

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Augsburg, den 17.01.2011

_____

Jorge Andrés Cuartas Monroy

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Kapitel 1.0

Einführung

1.1 Beispielanwendung

Im Zuge der Diplomarbeit soll neben einem Vergleich von Python-Webframeworks auch eine Anwendung für das Rechnertechniklabor der HS Augsburg entwickelt werden. Dabei soll der Zugriff auf Informationen und Verwaltung der Mikrocontroller in Form einer Webanwendung möglich sein. Der Benutzer soll die einzelnen Boards reservieren und anschließend per SSH über das Internet mit den Boards arbeiten können. Damit ein reibungsloser Umgang mit den Boards ermöglicht wird, darf ein Board nicht von mehreren Benutzern gleichzeitig reserviert werden. Dem Benutzer soll aber möglich sein, mehr als nur ein Board über die Oberfläche parallel zu reservieren. Darüber hinaus wird folgende Funktionalität für den Umgang mit den Boards über die Weboberfläche zugänglich.

- Reservierung
- Freigeben
- Statusinformationen abrufen
- Power On/Off
- Reset

1.1.1 User Arbeitszyklus

Um somit diese Befehle ausführen zu können, muss sich der Benutzer vorher registrieren, um Zugang zu erhalten. Dazu wird ihm ein Formular zur Verfügung gestellt. Nach dem Absenden wird überprüft, ob alle benötigten Daten richtig in die Formularfelder eingegeben wurden. Nach erfolgreicher Eingabe der Daten werden dem Benutzer ein Aktivierungslink und die Zugangsdaten per Mail zugesandt. Der Login wird dem User nur dann ermöglicht, wenn dieser seinen Account aktiviert hat.

Nach der Anmeldung wird der Benutzer auf die eigentliche Startseite weitergeleitet und bekommt eine Übersicht der Boards, die für eine mögliche Reservierung zur

Verfügung stehen. Dabei wird ihm angezeigt, welche frei bzw. welche bereits von anderen Usern reserviert worden sind. Nach der Reservierung eines Boards stehen ihm die oben genannten Möglichkeiten zur Verfügung. Durch Reservierung eines Boards wird der Linux-Zugang unter dem das Board zur Verfügung gestellt wird, resettet, und eventuell noch eingeloggte User abgemeldet. Daraufhin wird ein neues Passwort generiert und dem User werden die Zugangsdaten mitgeteilt. Mittels *SSH* ist es dann möglich auf das Board über einen Server zuzugreifen und entsprechend anzumelden. Nach der Anmeldung werden dem Benutzer alle Möglichkeiten erteilt, um auf dem Mikrocontroller zu arbeiten und z.B. ein Rootfilesystem auf dem Board über z.B. *SCP* aufzuspielen.

Nach Beendigung der Arbeit mit dem Board ist es möglich das Board freizugeben. Wurden mehrere reserviert, können sie einzeln freigegeben werden. Auch nach einem *Logout* seitens des Users werden alle vom User noch reservierten Boards automatisch freigegeben. Ähnlich wie bei der Reservierung werden die Linux-Accounts der freigegebenen Boards gesperrt und ein zufälliges Passwort erzeugt, um eine erneute Anmeldung zu verhindern. Auch in diesem Fall werden alle per *SSH* eingeloggten User ausgeloggt, um sicherzustellen, dass der Account wieder benutzbar ist. Werden nach Beendigung der Arbeit mit den Boards diese nicht freigegeben oder das Logout vergessen, steht dem Administrator der Anwendung jederzeit die Möglichkeit offen, noch registrierte Boards freizugeben. Auch hier entweder nur einzelne oder alle reservierten Boards auf einmal. Dieser Administrationslink steht jedem User zur Verfügung, der über die Adminoberfläche von Django von einem Superuser für die Rolle des Administrators freigeschaltet wurde.

1.1.2 Verwendete Technologien und Herangehensweise

Die Anwendung wird mittels Django entwickelt und zur Verfügung gestellt. Die Verbindung zur Außenwelt geschieht mittels *Apache* und *mod_wsgi*. Nach dem Durchlesen dieser Arbeit soll der Leser ein Verständnis bekommen, wie Webanwendungen im Python Umfeld erstellt werden. Verschiedene Python Frameworks werden dabei mit Django verglichen und Parallelen zu diesem gezogen. Die Wahl fiel auf Django, weil dieses sehr verbreitet ist und eine sehr gute Dokumentation bietet. Darüber hinaus stehen dem Entwickler Tools zur Verfügung, die die tägliche Arbeit im Umgang mit Django erleichtern. Die hier vorgestellten Frameworks bieten ähnliche Funktionalität und unterscheiden sich in Details wie z.B. in einer anderen Herangehensweise. Django stellt Tools bereit, um

ein neues Projekt zu erstellen, weitere Unterapplikationen zu generieren oder ein Server für Testzwecke.

Die Anwendung wird lokal unter Ubuntu[1] mit Hilfe von Git als Versionsverwaltung erstellt und Eclipse[2] als Entwicklungsumgebung benutzt. Zusätzlich wurden die Eclipse Plugins Pydev, Vrapper und Aptana installiert.

Auf dem Server, der von der Hochschule Augsburg zur Verfügung gestellt wird, steht Debian Lenny[3] in der Version 5.0.5 als Betriebssystem mit allen nötigen freien Softwarepaketen zur Verfügung.

1.1.3 Python und Webentwicklung

Normalerweise wird heutzutage bei der Webentwicklung hauptsächlich an *PHP* als "die Web-Sprache" gedacht, für die es unzählige Frameworks und Anwendungen[4] gibt. Auch wird oft Ruby mit *Ruby on Rails* im gleichen Zusammenhang in einem Atemzug genannt, bei der es auch eine große Entwicklergemeinde gibt und ebenso viele bekannte Projekte[5] existieren.

Der angehende oder erfahrene Python-Entwickler soll mit dieser Arbeit zusätzlich erkennen können, dass seine hart erarbeiten Python-Kenntnisse auch im Webumfeld einsetzbar sind und keine neue Programmiersprache für gute und stabile Webanwendungen erlernt werden muss. Mit Django, Pylons, TurboGears, Repoze.bfg und weiteren, hier nicht aufgezählten Frameworks gibt es genug Alternativen für die Webentwicklung unter Python, die das Versprechen schneller Realisierung von Anwendungen einhalten können.

1.2 Durchführung

Bei der Entwicklung von Webanwendungen werden verschiedene Techniken eingesetzt, die miteinander harmonieren müssen. Für die Erstellung der Programmlogik ist eine entsprechende Programmiersprache auszuwählen, die den Anforderungen entspricht. Bei Verwaltungsprogrammen fallen Daten an, die mit Hilfe der Programmlogik verarbeitet

[1] Ubuntu: `http://www.ubuntu.com/`, 28.11.2010
[2] Eclipse IDE: `http://www.eclipse.org/`, 28.11.2010
[3] Debian Lenny: `http://www.debian.org/releases/stable/`, 28.11.2010
[4] Magento: `www.magentocommerce.com`, Atmail: `www.atmail.com`, 28.11.2010
[5] Bekannte Ruby Projekte: `www.twitter.com`, `www.shopify.com`, `www.github.com`, 28.11.2010

werden und für deren Persistenz eine Datenbank zur Verfügung stehen sollte.

Um einen reibungslosen Ablauf beim Datentransfer zwischen Entwicklungs- und Produktionsserver zu gewährleisten, ist zusätzlich eine gute Strategie und Technologie nötig, um die Konsistenz der Daten an verschiedenen Stellen zu erhalten. Näheres dazu im Abschnitt 1.2.4

1.2.1 Webschnittstelle

Da es sich bei dieser Arbeit um eine Webanwendung handelt, kommen für die Interaktion des Benutzers mit der Schnittstelle XHTML[6], CSS und JavaScript infrage. Für die Struktur wird XHTML eingesetzt, CSS2[7] wird zur Gestaltung benötigt und JavaScript erleichtert die Benutzerführung.

Auch im JavaScript Bereich gibt es verschiedene Frameworksh. Beispiele für JavaScript-Frameworks[8] sind jQuery, Dojo oder Prototype, mit denen eine vereinfachte Interaktion mit dem Benutzer umgesetzt werden kann. Da diese Frameworks die Möglichkeit bieten Ajax[9] Funktionalität einzusetzen, kann eine ressourcenschonende Applikation mit schnellen Reaktionszeiten erstellt werden. Grund hierfür ist, dass nicht alle Teile der Anwendung neu geladen werden müssen, sondern nur die wirklich benötigten.

Zum Einsatz kommt jQuery mit schnellen Ausführungszeiten[10], einer guten Dokumentation und guter Handhabung im Vergleich zu normaler JavaScript Syntax. In der Anwendung wird auf dem Template für Boardreservierungen jQuery eingesetzt, da für das Absetzen der Befehle wie z.B. Registrierung oder die Freigabe der Boards die Seite nicht neu geladen, sondern lediglich die Django-View, welche die gewünschte Funktionalität bietet, angestoßen werden muss. Das Feedback über die erfolgreiche Ausführung der Views wird mittels DOM[11]-Manipulation und Ajax bzw. entsprechende jQuery Funktionen dem User angezeigt.

Auf folgender Vorlage, zu sehen in Abbildung 1.1, die mittels HTML und CSS erstellt

[6] Extensible HyperText Markup Language, `http://www.w3.org/TR/xhtml1/`, 28.11.2010
[7] Cascading Stylesheets Level 2, `http://www.edition-w3.de/TR/1998/REC-CSS2-19980512/`, 28.11.2010
[8] JavaScript-Frameworks: `http://webstandard.kulando.de/post/2008/08/21/top-10-aller-javascript-frameworks`, 28.11.2010
[9] AJAX: Asynchronous Javascript and XML, `http://openbook.galileocomputing.de/javascript_ajax/18_ajax_001.htm`, 28.11.2010
[10] Reaktionszeit jQuery: `http://www.golem.de/1002/73272.html`, 28.11.2010
[11] Document Object Modell: `http://www.w3.org/DOM/`, 28.11.2010

worden ist, basiert die GUI der Anwendung. Es wurde durchgehend darauf geachtet, dass *em* als Maßeinheit verwendet wird, um ein elastisches[12] Layout sicherzustellen. Der Vorteil liegt hier in der besseren Zugänglichkeit der Seite und der Anpassung des Designs an die Schriftgröße des Browsers, welche der User benutzt. So bleiben Skalierungen proportional und es entstehen weniger negative Seiteneffekte beim Zoomen. Dies kann z.B. das Umbrechen des Layouts sein, wenn nicht genug Platz in der Breite für eingesetzte Bilder zur Verfügung steht.

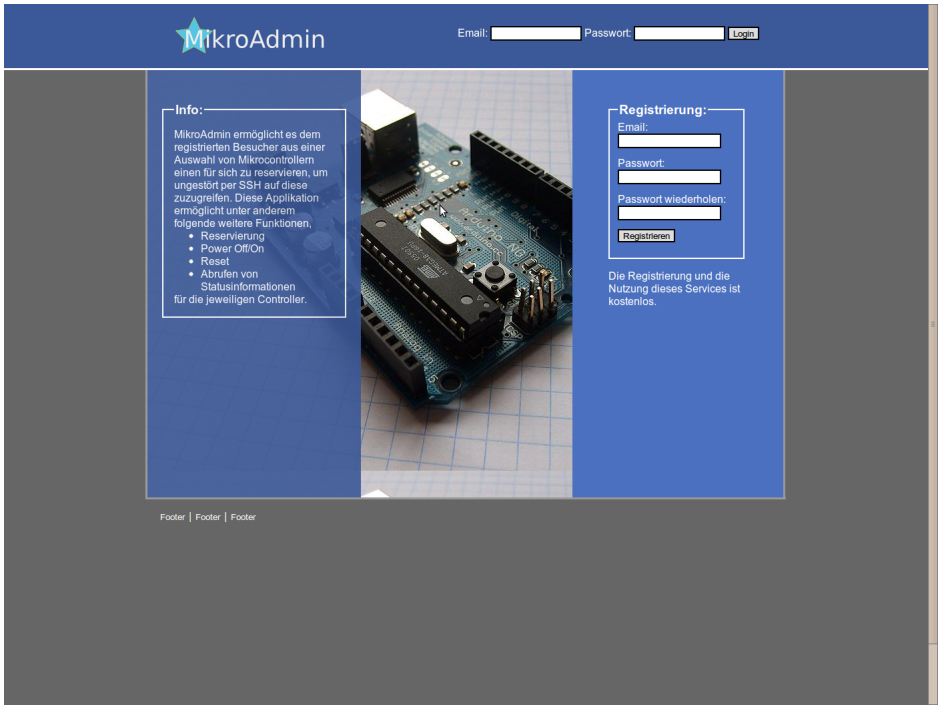

Abbildung 1.1: GUI der Anwendung

Aufbauend auf der *index.html*-Datei werden weitere Templates für die Seite abgeleitet, um erstens globale Änderungen an die darunter liegenden Templates weiterzugeben und zweitens das Python-Prinzip *DRY: Don't Repeat Yourself* so gut es geht zu übernehmen. So kann der XHTML-Aufwand der Seite so gering wie möglich gehalten werden und die Struktur konsistent.

Alle hier gezeigten Codebeispiele werden bei Gitorious[13] gehostet und sind mittels

[12] Elastische Layouts: `http://greattalk.ch/2008/02/09/die-unglaublichen-em-elastischen-layouts-mit-css/`, 28.11.2010

[13] Main Repository: `http://gitorious.org/webframeworks`, 28.11.2010

Git[14] als Versionskontrollsystem verfügbar.

1.2.2 Verwendete Datenbank

Daten der Anwendung, die persistent gehalten werden sollen, dienen hauptsächlich der Beschreibung der Mikrocontroller. Darüber hinaus werden zur Interaktion User mit ihren Daten benötigt, welche auch gespeichert werden müssen. In der Datenbank werden folgende Informationen für jeden Mikrocontroller festgehalten. C.3

- name
- description
- linuxusername
- currentPassword
- isReservedBy
- reservedDate

Diese Informationen sind nötig um ein Objekt zu verwalten. Unter der Beschreibung *description* wird ein Link hinterlegt, um weitere Informationen über das genutzte Board zu erhalten.

In *currentPassword* wird das momentan geltende Passwort festgehalten, mit dem sich der User auf der Konsole unter dem Board-Account anmelden kann, dieses wird bei der Freigabe der Boards oder einem Logout des Users überschrieben. Diese Information ist insofern wichtig, da ohne passendes Feld im Board-Objekt jedes mal beim Aufruf des Boards auf der Oberfläche ein neues Passwort generiert werden müsste, um es dem User für den Login zur Verfügung zu stellen. So wird nur dann ein neues Passwort generiert, wenn der User ein Board zum ersten Mal reserviert.

Das *reservedDate*-Feld wird genutzt, um anderen Usern die Information zu liefern, zu welchem Zeitpunkt ein Mikrocontroller reserviert wurde.

Für Vergleiche und Bereitstellung der Rechte zur Ausführung der Board-Funktionen wird *isReservedBy* benötigt. Zusätzlich kann so die Emailadresse des Users für andere Benutzer der Applikation als Kontanktinformation ausgegeben werden, da durch diese

[14]Git: `http://git-scm.com/`, 28.11.2010

Information auf den User geschlossen werden kann, der das Board reserviert hält. So kann dieser in dringenden Fällen zur Freigabe des Boards aufgefordert werden.

Die anderen Felder sind soweit selbsterklärend und werden für die Identifizierung des Boards und der Verwaltung benötigt.

Als Datenbank wird *SQLite*[15] verwendet. Da keine große Daten für die Anwendung verwaltet werden müssen und es sich nicht um eine zeitkritische Anwendung handelt, ist die Wahl auf SQLite gefallen. Auch aus dem Grund, da SQLite schnell produktiv verwendet werden kann und ohne größere Konfiguration einsetzbar ist. Daten werden in gewöhnlichen Dateien gespeichert, was eine Portierung auf einen anderen Server, falls notwendig, erleichtert. Ferner ist die Software *Public Domain* und kann ohne Lizenzprobleme in jedem Projekt eingesetzt werden. Seit der Python Version 2.5[16] wird SQLite bei der Installation von Python mitgeliefert. Für zusätzliche Administrationszwecke und den Zugriff über die Konsole auf die Datenbank ist die Installation von SQLite in der Version 3 zusätzlich notwendig.

1.2.3 Anwendungslogik mittels Python

Alle im weiteren Verlauf näher vorgestellten Webframeworks werden mit Hilfe von Python entwickelt. Auch die Logik der Anwendung, im MVC[17] Pattern Controller genannt, wird mit Python erstellt. Es kann passieren, dass die Templatesprachen der jeweiligen, vorgestellten Webframeworks nicht 100%ig auf Python aufbauen, aber von dieser Programmiersprache abstammen und z.B. Kontrollstrukturen, wie Abfragen oder Schleifen mit ähnlicher Syntax einsetzen. Darüber hinaus kann vollständig auf die Python-Bibliothek zugegriffen werden und der Anwender ist nicht darauf beschränkt nur frameworktypische Bibliotheken einzusetzen. So steht dem Webentwickler beim Einsatz von Python die ganze Funktionalität, die diese Programmiersprache bereithält, auch im Web zur Verfügung.

So weit möglich wird ein objektorientierter Ansatz verfolgt, vornehmlich bei der Strukturierung der Objekte und beim Einsatz von OR-Mappern bei Änderungen von Eigenschaften und somit von Daten. Formulare zur Userinteraktion können in verschiedenen Frameworks mittels Klassen definiert werden und diese Art der Formularerzeugung wird

[15] SQLite: `http://www.sqlite.org/`, 28.11.2010
[16] Python und SQLite: `http://docs.python.org/library/sqlite3.html`, 28.11.2010
[17] Model View Controller: `http://c2.com/cgi/wiki?ModelViewController`, 28.11.2010

z.B. in Django mittels des *forms*-Pakets[18] ermöglicht. Pakete zur Generierung von Formularen werden so oft wie möglich eingesetzt, Gründe für die Benutzung sind die einfache Einbindung gleicher Formulare in verschiedenen Templates und somit die Verwaltung der Formulare an zentraler Stelle.

Die einfache[19] und schnelle Entwicklung von Programmen unter Python kann mit Hilfe von Webframeworks wie Django in die Welt der Webentwicklung übertragen werden.

1.2.4 Git Arbeitsstrategie

Zur Versionskontrolle wird wie in 1.2.1 bereits erwähnt, Git eingesetzt. Die Entscheidung fiel aus der Erfahrung heraus auf Git[20], wegen des einfacheren Umgangs und des leichteren Auflösens von Merge Problemen im Vergleich zu SVN[21], was auch als Alternative zur Auswahl stand.

Bei der Erstellung der Mikroadmin-Anwendung wurde die Grundkonfiguration von Django zuerst auf dem Produktionsserver vorgenommen und getestet, sobald ein grundsätzliches System stand, dieses auf Gitorious in ein entsprechendes Repository übertragen. Der Vorteil an Gitorious z.B. gegenüber Github ist, dass Projekte gruppiert werden können. So kann ein Projekt, welches aus verschiedenen Repositories besteht, unter einer URL aufgerufen werden.

Nach der Übertragung der Grundkonfiguration auf Gitorious wurde die eigentliche Django-Anwendung mit dem Befehl *git clone* auf dem Entwicklungsserver gespiegelt. Während der Entwicklungszeit wurden anschließend mittels *git push* die neuesten Versionen auf den Upstream gelegt. Auf der Serverseite konnten dann die neuesten Änderungen mit *git pull origin master* auf den neuesten Stand gebracht und getestet werden.

Der Vorteil an dieser Strategie liegt darin, mit der Entwicklungsumgebung lokal z.B. Eclipse arbeiten zu können und so wenig wie nötig Änderungen auf dem Server direkt vornehmen zu müssen. So ist ein einfacherer Umgang mit den Programmdateien gewährleistet. Durch Dreifachsicherung an den verschiedenen Stellen ist auch der Verlust von Programmcode gering, da Git kein zentrales Repository benötigt und alle drei Kopien

[18] HTML Formulare mittels Klassen: `http://docs.djangoproject.com/en/1.2/ref/forms/api/`, 28.11.2010
[19] Warum Python einsetzen: `http://pythoncard.sourceforge.net/what_is_python.html`, 28.11.2010
[20] Git im Vergleich: `http://de.whygitisbetterthanx.com/`, 28.11.2010
[21] SVN: `http://subversion.apache.org/`, 28.11.2010

synchron gehalten werden. In diesem Fall dient die Einbindung eines Onlinerepositories als zentraler Umschlagplatz zwischen Produktions- und Entwicklungsrechner. Wegen des Web- und Open-Source-Charakters der Anwendung und der Sicherung des Arbeitsstandes an zentraler Stelle ist das ein großer Vorteil, da so jeder Interessierte immer auf die aktuelle Version online zugreifen kann.

Weitere Vorteile mit Git ergeben sich dadurch, dass z.B. die Menge der zu übertragenen Daten auf einem Mindestmaß gehalten werden, da nur die Deltas, also nur die Veränderungen an Dateien, übertragen werden. So verringert sich die Menge der zu übertragenen Informationen und Änderungen an der Codebasis stehen schnell an allen genutzten Stellen zur Verfügung. Im Gegensatz z.B. zur Übermittlung von Dateien über SCP oder FTP auf den Server, da auch nur bei geringer Änderungen diese vollständig neu übertragen werden müssen. Auch die richtige Versionierung auf den drei Parteien - Lokal, Server und Gitorious - wird so durch den Einsatz von Git vereinfacht.

Kapitel 2.0

Webframeworks

2.1 Aufbau eines Webframeworks

Webframeworks sind im Grunde eine Sammlung von Werkzeugen, die es dem Webentwickler erlauben Anwendungen schnell und effizient zu schreiben. Die Funktionalität besteht normalerweise aus verschiedenen Bibliotheken, die den Zugriff auf Datenbanken ermöglichen, die Erstellung von Templates für die Ausgabe in HTML erleichtern oder die Erstellung von Klassen und Methoden für die Geschäftslogik strukturieren. Die meisten Frameworks folgen somit dem MVC-Pattern, also der Trennung der Datenstruktur, der Darstellung beziehungsweise der Ausgabe der Daten und der verbindenden Applikationslogik.

Webanwendungen, die sich auf Datenbanken stützen können mit Webframeworks relativ schnell erstellt werden, dabei helfen OR-Mapper[1], die aus Klassenkonstrukten SQL[2] für eine ausgewählte Datenbank erstellen. Die Mapper bieten darüber hinaus eine eigene Sprache, um auf die Klassen bzw. Informationen in der Datenbank zuzugreifen. Der Vorteil liegt darin, dass der Code unabhängig von einer Datenbank erstellt werden kann und somit bei einer Umstellung auf eine Andere, keine oder nur geringfügige Anpassungen nötig sind.

Die Geschäftslogik kann abhängig vom Framework in einer beliebigen Programmiersprache geschrieben werden, meistens aber in Java, PHP, Ruby oder Python. Dieser Code dient als Bindeglied zwischen den Daten und der Darstellungsebene und ist je nach Implementierung[3] des MVC-Patterns, der wesentliche Teil der Applikation.

Um die Ausgabe zu generieren, bringen die Frameworks eine Template Engine[4] mit. Diese ermöglicht die einfache Ausgabe von Inhalten aufgrund der Mischung von HTML und Templatecode. Informationen werden von der Geschäftslogik an die View und somit an das zugehörige Template weitergereicht. Ein Einsatz von rudimentären Kontrollstrukturen mit diesen Templatesprachen ist möglich, dabei werden diese vor der Erzeugung

[1] Objekt Relationaler Mapper: `http://pythonnotes.blogspot.com/2004/09/python-orm-tools.html`, 30.11.2010

[2] Structured Query Language: Einstieg in SQL [6, Kap. 3, S. 17]

[3] MVC - Passive and Active Model: `http://msdn.microsoft.com/en-us/library/ff649643.aspx`, 30.11.2010

[4] Template Engines in Python: `http://wiki.python.org/moin/Templating`, 31.11.2010

von HTML ausgewertet.

Der modulare Aufbau eines Frameworks bietet viele Vorteile. Vor allem, wenn die einzelnen Teile gegen andere ausgetauscht werden können, falls sich die Anforderungen an die Anwendung ändern. Nachteilig ist hingegen, dass der Entwickler an die Vorgaben, die ein Framework vorgibt gebunden ist. Wie z.B. die Konvention wie und wo Dateien abgespeichert werden müssen, um vom Framework automatisch gefunden zu werden. Diese Konventionen werden von manchen Frameworks gefordert. Weiterhin können Datenbanken vorgegeben sein oder nur bestimmte JavaScript Bibliotheken unterstützt werden. Dies schadet zwar der Flexibilität, kann aber Struktur und Standardisierung in ein Projekt bringen und den Einstieg in das Framework erleichtern. Vor allem dann, wenn der Entwickler mit *best practices* beraten wird und die Anzahl der austauschbaren Modulen überschaubar bleibt.

2.2 Python und Web

> Ja, Python ist gewachsen und hat sich zu einer ausgereiften Sprache samt eigenem Ökosystem entwickelt[5].

Python ist eine sehr flexible Programmiersprache. Sie kann auf verschiedenen Gebieten eingesetzt werden. Unter anderem bei der Verarbeitung von XML, zur Verbindung von Datenbanken und Programmcode, in der Netzwerk Programmierung, im Webbereich und auch die Erstellung von Programmen mit grafischer Oberfläche (wxPython, PyQt) ist möglich.

Python ist nicht an ein bestimmtes Programmierparadigma gebunden und je nach Aufgabe kann ein Paradigma, wie z.B. Objektorientierung, Funktionale Programmierung oder ein aspektorientierter Ansatz gewählt werden. Dabei ist die Sprache sehr mächtig und dennoch leicht zu erlernen.

Im Jahre 1991 wurde die Sprache[6] veröffentlicht. Sie wurde im *Centrum voor Wiskunde en Informatica* in Amsterdam entwickelt. Dabei gilt *Guido van Rossum* als Erfinder der Sprache. Python wurde unter anderem von C, LISP und Smalltalk beeinflusst. Wobei Python selbst wiederum Ruby, Boo oder Groovy beeinflusst hat. Dabei bezieht sich der Name auf die Komikergruppe Monty Python und nicht auf die Schlange, obwohl sie zum

[5] Das Python Praxisbuch [1, Kap. 0, S. xviii]
[6] Python 3 Das umfassende Handbuch [2, Kap. 2, S. 23]

Symbol für die Sprache geworden ist.

Heutzutage steht Python unter der PSF-Lizenz und der Code ist im Besitz der *Python Software Foundation*, die sich für die Sprache einsetzt. Die OpenSource PSF-Lizenz[7] ist weniger restriktiv als die GNU/GPL und auch für kommerzielle Zwecke geeignet.

Im Vergleich zu z.B. PHP[8] gibt es auch bei Python viele Webframeworks[9], die aber weniger bekannt sind. Trotzdem wird Python auch im Web erfolgreich eingesetzt unter anderem durch Mithilfe von Django, Pylons oder TurboGears. Mit WSGI ist auch eine robuste und schnelle Schnittstelle entstanden, mit der Python Programme über einen Webserver aufgerufen werden können.

2.3 Unterscheidungsmerkmale von Webframeworks

2.3.1 Full-Stack und Glue

Frameworks werden auf den ersten Blick grob zwischen Full-Stack- und Glue Frameworks unterschieden.

Ein Full-Stack Framework[10] bringt alle nötigen Komponenten mit, um eine Webanwendung schreiben zu können. Die einzelnen Komponenten können Template Engines, OR-Mapper oder eingebaute Testserver sein. Die meisten Frameworks in Python stellen aus verschiedenen bereits existenten Paketen ein Gesamtpaket zusammen, das die Entwicklung von Applikationen für das Web erleichtert. Eine Ausnahme bildet Django, welches unter einer Haube die wichtigsten Teile eines Webframeworks entwickelt hat und sich im Vergleich zu TurboGears in der Hinsicht unterscheidet. Der Vorteil eines Full-Stack Frameworks ist, dass es nach der Installation mit geringer Konfiguration sofort out of the box funktioniert. Allen Gemeinsam ist, dass sie Glue Code benutzen, um die einzelnen Teile miteinander zu verbinden. Bei einigen ist dieser Glue Code etwas lockerer, wie bei Pylons, das keine Vorgaben macht, welche Pakete für die Entwicklung einer Applikation herangenommen werden müssen. Dies bringt Vor- und Nachteile mit sich. Der Vorteil liegt in der Flexibilität und Verwendung der Pakete, die der Entwickler bevorzugt. Nachteilig ist hingegen, dass der Glue Code bestimmte Komponenten nicht

[7] PSF-Lizenz: `http://www.etsimo.uniovi.es/python/psf/license/`, 28.07.2010
[8] PHP Webframeworks: `http://www.phpwact.org/php/mvc_frameworks`, 28.07.2010
[9] Webframeworks für Python: `http://wiki.python.org/moin/WebFrameworks`, 28.07.2010
[10] Full-Stack- und Glue Frameworks: `http://www.b-list.org/weblog/2007/feb/19/python-framework-design/`, 2.12.2010,

vorhersehen kann und der Aufwand das Uhrwerk abzustimmen, mit mehr Konfigurationsaufwand auf der Seite des Entwicklers einhergeht.

So schließen sich Full-Stack und Glue Code in einem Framework[11] nicht unbedingt aus, auch wenn darüber Diskussionen entstanden sind, was das bessere Design ist. Oft wird Django als ein Full-Stack Framework bezeichnet, da es alle nötigen Teile für die Webentwicklung vereint, obwohl es auch möglich ist z.B. SQLAlchemy in das Framework einzubinden, um den eingebauten OR-Mapper zu ersetzen. Andere Webframeworks, wie Pylons oder TurboGears werden als Glue Frameworks beschrieben, weil sie verschiedene nicht interne Pakete zu einer Einheit vereinen. Doch auch hier ist TurboGears ein Full-Stack Framework, da dieser verschiedene aufeinander aufbauende Komponenten für die Entwicklung von Webapplikationen out of the box mitliefert. So sollte sich der Entwickler vor der Auswahl eines Python Webframeworks die Frage stellen, wie viel Flexibilität er benötigt und inwieweit er den Vorgaben vertrauen kann. Meistens kann die Entscheidung erst durch das Ausprobieren der infrage kommenden Frameworks oder den bereits gemachten Erfahrungen mit einzelnen Komponenten getroffen werden.

2.3.2 Funktionalität

Wenn ein neues Produkt das Interesse des Entwicklers geweckt hat, liegt es meistens an der Funktionalität, die es bietet. Für eine Webapplikation ist es wichtig, dass der Client auf die Methoden und Funktionen mittels Browser zugreifen kann, diese ausgeführt werden und eine Antwort zurückgeliefert wird. Als Schnittstelle bietet sich im Python Umfeld WSGI[12] an, damit der Webserver mit der Webapplikation kommunizieren kann, die wiederum HTML an den Client liefert.

Für einfache Anwendungen reicht es aus HTML-Code mit Hilfe von Python oder einer beliebigen Skriptsprache zu generieren und die Ergebnisse anzuzeigen. Soll die Anwendung aber komplexere Ausmaße haben und zum Beispiel eine Userverwaltung mit Login Möglichkeit, Session Handling und ein hohes Maß an Sicherheit gewährleisten, ist es ratsam ein Framework zu benutzen. Vor allem, weil sie ständig weiterentwickelt werden und an heutige Anforderungen für Webapplikationen angepasst sind.

Auch die Kommunikation zu Datenbanken ist ein wichtiger Bestandteil, da viele Webapplikationen Klassen bereitstellen, mittels derer Objekte generiert werden, die mit-

[11] Full-Stack vs. Glue: `http://blog.ianbicking.org/full-stack-vs-glue.html`, 29.07.2011
[12] Überblick WSGI: `http://wsgi.org/wsgi/`, 2.12.2010

einander interagieren. Bei der Interaktion entstehen neue Daten, die verändert und anschließend persistent gehalten werden müssen. Beispiele für Anwendungen, die ständig neue Daten verarbeiten sind z.B. Webshops, Blogs/Wikis oder soziale Netzwerke.

Bei der Entwicklung sollte auch berücksichtigt werden, dass Webdesigner leicht in den Entwicklungsprozess integriert werden können und auch unabhängig von den Entwicklern agieren können. Dabei sollte wichtig sein, dass CSS und JavaScript Code (oder JavaScript-Frameworks), die als Standard bei der Gestaltung von Webseiten gelten, parallel zur Geschäftslogik entwickelt werden. In dem Zusammenhang spielen auch Template Engines für die Erstellung des HTML-Codes eine große Rolle, denn viele Webframeworks bieten Templatesprachen an, die sich z.B. an Python anlehnen aber nur minimale Kontrollstrukturen bieten. Im Grunde sind sie für die Steuerung der Auswahl bei der Erzeugung des HTML-Codes zuständig und fallen ins Arbeitsgebiet der Designer. Durch die Trennung der Arbeitsaufgaben entstehen schnellere und robustere Anwendungen, da die anfallenden Aufgaben so parallel und von einem jeweiligen Experten durchgeführt werden können.

Darüber hinaus existieren Filter, die nach einem Request oder vor einem Response ausgeführt werden und automatisch die Ausgabe Manipulieren indem sie den HTML-Code verändern. Dabei werden dem Code zusätzliche Informationen injiziert, um z.B. die Sicherheit für Formulare zu erhöhen oder die Authentifizierung der Anwender durchzuführen. Diese Klassen werden bei Django Middleware[13] genannt und werden nacheinander vor oder nach einer View ausgeführt.

2.3.3 Dokumentation und Popularität

Damit ein Framework auch benutzt werden kann, bieten viele Projekte eine Fülle an Dokumentation. Von Anfängertutorials bis Modulbeschreibungen werden viele Themen abgedeckt. Auch schreiben viele erfahrene Entwickler für die jeweiligen Frameworks Bücher und tragen so zur Verringerung der Einstiegshürden bei.

Innerhalb kurzer Zeit entstehen Hypes um diese Frameworks. Dabei werden im Web viele Berichte und Beschreibungen rund um die neuesten Technologien geschrieben. Mittlerweile existieren auch Plattformen, die die Beliebtheit verschiedener Webframeworks

[13]Django Middleware: `http://docs.djangoproject.com/en/dev/topics/http/middleware/`, 2.12.2010

nach bestimmten Kriterien[14] auflisten und daraus Ranglisten generieren. Auch eine Auflistung nach Programmiersprache ist möglich. Diese Popularität kann oft dazu beitragen, dass Entwickler neue Funktionen testen und sich überzeugen lassen auf ein anderes Framework umzusteigen.

Diese zwei Faktoren entscheiden oft neben der Funktionalität, inwieweit sich neue Technologien und in diesem Fall Webframeworks durchsetzen können. Deswegen bemühen sich jeweilige Projekte neben der Funktionsvielfalt, die Akzeptanz durch gute Qualität ihrer Dokumentation zu erhöhen.

2.4 Bekannte Webframeworks im Python Umfeld

Zwar werden Webapplikationen sehr oft mit Hilfe von PHP, Ruby oder Java geschrieben und mit Frameworks, die sich schon länger etabliert haben. Doch unter Python ist Webentwicklung leicht möglich. Für Python Entwickler ist das optimal, da keine neue Sprache dafür gelernt werden muss. Die einzige Hürde ist die Einarbeitung in die Zusammenhänge des ausgewählten Frameworks. Die folgend beschriebenen Frameworks[15] sind nur eine kleine Auswahl und sind sicher die bekanntesten. Da eine große Auswahl herrscht, findet sich für jede Projektgröße und Anforderung das passende Framework. Nachfolgend wird die Art und Weise wiedergegeben, wie die jeweiligen Framework Entwickler ihre Produkte auf den Projektseiten anpreisen.

2.4.1 Django

Django[16] ist ein High-Level Python Webframework, das eine schnelle Entwicklung fördert und ein ein sauberes und pragmatischen Design verfolgt.

Django wurde designed, um zwei Herausforderungen zu meistern: Intensive Deadlines eines Newsrooms einzuhalten und den strengen Anforderungen der entwickelten Projekte gerecht zu werden. Django ermöglicht es performante und elegante Anwendungen in kürzester Zeit zu erstellen.

[14] HotFrameworks: `http://hotframeworks.com/languages/python`, 2.12.2010
[15] Überblick verschiedener Python Webframeworks: `http://wiki.python.org/moin/WebFrameworks`, 2.12.2010
[16] Django Project: `http://www.djangoproject.com/`, 2.12.2010

2.4.2 Pylons

Pylons[17] verbindet Ideen aus der Welt von Ruby, Python und Perl. Dabei ist es ein sehr flexibles Python Webframework und verhalf dem WSGI-Standard zum Durchbruch. Das Ziel von Pylons ist es, die Webentwicklung schnell, flexibel und einfach zu gestalten.

2.4.3 Turbo Gears

Aufbauend auf der Erfahrung von *next generation* Webframeworks darunter natürlich TurboGears 1, Django und Rails, ist TurboGears 2[18] entstanden. All diese Framweworks, auf die Turbogears 2 aufbaut, haben ihre Grenzen, die auf verschiedene Weisen frustrierend sind, TG2 ist die Antwort auf diesen Frust.

2.4.4 Zope

Die Zope[19] Softwarebibliothek ermöglicht die komponentenbasierte Entwicklung von Webanwendungen in der objektorientierten Programmiersprache Python. In der komponentenbasierten Programmierung werden komplexe Anwendungen mit Hilfe wiederverwendbarer Komponenten erstellt. Eine Komponente stellt dabei die Implementierung einer bestimmten genau spezifizierten Funktionalität dar.

2.4.5 web2py

web2py[20] wurde inspiriert von Ruby on Rails und, wie Rails, liegt der Fokus in der schnellen Webentwicklung und folgt dem Model-View-Controller Muster. web2py unterscheidet sich von Rails, weil es auf Python basiert. Weil es eine umfassendes Web basierte Administrationsoberfläche bereitstellt und darüber hinaus beinhaltet es Bibliotheken, um mit mehr Protokollen zurechtzukommen. Zusätzlich ist web2py unter der Google App Engine lauffähig.

[17] Pylons: `http://pylonshq.com/`, 2.12.2010
[18] TurboGears 2: `http://turbogears.org/2.1/`, 2.12.2010
[19] Zope `http://www.zope.org/WhatIsZope`, 2.12.2010
[20] web2py: `http://web2py.com/examples/default/what`, 2.11.2010

2.5 Webframeworks anderer Programmiersprachen

Zur Erstellung von Webapplikationen gibt es viele Möglichkeiten. Der einfachste Ansatz ist, eine Programmiersprache zu lernen, mit deren Hilfe HTML-Code erzeugt und dem Client zur Verfügung gestellt wird.

Der erste Kontakt mit der Webentwicklung wird sicherlich mittels PHP[21] hergestellt. Da die Sprache leicht zu erlernen ist und eine Vielzahl von Beispielen im Web zu finden sind. Darüber hinaus existieren eine Menge Frameworks, die den Entwickler bei der Arbeit unterstützen. Unter anderem Zend, CakePHP oder Symphony.

Andere Programmiersprachen, wie Ruby, Java oder Perl ermöglichen ebenfalls Programme fürs Web, auch mit Hilfe von Frameworks, zu schreiben und einem breiten Publikum zur Verfügung zu stellen. In diesen Sprachen sind die bekanntesten Webframeworks sicherlich Ruby on Rails, Spring oder Jifty. Zwar liegt der Fokus dieser Arbeit auf Python, dennoch ist es wichtig andere Möglichkeiten der Webentwicklung auf anderen Plattformen vorzustellen, um den Überblick über Webframeworks abzurunden. Diese werden auf ihren jeweiligen Projektseiten folgendermaßen beworben.

2.5.1 Zend

Das Zend Framework[22] erweitert die Kunst und den Geist von PHP, dabei basiert es auf Einfachheit, beste praktische Vorgehensweise, freundliche Lizenzierung für Firmen und einer gründlich getesteten Codebasis. Dabei liegt der Fokus des Zend Frameworks auf sicherere und moderne Web 2.0 Anwendungen und Webservices.

2.5.2 Ruby on Rails

Ruby on Rails[23] (RoR) ist ein OpenSource Webframework, optimiert um den Programmierer zufrieden zu stellen und ihn bei seiner Kreativität zu unterstützen. Damit lässt es sich wunderbaren Code schreiben, indem es *Convention over Configuration*[24] bevorzugt.

[21] What is PHP: `http://www.php.net/`, 2.12.2010
[22] Zend Framework: `http://framework.zend.com/about/overview`, 2.12.2010
[23] Rubo on Rails `http://rubyonrails.org/`, 2.12.2010
[24] CoC: Standardeinstellungen und Vereinbarungen , an die sich der Entwickler halten soll, Agile Web Development with Rails [7, Kap. 1, S. 17]

> Ruby on Rails is a breakthrough in lowering the barriers of entry to programming. Powerful web applications that formerly might have taken weeks or months to develop can be produced in a matter of days[25].

2.5.3 Catalyst

Das Catalyst[26] ist ein in Perl geschriebenes MVC Webframework. Dieses Framework kann auf verschiedenen Plattformen installiert werden und ist für schnelle Webentwicklung geeignet.

Bei der Entwicklung wird Bedacht auf KISS[27] gelegt, was dazu führt, dass Catalyst skalierbar und robust bleibt. Was sich bei den daraus entwickelten Applikationen bemerkbar macht.

2.5.4 Wicket

Mit zweckmäßiger Aufteilung von Darstellung und Logik, ein POJO[28] Datenmodell und erfrischendem Verzicht von XML, macht Apache Wicket das Entwickeln von Webapplikationen wieder einfach und angenehm. So bricht Wicket[29] mit unnötigem und unsicherem Code und komplexem Debugging, um wiederverwendbare Komponenten zu erstellen, die nur auf Java und HTML basieren.

2.6 Arbeiten ohne Framework

Die Erstellung einer Anwendung ohne ein entsprechendes Framework ist natürlich möglich, doch oft sehr mühsam. Der Entwickler hat zwar alle Freiheiten eine Anwendung zu schreiben, die seine ausgewählte Programmiersprache bietet. Dennoch erleichtert ein Framework viele Aufgaben.

Webframeworks bringen Unterstützung in Aspekten, wie z.B. Sicherheit, Benutzerverwaltung und Applikationsadministration. Diese müssen, falls benötigt von Grund auf

[25] Tim O'Reilly, Founder of O'Reilly Media: `http://rubyonrails.org/quotes`, 2.12.2010
[26] Catalyst: `http://www.catalystframework.org/`, 2.12.2010
[27] KISS: Keep it simple stupid `http://techcrunch.com/2009/04/28/keep-it-simple-stupid/`, 2.12.2010
[28] POJO: Plain Old Java Object `http://www.martinfowler.com/bliki/POJO.html`, 2.12.2010
[29] Wicket: `http://wicket.apache.org/`, 2.12.2010

immer wieder neu geschrieben werden und lenken somit von der eigentlichen Entwicklung der Kernapplikation ab.

Aus der Erfahrung heraus, können anfangs kleine Anwendungen schnell größere Ausmaße bekommen. Ohne eine vorgegebene Struktur kann es schwierig werden Fehler aufzuspüren und zu beheben. Änderungen und neue Funktionalitäten können nur langsam implementiert werden, da an verschiedenen Stellen für einen reibungslosen Ablauf gesorgt werden muss. Einer der größten Nachteile ist, dass ohne Struktur Entwickler sich nur schwer oder langsam in die Zusammenhänge fremder Software einarbeiten können. Das ist aber von Nachteil bei Projekten, an denen mehrere Entwickler arbeiten. So ist es ratsam Webapplikationen mittels Webframeworks zu entwickeln, obwohl teilweise Flexibilität bei der Entwicklung verloren gehen kann.

Kapitel 3.0

Django

3.1 Einführung

Ursprünglich entstanden, um eine Nachrichtenplattform[1] zu betreiben, ist Django speziell dafür entwickelt worden, strikte Deadlines einzuhalten. So fördert Django die schnelle Entwicklung von Applikationen und nimmt dem Softwareentwickler vieles ab, damit er sich auf die Kernimplementierung konzentrieren kann.

Das Framework ist nach dem Jazz Gitarristen *Django Reinhardt*[2] benannt, der zwischen 1930 und 1950 bekannt wurde.

Unter der BSD-Lizenz vertrieben, kann der Django Quellcode proprietär weiterentwickelt beziehungsweise dürfen darauf aufbauend kommerzielle Projekte betrieben werden. Die Applikation in dieser Arbeit *MikroAdmin* wird unter der aktuellen Version 1.2.1[3] entwickelt.

3.2 Dokumentation

Da Django in vielen Projekten eingesetzt wird, gibt es auch entsprechende Anlaufstellen und Bücher, die erklären, wie Projekte unter Django angelegt werden können. Die erste Anlaufstelle ist sicherlich die offizielle Seite *djangoproject.com*, hier findet man alles nötige[4], um ein neues Projekt zu erstellen. Ein Anfängertutorial[5], das in vier Teile gegliedert ist, erleichtert neuen Benutzern den Einstieg. Darüber hinaus gibt es genügend Literatur zum Framework selbst. Folgende Aufzählung ist nur eine kleine Auswahl, was an Literatur zu Django angeboten wird.

- Definitive Guide to Django[6] ein guter Einstieg, wobei nur die Versionen 0.96 und 1.0 beschrieben werden
- Django 1.0 Website Development[7]

[1] Djangos ursprüngliche Plattform: `http://www2.ljworld.com/`, 3.12.2010
[2] Django Reinhardt: `http://www.spiegel.de/kultur/musik/0,1518,673164,00.html`, 3.12.2010
[3] Django Version 1.2.1, Stand 24.05.2010
[4] Dokumentation: `http://docs.djangoproject.com/en/1.2/`, 3.12.2010
[5] Tutorial: `http://docs.djangoproject.com/en/1.2/intro/tutorial01/`, 3.12.2010
[6] The Definitive Guide: [3]: `http://djangobook.com/`, 3.12.2010
[7] Django 1.0 Website Development [4]

- Python Web Development with Django[8]

Des Weiteren existieren Tutorials[9] und Howtos im Web, die viele Themen abdecken.

3.3 Installation

Auf dem Applikationsserver *Debian Lenny* kann zwar über die Paketverwaltung Django installiert werden, dies liegt aber in der älteren Version 1.0.2[10] vor. Um auf einen besseren Vergleich zwischen den Frameworks und den aktuellen Stand Bezug zu nehmen, werden so weit möglich, die neuesten Versionen, der jeweiligen Frameworks eingesetzt. Die aktuelle Version kann - distributionsunabhängig - auf der Django-Seite[11] heruntergeladen werden. Mit folgenden Befehlen kann die Version 1.2.1 unter Debian Lenny installiert werden.

```
1 $ wget -t 45 http://www.djangoproject.com/download/1.2.1/tarball/
2 $ tar -xvf Django-1.2.1.tar.gz
3 $ cd Django-1.2.1
4 $ sudo python setup.py install
```

Listing 3.1: Django Installation

Dabei sollte der Benutzer, unter dem Django installiert wird, auf dem Server in die *suodoers* Liste unter */etc/sudoers* aufgenommen werden, um den Befehl *sudo* ausführen zu können. Gegebenenfalls kann auch unter dem *root*-Account, wie in Linux üblich, die Installation vorgenommen werden.

Die aktuellste Entwicklerversion kann auch über SVN bezogen werden.

```
1 $ svn co http://code.djangoproject.com/svn/django/trunk/
```

Listing 3.2: Django Entwicklerversion

Um festzustellen, ob die Installation erfolgreich war, kann mittels Python-Konsole der Import von Django getestet und z.B. die Version ausgegeben werden.

[8] Python Web Development with Django [5]
[9] Weitere Tutorials: `http://code.djangoproject.com/wiki/Tutorials`, 3.12.2010
[10] Debian Django Paket: `http://packages.debian.org/search?searchon=names&keywords=python-django`, 3.12.2010
[11] Django Project: `http://www.djangoproject.com/download/`, 3.12.2010

```
user@server:~$python
Python 2.5.2 (r252:60911, Jan 24 2010, 14:53:14)
[GCC 4.3.2] on linux2 Type "help", "copyright", "credits" or "license" for more
    information.
>>> import django
>>> print django.VERSION
(1, 2, 1, 'final', 0)
```

Listing 3.3: Testen der erfolgreichen Django Installation

Wenn die Versionsnummer erscheint, ist der Import erfolgt und Django kann eingesetzt werden. Die Python Konsole kann wieder mit *[STRG+D]* geschlossen werden.

3.4 Erste Schritte

Um ein neues Projekt zu erzeugen, kann man auf Skripte zurückgreifen, die das Framework mitliefert. Dafür wird ein Ordner erzeugt, in dem das Projekt erstellt werden soll, und mit folgender Anweisung

```
$ django-admin.py startproject mikroadmin
```

Listing 3.4: Generierung eines neuen Projekts

wird das Projekt generiert.

Dabei ist zu beachten, dass dem Skript ein Name für die zu erstellende Applikation übergeben werden muss.

Bei der Generierung des Projekts werden folgende Dateien im Verzeichnis erzeugt

```
mikroadmin/
__init__.py
manage.py
settings.py
urls.py
```

Listing 3.5: Mikroadmin Dateistruktur

- Mit **__init__.py** wird Python mitgeteilt, dass das Verzeichnis als Package[12] angesehen werden soll.

[12]Python Module: `http://docs.python.org/tutorial/modules.html`, 3.12.2010

- **manage.py** ist ein Kommandozeilentool mit dessen Hilfe weitere Funktionalität im Projekt zur Verfügung gestellt wird. Unter anderen das Starten eines internen Testservers
- In der **settings.py**-Datei befinden sich die möglichen Einstellungen, die das Projekt betreffen. Hier kann z.B. auch die zu benutzende Datenbank definiert werden
- Die Datei **urls.py** ist für die Generierung der URLs und die Verbindung der Views mit den URLs des Projektes zuständig. Bei der Gestaltung der URLs hat man freie Hand und ist an keine bestimmte Konvention gebunden

Um zu überprüfen, ob die neu erstellte Applikation ordnungsgemäß funktioniert, kann folgender Code in der Shell eingegeben werden. Dabei sollte ins Hauptverzeichnis des Applikationsverzeichnis gewechselt werden, um *manage.py* ausführen zu können.

```
1 $ cd mikroadmin
2 $ python manage.py runserver [servername|serverip][:port]
```

Listing 3.6: Ausführen des internen Testservers

Dem Skript können weitere optionale Anweisungen übergeben werden, wie ein Servername oder zusätzlich ein Port, auf dem die Applikation Anfragen entgegen nehmen soll. Standardmäßig wird dieser auf 8000 eingestellt, wenn nichts übergeben und gleichzeitig die Standard-IP 127.0.0.1 für den Localhost benutzt wird. Um Probleme zu vermeiden sollte drauf geachtet werden, dass auf dem gewählten Port nicht schon ein anderer Server wie Apache läuft. Wenn die Konfiguration klappt, wird nach Eingabe von `127.0.0.1:8000` folgende Ausgabe in Abbildung 3.1 wird im Browser angezeigt.

Abbildung 3.1: Django: Erfolgsmeldung

3.5 Datenbanksysteme

Django unterstützt unter anderen folgende Datenbanksysteme.

- Oracle
- PostgreSQL
- MySQL
- SQLite

Es gibt auch die Möglichkeit NoSQL Datenbanken unter Django zu betreiben z.B. CouchDB[13] als dokumentenorientierte Datenbank. Der Connector kann über Github[14] bezogen werden.

Die standardmäßig unterstützten Datenbanken sind mehr als ausreichend, um Webanwendungen betreiben zu können. Ohne jegliche Installation kann SQLite sofort genutzt werden, da Django und Python alles Nötige für den Betrieb liefern. Diese Datenbank wird auch für die Mikroadmin-Applikation benutzt. Die einfache Inbetriebnahme und der Einsatz des in Django mitgelieferten OR-Mappers[15] senken die Einstiegshürden für die Arbeit mit Datenbankabfragen in Django.

[13]CouchDB: `http://couchdb.apache.org/`, 3.12.2010
[14]CouchDB Django Connector: `http://github.com/42/42-django-couchdb`, 3.12.2010
[15]Django QuerySets: `http://docs.djangoproject.com/en/1.2/ref/models/querysets/`, 3.12.2010

In der *settings.py*-Datei wird die passende Schnittstelle für den Einsatz von SQLite ausgesucht. Dabei müssen folgende Anpassungen vorgenommen werden.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '/path/databasename.db',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    }
}
```

Listing 3.7: Einstellungen für den Betrieb von SQLite

Im Grunde muss das richtige Backend ausgewählt, hier *SQLite3*, und zusätzlich der Pfad angegeben werden, wo die Daten abzuspeichern sind. Diese Datenbank muss nicht bereits vorhanden sein, da nach der Datenbanksynchronisation diese automatisch angelegt wird. Im Gegensatz zu anderen Datenbanksystemen ist hier kein User oder Passwort nötig, da diese Funktionalität von SQLite nicht unterstützt wird.

Die Datenbankdatei darf aus Sicherheitsgründen nicht im Apache Webverzeichnis liegen und nur über Django oder SQLite3 ansprechbar sein. Damit auch keine anderen Linux-User auf dem Server die Datei auslesen können, müssen die Rechte der Datei mittels *chmod 750 datenbank.db*[16] so eingestellt sein, dass nur der User, der die Datei erstellt hat, lesend und schreibend drauf zugreifen darf. Da die Anwendung im späterem Verlauf unter Apache und seinem entsprechendem User unter Debian *www-data* ausgeführt wird, ist zwingend erforderlich diesen User in die Gruppe des Users, unter der die Anwendung erstellt wurde, mittels *adduser*[17] aufzunehmen. Somit ist gewährleistet, dass *www-data* auf die Datenbank lesender und schreibender Zugriff erlaubt ist.

Für die Erstellung der Datenbank und allen Datenbanktabellen steht folgender Befehl zur Verfügung.

```
$ python manage.py syncdb
```

Listing 3.8: Datenbanksynchronisation mit Django

[16] chmod: `http://www.linuxforums.org/articles/file-permissions_94.html`, 3.12.2010
[17] adduser: `http://www.cyberciti.biz/faq/howto-linux-add-user-to-group/`, 3.12.2010

Es werden Tabellen generiert, abhängig von den installierten Apps[18], die in der *settings.py* einzutragen sind. Diese Apps[19], die im Paket *django.contrib* liegen, sind optional und können bei Bedarf eingebunden werden. Wie z.B. das *Django-authentication-framework* oder *Djangos Admin-Site*. Unter INSTALLED_APPS können diese eingebunden oder wieder ausgetragen werden. In Django werden standardmäßig Applikationen[20] für Authentifizierung, Session Handling, ein Framework für Content Types und ein Framework zur Bereitstellung mehrerer Applikationen bei der Installation von Django mitgeliefert. Diese bilden eine gute Basis für die Konfiguration von Django-Applikationen.

Zusätzlich wird nach Modellen innerhalb einer selbst erstellten Applikation in *models.py* gesucht, die für die Kernanwendung nötig sind. Aus den Klassenmodellen wird entsprechend über den verwendeten OR-Mapper Tabellen in der Datenbank erzeugt. In dieser Arbeit wird die Applikation mit beinhaltenden Modellen *appMikroadmin* genannt. Diese wird, wie auch andere Apps, in die Konfigurationsdatei eingetragen. Wie Applikationen generiert werden, wird im Listing 3.10 auf Seite 40 erklärt.

A.2

```
1 INSTALLED_APPS = (
2     'django.contrib.auth',
3     'django.contrib.contenttypes',
4     'django.contrib.sessions',
5     'django.contrib.sites',
6     'django.contrib.messages',
7     'django.contrib.admin',
8     'mikroadmin.appMikroadmin',
9 )
```

Listing 3.9: Apps zur Synchronisation mit der Datenbank

- django.contrib.auth – Dient der Userauthentifizierung
- django.contrib.contenttypes – Abstrakte, generische Schnittstelle, um mit Modellen z.B. Datenbankmodellen zu arbeiten
- django.contrib.sessions – Ein Session Framework

[18]Contrib Packages: `http://docs.djangoproject.com/en/dev/ref/contrib/`, 3.12.2010
[19]Unter Django werden selbst erstellte Applikationen oder installierbare Pakete Apps genannt
[20]Standardmäßig installierte Applikationen: `http://docs.djangoproject.com/en/dev/intro/tutorial01/#database-setup`, 3.12.2010

- django.contrib.sites – Framework welches erlaubt verschiedene Seiten mit einer einzigen Django Installation zu verwalten
- mikroadmin.appMikroadmin – Die eigentliche Kernapplikation des Projektes

Es ist nicht nötig SQLite zusätzlich zu installieren, da die Kommunikation mit der SQLite3-Datei über python-pysqlite2[21] im Django DB-Backend geschieht. Falls Änderungen in der Datenbankdatei über die Shell nötig sind, wird z.B. die Amalgation[22] Version von SQLite für Linux benötigt und muss zusätzlich installiert werden. Zwar ist SQLite2 bereits unter Debian Lenny installiert, leider kann mit dieser Version kein Zugriff auf die SQLite3-Datei erfolgen.

Sollen weitere Applikationen[23] installiert werden, müssen diese in *settings.py* hinzugefügt und nochmals das DB-Synctool ausgeführt werden. Leider werden keine Änderungen an Klassenmodellen automatisch in die Datenbank überführt und auch hier ist eine Synchronisation mit der Datenbank nötig, dabei müssen die entsprechenden Tabellen in der Datenbank gelöscht werden. Dies führt zu Verlusten im Datenbestand. Diese sollten vorher gesichert und später wieder in die Datenbank eingespielt werden. SQLite3 bietet keine Möglichkeit im Nachhinein Constraints[24] in der Datenbank direkt zu ändern. Deswegen sollte vorher genau überlegt werden, welche Constraints nötig sind. Ansonsten besteht die Möglichkeit, die Tabelle zu löschen, Änderungen an der Klasse vorzunehmen und die Datenbank wie bereits erwähnt, neu zu synchronisieren.

3.6 Erzeugung der Applikation

Nach erfolgreicher Installation und reibungsloser Kommunikation mit der Datenbank kann die Erstellung der Applikation erfolgen. Die eigentliche Applikation kann unter Django ausgelagert werden. Dieser Schritt ist zwar nicht nötig, wird aber bevorzugt, wenn Teile einer Webanwendung in anderen Projekten eingebunden werden sollen. Zwar ist die zu entwickelnde Applikation eng mit dem bereits erstellten Projekt verbunden, doch ein Auslagern bietet Vorteile. Durch die Auslagerung als Applikation innerhalb des Projekts kann diese in einem späteren Verlauf leicht erweitert oder ausgetauscht werden.

[21] pysqlite2: `http://packages.debian.org/de/sid/python-pysqlite2`, 3.12.2010
[22] SQLite-Amalgation: `http://www.sqlite.org/download.html`, 3.12.2010
[23] Contrib: `http://docs.djangoproject.com/en/dev/ref/contrib/`, 3.12.2010
[24] SQLite3 und Constraints: `http://www.sqlite.org/omitted.html`, 3.12.2010

Voraussetzung, um Funktionen abzurufen ist, dass die Namen der URLS unverändert bleiben. So müssen im XHTML-Code Links nicht neu angepasst werden und können so unangetastet bleiben.

Durch diese lose Kopplung zwischen URL-Dispatcher 3.8 und der View ist die Einbindung oder der Austausch von Applikationen sehr einfach gestaltet. Einstellungen, die nur die Unterapplikation betreffen, können im entsprechenden Verzeichnis beziehungsweise in der entsprechenden Datei geändert werden, so ist die Wahrscheinlichkeit gering, andere Teile des Projektes bei Fehleinstellungen in Mitleidenschaft zu ziehen.

Fehler können auch leichter aufgespürt werden, da die Komplexität durch die Modularisierung verringert wird. Der Developer-Modus, sollte während der Entwicklungsphase oder bei Problemen aktiviert bleiben. Dieser liefert wegen ausführlichem Backtracking gute Anhaltspunkte, um festzustellen, an welcher Stelle Probleme auftauchen.

Um eine Applikation im Projektverzeichnis zu erstellen, kann das *manage*-Tool verwendet werden.

```
1 $ python manage.py startapp appMikroadmin
```

Listing 3.10: Erstellung einer Applikation unter Django

Im Projekt wird ein Verzeichnis mit dem gleichen Namen wie die neu zu erstellende Applikation erzeugt und folgende Dateien hinzugefügt:

```
1 appMikroadmin/
2     __init__.py
3     models.py
4     tests.py
5     views.py
```

Listing 3.11: Dateien nach Erstellung einer neuen Applikation

In der *models.py*-Datei werden die Klassen angelegt, die für die Arbeit mit der Applikation und der Datenbank nötig sind. Diese werden wiederum mit der Datenbank synchronisiert. Diese Modelle sind einfache Python-Klassen, die von der Klasse *from django.db import models* Eigenschaften erben. Dabei repräsentiert jedes Klassenattribut auch ein Datenbankattribut in der Datenbank selbst.

Die ganze Funktionalität der Applikation wird in der *views.py*-Datei erstellt. Hier liegen die Funktionen, die vom Browser aus, über eine URL ansprechbar sind. Auch weitere Hilfsfunktionen befinden sich in dieser Datei, die jedoch nicht vom Benutzer direkt über

den Browser aufrufbar sind. Diese können aber wie in Python üblich, von anderen Funktionen innerhalb der gleichen Datei oder von anderen Funktionen in anderen Paketen benutzt und aufgerufen werden.

3.7 Definition von Modellen und Zugriff auf Modellinstanzen

In Django können Modelle als Klassen definiert werden. Die mit Hilfe der Klassen erstellten Objekte können wiederum in der Datenbank, falls benötigt, persistent gehalten werden. Bei der Beispielanwendung sind die Kernobjekte die Benutzer der Applikation und die zu reservierenden Boards. Die Interaktion bildet die Kernfunktionalität der Webschnittstelle mit den Mikrocontrollern und dem Betriebssystem.

Das Framework bietet bereits ein Klassenmodell für Benutzer und dieses Modell wird, falls die entsprechende App aktiviert wurde, durch den Synchronisationsbefehl in die Datenbank überführt. Alle Benutzer der Applikation werden von *from django.contrib.auth.models import User* abgeleitet, die alle nötigen Attribute und Methoden für eine Userverwaltung mitbringen. Unter anderem Methoden für das Login/Logout, das Setzen von verschlüsselten Passwörtern und der Zugriffsrechte für z.B. den Adminbereich. User werden in Django folgendermaßen generiert.

```
1 from django.contrib.auth.models import User
2 user = User.objects.create_user('username', 'user@email.com', 'passwort')
```

Listing 3.12: Erstellung neuer User

Ab diesem Zeitpunkt kann ein Zugriff auf den User erfolgen und es wird ein Datensatz in der Datenbank erstellt. Werden weitere Veränderungen nötig, wie z.B. die Erweiterung von Nutzerrechten oder der Emailadresse, muss der User in der Datenbank identifiziert werden und nach Änderungen die Methode *save()* für die Speicherung der Daten aufgerufen werden.

```
1 from django.contrib.auth.models import User
2 user = User.objects.get(name='username')
3 user.email = 'neuerUser@email.com'
4 user.is_superuser = 'True'
5 user.save()
```

Listing 3.13: Objekte verändern und abspeichern

Alle möglichen Attribute und Methoden für die Verwendung der Django User Klasse werden unter *User authentication in Django*[25] aufgeführt.

Weiterhin wird eine Klasse benötigt, welche die Mikrocontroller für deren Verwaltung repräsentiert. Klassenmodelle werden bei Django unter einer neu erzeugten Applikation in der Datei *models.py* abgelegt. Um Django Modelle erstellen zu können, ist es nötig von der Klasse *from django.db import models* abzuleiten. Hier die komplette Klassendefinition für die Mikrocontroller der Mikroadmin-Applikation.

C.3

```
1
2  from django.db import models
3  from django.contrib.auth.models import User
4
5  # Create your models here.
6  class Board(models.Model):
7      name = models.CharField(max_length=30)
8      description = models.CharField(max_length=30, blank=True, null=True)
9      linuxUsername = models.CharField(max_length=30, blank=True, null=True)
10     image = models.CharField(max_length=30, blank=True, null=True)
11     currentPasswort = models.CharField(max_length=30, blank=True, null=True)
12     isReservedBy = models.ForeignKey(User, blank=True, null=True)
13     reservedDate = models.DateTimeField(blank=True, null=True)
14
15     def __unicode__(self):
16         return self.name
```

Listing 3.14: Klassenmodell für Mikrocontroller

Wichtig hierbei ist das Zusammenspiel zwischen den Klassen Board und User zu beachten. Dabei wird im Feld *isReservedBy* eine Fremdschlüsselbeziehung zwischen Board und User hergestellt. Dies ermöglicht die Sicherstellung, dass bei einer Reservierung durch einen angemeldeten Benutzer dieser einer autorisierten Userklasse entspringt. Um festzustellen, dass das Board momentan nicht reserviert ist, wird in den Constraints zugelassen, dass das Feld *Null* sein darf. Wichtig hierbei ist zu beachten, dass bei der Zuweisung im Pythoncode das Feld, falls dieses leer bleiben sein soll, mittels dem Keyword None als frei markiert werden muss. Also nicht wie in SQL üblich *Null* zugewiesen wird. Sonst kann es bei Abfragen zu Fehlern kommen.

[25] User Auth: `http://docs.djangoproject.com/en/1.2/topics/auth/`, 3.12.2010

```
aktuellesBoard = Board.objects.get(name='AVR TOOL ATSTK600')
aktuellesBoard.isReservedBy = None
aktuellesBoard.save()
```

Listing 3.15: Markierung leerer Felder

Alle anderen Felder sind im Grunde selbsterklärend und sind dazu da Informationen über das Board an den User zu liefern. Unter anderem das aktuelle Passwort, falls das Board reserviert wurde, auf der Weboberfläche anzuzeigen. Unter *Description* wird ein Link festgehalten, der nähere Informationen über das Board im Allgemeinen liefern soll. Um den Namen des Mikrocontrollers vom Linuxusernamen zu entkoppeln, wird das Feld LinuxUsername benötigt. So soll vermieden werden, dass beim Login mittels SSH Leerzeichen und Umlaute eingegeben werden müssen, die Bezeichnungen von Mikrocontrollern beinhalten können.

Mit Hilfe des OR-Mappers ist es mit wenig Aufwand und ohne SQL-Kenntnisse möglich, verschiedene Operationen über Datensätze auszuführen. Dieser Ansatz ist *pythonischer*[26] als SQL-Statements zu schreiben und beugt zusätzlich z.B. SQL-Injection Angriffen vor, da Variablen vor dem Aufruf der Datenbank gefiltert werden. Um z.B. alle Mikrocontroller innerhalb der Anwendung auszuwählen, wird die Methode *all()* aus den öffentlichen Methoden der *QuerySet()*-Klasse benötigt. Diese und andere Methoden für das Ausführen von Anfragen an die Datenbank befinden sich unter *django.db.models.query*.

Die Verbindung zwischen der Datenbank und den Modellen wird durch die Manager-Klasse in *django.import.models.manager.Manager* durchgeführt. Dabei wird für jedes Klassenmodell eine Manager Instanz erzeugt. Die Manager Instanz wird mittels *objects()* aufgerufen. Dieser Name[27] kann für jedes Modell in der Datei *models.py* nach Belieben geändert werden. Hier ein paar Beispiele, um Anfragen an die Datenbank zu schicken.

```
from appMikroadmin.models import Board
# Alle Boards auswaehlen und die jeweiligen Namen ausgeben
boards = Board.objects.all()

# Hier kann .name auch weglassen werden, da in der Klassendefinition
# beim Aufruf eines Boards automatisch
# der Name zurueckgeliefert wird
for board in boards:
    print board.name
```

[26] Der Pythonische Weg: `http://www.oreilly.de/artikel/2009/10/visionaere.html`, 3.12.2010
[27] Manager-Name: `http://docs.djangoproject.com/en/dev/topics/db/managers/`, 3.12.2010

```
10
11 # Alle reservierten Boards finden:
12 # __isnull verhaelt sich wie die SQL-Abfrage
13 # IS NULL/IS NOT NULL und kann nach False oder True abgefragt werden
14 boards = Board.objects.filter(isReservedBy__isnull=False)
```

Listing 3.16: Datenbankabfragen mittels OR-Mapper

Weitere wichtige Methoden zur Kommunikation mit der Datenbank und zur Erstellung von Abfragen können unter *Making Queries*[28] eingesehen werden.

Es ist jedoch nötig zu erwähnen, dass der Django OR-Mapper nicht alles abdecken kann, was mit purem SQL möglich ist. Es sind nur ein paar wenige ORMs, die so etwas von sich behaupten können. Dabei deckt Djangos OR-Mapper[29] nicht vollständig die Funktionalität, die RDBMS vieler Datenbanken anbieten. Unter anderem Tabellen-Views, Trigger oder cascadisches Verhalten beim Löschen oder Update von Datensätzen. Dennoch ist es möglich unter der erstellten Applikation z.B. *meinProjekt/meineApp/sql/trigger.sql* Trigger in SQL zu definieren, die jedes Mal bei der Benutzung der Management-Tools *manage.py* z.B. bei der Synchronisation mit der Datenbank ausgeführt werden.

Zusätzlich können Abfragen, die mit dem OR-Mapper nicht oder nur schwer möglich auszuführen sind, anderweitig erstellt werden. Diese Abfragen, die mittels eines Datenbankadapters ermöglicht werden, sind aber datenbankspezifisch und müssen bei der Migration zu einer anderen Datenbank im ungünstigsten Fall umgeschrieben werden. Hier ein Beispiel für eine *low-level-Abfrage*[30] mit Hilfe der Manager-Methode.

```
1 from appMikroadmin.models import Board
2
3 boards = Board.objects.raw(SELECT *
4                 FROM appMikroadmin_board WHERE
5                 isReservedBy_id IS NOT NULL")
6 for board in boards:
7     print board.name
```

Listing 3.17: Ausführen von SQL-Statements mittels raw()-Methode

Hierbei werden RawQuerySet-Instanzen zurückgeliefert, die man wie gewohnt benutzen kann. Manchmal reicht das auch nicht aus, um Statements an die Datenbank zu

[28] Queries: `http://docs.djangoproject.com/en/dev/topics/db/queries/`, 5.12.2010
[29] Python Web Development with Django [5, Kap. 4, S. 112]
[30] Low level Abfragen: `http://docs.djangoproject.com/en/1.2/topics/db/sql/`, 5.12.2010

stellen. So kann mittels der Python DB-API direkt auf Datensätze ohne Wrapper zugegriffen werden.

```
1  from django.db import connection
2
3  cursor =  connection.cursor()
4  # Alle reservierten Boards finden
5  cursor.execute("SELECT *
6                  FROM appMikroadmin_board WHERE
7                  isReservedBy_id IS NOT NULL")
8  reservierteBoards = cursor.fetchall()
9
10 for board in reservierteBoards:
11     print "%s %s %s" % (board[0], board[1], board[2])
```

Listing 3.18: Ausführen von direkten SQL-Statements

Wie beschrieben, ist es möglich die Beschränkungen aufzuheben, die der OR-Mapper mitbringt. Die meisten Webanwendungen können aber ohne Einsatz von Low-Level Abfragen erstellt werden.

Darüber hinaus kann Djangos-ORM gegen einen anderen OR-Mapper, wie das ausgereifte und seit 2006 bestehende SQLAlchemy[31] ausgetauscht[32] werden. Was aber für das MikroAdmin-Projekt nicht nötig wurde.

3.8 URL-Dispatcher

Ein sehr wichtiger Aspekt einer Webanwendung ist die Kommunikation zwischen User und dem Server. Um Befehle abzusenden, gibt es die Möglichkeit, dem Server entweder per GET oder POST diese mitzuteilen. Mittels POST ist es möglich dem Server anzugeben, dass einer Anfrage weitere Informationen zur Ausführung bzw. Verarbeitung angehängt sind. Diese Methode wird vorwiegend bei der Übertragung von Formularen eingesetzt z.B. nach einem Klick auf einen Submit-Button. In einem Datenblock werden somit mehrere zusammenhängende Variablen übermittelt, die eine Funktion bzw. ein Skript verarbeiten sollen. Nach der Ausführung der Funktion bekommt der Aufrufer bei Erfolg ein Feedback standardmäßig in HTML.

[31] SQLAlchemy: `http://www.sqlalchemy.org/`, 5.12.2010
[32] Django-ORM gegen SQLAlchemy austauschen: `http://lethain.com/entry/2008/jul/23/replacing-django-s-orm-with-sqlalchemy/`, 5.12.2010

Ähnlich verhält es sich bei der Übermittlung von Daten über die GET-Methode. Dabei wird eine URL so definiert, dass sie z.B. ein Skript aufruft, und Daten, durch ein *Fragezeichen-?* getrennt, dem Skript mitgeteilt werden. Die Ausführung geschieht hierbei über den Klick auf einen Link. Dabei besteht die vollständige URI aus der URL und den zu übertragenden Daten.

3.8.1 Verbindung zwischen URLs und Views

Mit Django ist es möglich URLs zu definieren, die wiederum mit sogenannten *Views*[33] verknüpft sind. Diese Views verarbeiten wiederum die übertragenen Informationen, die entweder über POST oder GET verschickt werden, können und senden einen HTML-Response zum Aufrufer zurück.

Bei der Gestaltung von URLs gibt es seitens des Frameworks keinerlei Einschränkungen. Wobei URLs durchdacht sein müssen, da sie die Schnittstelle zur Außenwelt sind und im Normalfall nicht mehr geändert werden sollten. Informationen zum URL-Design können beim W3C unter dem Artikel *Cool URIs don't change*[34] bezogen werden.

Unter Django werden keine Dateiendungen, wie *.php* unter PHP, benötigt, um einen Controller zu lokalisieren. Die Bezeichnungen der URLs sollten so weit möglich sprechend sein, damit der User sich diese gut merken kann. In der Datei *urls.py* werden alle nötigen URLs definiert, mit denen der User Verbindung mit dem Server und somit zu den *Views* aufnehmen kann. Jede URL ist dabei mindestens einer *View* zugeordnet. Hier ein Auszug aus der *urls.py* der Mikroadmin-Applikation.

A.3

```
1  from django.conf.urls.defaults import *
2
3  # Uncomment the next two lines to enable the admin:
4  from django.contrib import admin
5  admin.autodiscover()
6
7  urlpatterns = patterns('',
8      (r'^admin/', include(admin.site.urls)),
9      (r'^$', 'mikroadmin.appMikroadmin.views.index'),
10     (r'^login/(?P<loginerror>\w*)$', 'mikroadmin.appMikroadmin.views.index'),
11     (r'^start/$', 'mikroadmin.appMikroadmin.views.start'),
12     (r'^board/(?P<board>\w+)$', 'mikroadmin.appMikroadmin.views.board'),
13     (r'^accounts/login/$', 'mikroadmin.appMikroadmin.views.mylogin'),
```

[33] View: http://docs.djangoproject.com/en/1.2/topics/http/views/, 6.12.2010
[34] Cool Uris don't change: http://www.w3.org/Provider/Style/URI, 6.12.2010

```
14 )
```

Listing 3.19: Auszug aus der urls.py der Mikroadmin-Applikation

Der zentrale Punkt dieser Datei ist die Funktion *patterns()*, diese wird im Paket *django.conf.urls.defaults* zur Verfügung gestellt. Sobald eine Anfrage über eine URL an den Server gestellt wird, wird in der *settings.py*-Datei ermittelt, wie die *ROOT_URLCONF* lautet. Im Beispiel befindet sich diese in *mikroadmin.urls*, also direkt im Projektverzeichnis der Applikation. Daraufhin wird in dieser Datei nach der Variablen *urlpatterns* gesucht, die eine Liste zurückgibt im Format, welches von der Methode *patterns()* zurückgeliefert wird. Django durchläuft alle Einträge der Liste und stoppt, sobald die übermittelte URL mit einem Muster in der Liste übereinstimmt. Der so ermittelten *View* wird ein *HttpRequest*[35] als erstes Argument übermittelt und falls die *View* noch andere benötigt, zusätzliche Argumente aus der URI abgefangen. Daraufhin wird die *View* ausgeführt und nach Beendigung der Ausführung ein *HttpResponse* an den Aufrufer zurückgeschickt.

Somit verknüpft die *patterns*-Funktion alle Verbindungen zwischen URLs und Views. Diese Verbindungen liegen in Tupelform vor, aufgeteilt in einen regulären Ausdruck und den Namen der *View*. Noch vor der Zuweisung einzelner Einträge ist die erste Stelle dafür reserviert, die Bezeichnung der Applikation in folgender Form *appName.view* einzutragen. Diese Stelle wird als View Prefix bezeichnet. So müssen anschließend Views nicht mit vollständigem Namen ausgezeichnet werden. Zu beachten ist, dass den URLs ein *r* vorangestellt wird. Das deutet an, dass der beinhaltende String ein regulärer Ausdruck ist und so gematched wird, wie angegeben und dadurch keine Zeichen *escaped*[36] werden. Die runden Klammern innerhalb des Ausdrucks signalisieren, dass an der Stelle Informationen für die View abgefangen werden.

3.8.2 Möglichkeiten zur Strukturierung

Wird zum Beispiel in der Applikation ein Link namens *http://host/mikroadmin/board/board1/)* angeklickt, wird die board-View ausgeführt und als Argument der Name des Boards, in diesem Fall *board1*, der Variablen *board* zugewiesen. Diese Art

[35] HTTPRequest: `http://www.mediaevent.de/tutorial/http-request.html`, 6.12.2010
[36] Matching Whole Words: `http://diveintopython.org/regular_expressions/street_addresses.html#re.matching.2.3`, 6.12.2010

der Weitergabe mit benannten Variablen wird *Named Groups* genannt. Werden nur die regulären Ausdrücke ohne eine benannte Variable abgefangen, erfolgt die Zuweisung in der View der Reihenfolge nach, wie sie in der URL eingebeben wird. Informationen werden in der URL mittels *(\w+)*, wenn es sich um Strings, oder *(\d+)*, wenn es sich um Zahlen handelt, abgefangen. Dabei kann die Länge der abgefangenen Zeichen mittels *(\d{4})* z.B. auf vier Zeichen begrenzt werden. Wenn kein regulärer Ausdruck mit einer übergebenen URL übereinstimmt, wird eine Fehlermeldung mit dem Statuscode 404[37] zurückgegeben. Wenn zusätzlich während der Entwicklungsphase der Applikation in der Settings-Datei der Variablen *DEBUG* True zugewiesen wurde, werden dem Aufrufer alle regulären Ausdrücke und der Name der mit ihnen verbundenen Views angezeigt. Das Bild 3.2 zeigt eine mögliche Ausgabe, wenn kein Matching erfolgt.

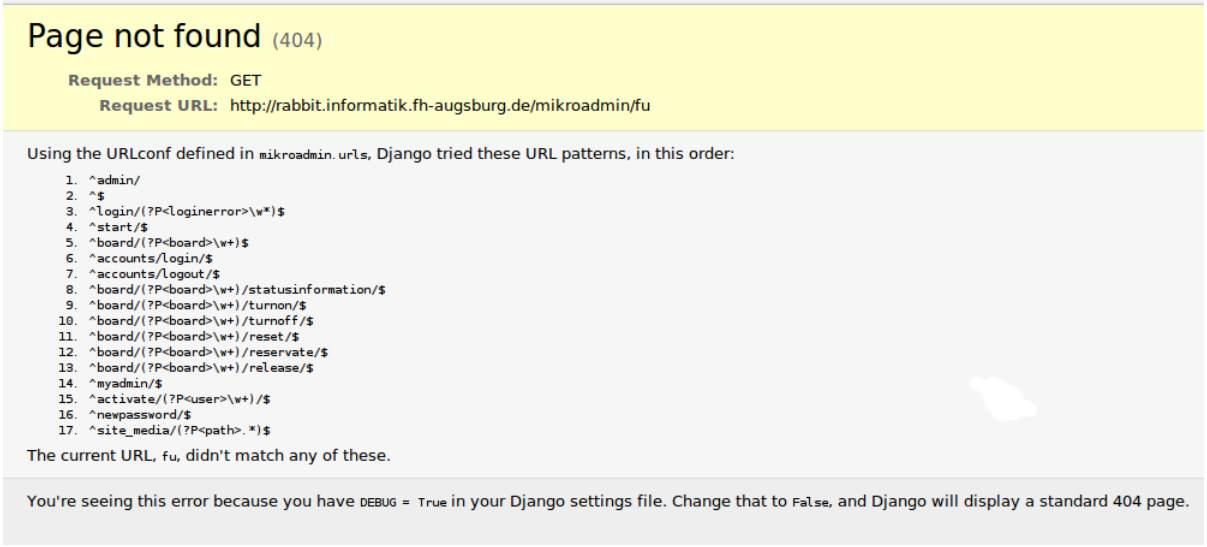

Abbildung 3.2: Django: Status 404 im Debug-Mode

Neben dem View Prefix gibt es weitere Möglichkeiten die regulären Ausdrücke zu strukturieren oder bei Erstellung mehrerer Applikationen in diese auszulagern.

Die erste Möglichkeit nennt sich *Multi View Prefix*. Dabei werden Views nach Applikationen gruppiert und der jeweilige Prefix ins erste Listenfeld eingetragen.

```
1
2 from django.conf.urls.defaults import *
```

[37] Statuscodes: `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`, 6.12.2010

```
3
4  urlpatterns = patterns('mikroadmin.appMikroadmin.views',
5      (r'^$', 'index'),
6      (r'^start/$', 'start'),
7  )
8
9  urlpatterns += patterns('mikroadmin.appBlog.views',
10     (r'^$', 'index'),
11     (r'myblog/(\d+)^$', 'blog'),
12 )
```

Listing 3.20: Multi View Prefix

Hierbei muss dem Applikationsnamen und der View-Datei bei der Auszeichnung am Ende kein Punkt angegeben werden, darum kümmert sich Django. Die Namen der jeweiligen Views können wie gewohnt und ohne Prefix angegeben werden.

Eine andere Art der Strukturierung ist das Auslagern der regulären Ausdrücke in die jeweiligen Applikationen. Dafür wird in jede Applikation eine zusätzliche *urls.py*-Datei erstellt. Auf diese wird bei der Einbindung mittels *include* in der *patterns* Funktion verwiesen.

```
1  from django.conf.urls.defaults import *
2
3  urlpatterns = patterns('',
4      (r'^mymikroadmin/',        include('mikroadmin.appMikroadmin.urls')),
5      (r'^blog/',      include('mikroadmin.appBlog.urls')),
6  )
```

Listing 3.21: Auslagern regulärer Ausdrücke in Applikationen

In den jeweiligen Dateien muss die URL-Patterns-Variable vorhanden sein, die wiederum eine neue Liste zurückliefert. Diese Möglichkeit ist besonders anzuraten, wenn Applikationen erstellt werden sollen, die später unabhängig von der Hauptapplikation lauffähig sein sollen, da der ganze Code dieser Applikation vollständig in ein eigenes Verzeichnis ausgelagert wird.

Manchmal kann es jedoch passieren, dass die URL für eine View geändert werden muss. Wird diese URL aber sehr oft in der Anwendung gebraucht, muss sie in jedem Template, in dem der Link genutzt wird, geändert werden. Dies kann mit Hilfe von *Named URL Patterns* umgangen werden. Dabei wird die *url()*-Funktion benötigt. Diese kann anstelle eines Tupels benutzt werden, um *reguläre Ausdrücke* zu definieren.

```
1 urlpatterns = patterns('',
2     url(r'^index/$', index, name="hauptseite"),
3 )
```

Listing 3.22: Definition regulärer Ausdrücke

Der Zugriff auf die benannten Patterns kann in einem Template, wie im Abschnitt 3.10 beschrieben, dann folgendermaßen erfolgen: {% url hauptseite %}. Dabei können sich sowohl die View oder das Muster ändern, aber der Aufruf im Template bleibt immer derselbe. Dies bedeutet zwar etwas mehr Aufwand, da jedem Pattern eine zusätzliche Bezeichnung angefügt werden muss, aber dadurch bekommt die Anwendung eine höhere Flexibilität und die spätere Anpassung in den Templates ist dann obsolet.

3.8.3 Weitere Einstellungen

Der URL-Dispatcher in Django bietet viele Möglichkeiten, um die tägliche Arbeit zu erleichtern. Darunter das Definieren von *Handlern*, für die Erstellung von 404 bzw. 500 HTTP-Errorpages und zusätzliche *Utility*-Methoden[38]. Dabei dient der Dispatcher als flexibles und zentrales Werkzeug bei der Verbindung der einzelnen Teile einer Django Webapplikation.

3.9 Erstellung der Kernfunktionalität

Im Grunde beinhalten *Views* in Django, den Kern der Webanwendung. Die Bezeichnung für diese Art von Funktionen kann etwas irreführend sein und ungewohnt sein, da im MVC-Pattern, die View der Teilbereich ist, welches die Darstellung der Daten erzeugt. Die Django Entwickler bezeichnen in eigener Definition das Framework als ein MTV[39] Framework und definieren die Arbeit der View um. Die in ihren Augen nicht für die Darstellung der Inhalte zuständig ist, sondern viel mehr dazu da ist, die darzustellenden Daten vor Weitergabe an ein Template zu sammeln und vorzubereiten. In Django können die Views eher der Controllerschicht in anderen Frameworks zugeordnet

[38] Utility Methoden: `http://docs.djangoproject.com/en/1.2/topics/http/urls/#utility-methods`, 6.12.2010

[39] Model Template View: `http://docs.djangoproject.com/en/1.2/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names`, 6.12.2010

werden und bilden die Logik des Programms. Erst in einem nächsten Schritt und zwar bei der Erstellung der Templates, wird HTML an den Server und somit zum Client zurückgesandt. Diese Templates werden von der View als Returnwert geliefert. In anderen Websprachen bzw. anderen Frameworks werden die Views, als Businesslogik bezeichnet. Das ist auch der Ort an dem der Softwareentwickler in Django die meiste Zeit verbringt. Von den Views aus können auch die Modelle angesprochen und somit Einträge in der Datenbank verändert werden.

3.9.1 Views

Um unter Django Webapplikationen zu schreiben, reichen bereits erworbene Pythonkenntnisse vollkommen aus. Gesammeltes Wissen über Konstrukte, wie Abfragen, Schleifen und Funktionen können auch hier eingesetzt werden. Auch Kenntnisse aus anderen Bibliotheken, wie smtp, csv, String, os usw. können angewendet werden, da die ganze Pythonbibliothek für den Einsatz in Views zur Verfügung steht. Dabei sind *Views* ganz normale Python-Funktionen mit dem Zusatz, dass sie als Parameter ein HttpRequest aufnehmen und ein HttpResponse Objekt als Returnwert zurückgeben müssen. Die einfachste View kann folgendermaßen aussehen.

```
1 from django.http import HttpResponse
2
3 def myView(request):
4     return HttpResponse()
```

Listing 3.23: Grundlegender Aufbau einer View

Solange die View ein HttpResponse zurückgibt, sind bei der Beschreibung der Logik nur die Grenzen gesetzt, die Python vorgibt. Das aber mit seine großen Bibliothek viele Bereiche der Programmierung abdeckt.

3.9.2 Generic Views

Ein beliebter Grund, um Frameworks im Allgemeinen zu verwenden und Django im Speziellen, ist die Möglichkeit, vorgefertigte Codebausteine zu verwenden, die Ausgaben an den Browser vereinfachen. Diese Bausteine werden in Django Generic-Views genannt und dienen dazu dem Entwickler immer wiederkehrende Arbeit abzunehmen. Vornehm-

lich bei Operationen, die als CRUD[40] bezeichnet werden. Folgende Generic-Views können unter Django eingesetzt werden.

- Simple Generic Views
- Date Based Generic Views
- List/Detail Generic Views
- CRUD Generic Views

Simple Generic Views unterstützen einfache Aufrufe, wie z.B. die Darstellung eines Templates, in dem statische Informationen angezeigt werden. Oft verwendet in Fällen, bei denen sich die Erstellung einer View nicht lohnt, weil keine Logik vor der Darstellung des Templates zum Einsatz kommt. Darunter fallen z.B. *direct_to_template* und *redirect_to*. Bei beiden Funktionen werden wie gewohnt Regular Expressions definiert, die mit der View verbunden sind. Zusätzlich können weitere Parameter übergeben werden, die den Ort des aufzurufenden Templates oder die Weiterleitung definieren.

```
1 urlpatterns = patterns('django.views.generic.simple',
2     (r'^index/$',             'direct_to_template', {'template': 'index.html'}),
3     (r'^index/(?P<id>\d+)/$', 'direct_to_template', {'template': 'index-detail.html'}),
4
5     ('^alteseite/(?P<id>\d+)/$', 'redirect_to', {'url': '/neueseite/%(id)s/'}),
6     ('^alteseite/(?P<id>\d+)/$', 'redirect_to', {'url': '/neueseite/%(id)s/', '
          permanent': False}),
7     )
```

Listing 3.24: Simple Generic Views

Weiterhin ist es möglich Generic-Views zu benutzen, die Objekte nach bestimmten Zeitangaben geordnet darstellen. Diese Objekte müssen allerdings ein Datumsfeld besitzen, um mit diesen Funktionen aufgelistet zu werden. Diese Auflistung kann z.B. eine Darstellung nach Jahren sein, mit der Unterteilung nach Monaten oder Wochen. Im Normalfall werden dann nur die neuesten Objekte angezeigt. Objekte die laut Datums-Attribut noch in der Zukunft liegen, werden ausgeblendet, es sei denn die Option *allow_future* ist auf *True* gestellt.

[40] Create Read Update Delete: `http://www.postneo.com/2005/08/17/django-generic-views-crud`, 6.12.2010

Ähnlich den Date-Based Generic-Views können List-/Detail Generic Views benutzt werden. Dabei werden Objekte aufgelistet, die aber nicht unbedingt einer bestimmten zeitlichen Reihenfolge entsprechen müssen. Bei der List Generic-View werden Objekte einer bestimmten Klasse, nach Angabe eines Datenbankaufrufs, von der Datenbank zurückgeliefert und angezeigt. Mit der Option *paginate_by* kann angegeben werden, wie viele Objekte pro Seite dargestellt werden sollen. Hier ein Beispiel für die Umsetzung eines einfachen Pagings und der Darstellung von fünf Boards auf einer Seite.

```
1  from polls.models import Board
2
3  # Dictionary üfr die List Detail Generic-View
4  info_dict_list = {
5      'queryset': Board.objects.all(),
6      'paginate_by': 5,
7  }
8
9  # Dictionary üfr die Object Detail Generic-View
10 info_dict_detail = {
11     'queryset': Board.objects.all(),
12     'slug_field':'name',
13 }
14
15 urlpatterns = patterns('',
16     (r'^allboards/(?P<page>[0-9]+)$','django.views.generic.list_detail.object_list',
           info_dict_list),
17     (r'^specificboard/(?P<slug>\w+)/$', 'django.views.generic.list_detail.object_detail
           ', info_dict_detail), )
```

Listing 3.25: List Detail Generic Views

Der Aufruf der einzelnen Seiten geschieht dann mittels */allboards/#* im Browser, wobei # eine Zahl darstellt. Die Variable, welche die Zahlen aufnimmt, muss für die Übertragung unbedingt *page* heißen, da diese für das Paging ausgewertet wird. Wichtig ist, dass die erste Seite nicht wie in der Informatik gewohnt bei 0, sondern bei 1 anfängt. Der Generic-View kann zusätzlich ein Dictionary übergeben werden, z.B. wie das Template heißen soll, das die einzelnen Boards ausgibt. Standardmäßig wird der Name des Objekts mit kleinem Anfangsbuchstaben mit dem Zusatz *_list*.html als Name benutzt. Infolgedessen muss der Name des Templates *board_list.html* heißen, um angesprochen zu werden, sonst wird im Debug-Modus ein Fehler zurückgeworfen. Da der Templateloader unter dem *templates*-Verzeichnis der Applikation nach HTML-Vorlagen sucht, sollte dieser im entsprechenden Verzeichnis *board_list.html* abgelegt sein. Das Templatever-

zeichnis wird in der *settings.py* Datei in der Variable *TEMPLATE_DIRS* hinterlegt. Um die Boards mit Namen auszugeben kann man folgenden Code zur Darstellung nutzen.

```
<h2>Boards</h2>
    <ul>
        {% for board in object_list %}
            <li>{{ board.name }}</li>
        {% endfor %}
    </ul>
```

Listing 3.26: Auflistung von Boards mittels Generic-List-View: board_list.html

Ähnlich verhält es sich bei der Darstellung von einzelnen Objekten mittels *django.views.generic.list_detail.object_detail*. Um im Template das aktuelle Objekt anzusprechen, muss der abgefangene Name in der URL, der *object_id*, also dem Primary Key des Objektes entsprechen. Es gibt zusätzlich die Möglichkeit den *slug*, also das Schlagwort, zu ändern, zu dem in der Datenbank passenden Objekte gefunden werden sollen. Dieses Schlagwort entspricht einem Attribut des Objektes und wird im Dictionary mittels *'slug_field':'name'* definiert. Mit Hilfe der Variable *objekt*, kann das gefundene Objekt, hier das Board, referenziert und seine Eigenschaften im Template ausgegeben werden.

```
<h2>{{ object.name }}</h2>
   <ul>
      <li>{{ object.description }}</li>
      <li>{{ object.image }}</li>
   </ul>
```

Listing 3.27: board_detail.html

Die CRUD Generic-Views[41] arbeiten mit ähnlichen Mechanismen und benutzen zur Darstellung der Objekte die Forms-Library[42] für die einzelnen Objekte. So können mit diesen Views neue Objekte angezeigt, neu erstellt, geändert oder aus der Datenbank gelöscht werden. Was immer wiederkehrende Aufgaben einer Webapplikation automatisieren und die Arbeit des Entwicklers erleichtern kann.

[41] CRUD-Views: `http://docs.djangoproject.com/en/dev/ref/generic-views/#create-update-delete-generic-views`, 6.12.2010

[42] Forms-Library: `http://docs.djangoproject.com/en/dev/topics/forms/`, 6.12.2010

Zwar wurde bei der Arbeit mit Mikroadmin auf Generic-Views verzichtet, da die Erstellung neuer Board-Objekte auch mit der von Django mitgelieferten Adminoberfläche einfach gelingt. Somit wurden keine CRUD-Views für die Erstellung bzw. Veränderung der Daten über Formulare nötig. Für den Umgang mit den Daten und den einzelnen Rechten der angemeldeten User, wurde mehr Logik benötigt, als die Generic-Views liefern konnten und auch über die reine Darstellung von Daten hinaus geht.

3.9.3 Beschreibung der Vorgänge in Mikroadmin

Nach der Anmeldung auf der Mikroadminoberfläche, wird nach erfolgreichem Login zwischen Administrator und normalem User unterschieden. So wird Administratoren ein zusätzlicher Link eingeblendet der Zugang zu weiteren Funktionen gewährt. Darunter die Möglichkeit einzelne bzw. alle reservierten Mikrocontroller freizugeben. Die Entscheidung ob ein Benutzer die zusätzliche Funktionalität zur Verfügung steht, wird innerhalb der aufgerufenen View gefällt. Hierbei erhalten alle Views Zugang zur Userauthentifikations-App, die in der Klasse *django.contrib.auth* jeder View zur Verfügung steht. Wurde ein Administrator akzeptiert, wertet das Template diese Information aus und blendet den Link ein. Als zusätzliche Sicherheitsmaßnahme wird beim Klick auf den Link geprüft, ob es sich um einen autorisierten User handelt.

Die Erstellung einer View unter Django ist wie unter 3.23 gezeigt, recht einfach. Mit Hilfe des *request*-Arguments, der ein Dictionary mit vielen Informationen über den Status der Kommunikation zwischen User und Anwendung mit sich führt, kann auf viele Ereignisse während des Aufrufs einer View, reagiert werden. Während des Betriebs von Mikroadmin seitens des Users ist die Reservierung eines Boards eine zentrale Funktion. Beim Aufruf des entsprechenden Links wird per JavaScript, die entsprechende View *boardReservation(request, board)* aufgerufen. Hierbei wird zuerst überprüft, ob der Mikrocontroller bereits von einem anderen User reserviert worden ist, oder noch für die Reservierung zur Verfügung steht. Dies geschieht durch den Aufruf einer Unterfunktion, die ausgelagert ist, da auch andere Views diese Funktion benötigen. Diese Überprüfung hätte auch mittels Decoratoren[43] gelöst werden können. Da aber die Views bereits mit *@login_required* dekoriert sind, um sicherzustellen dass nur eingeloggte User diese Funktionen ausführen dürfen, ist diese Version mit der Unterfunktion eine gangbare aber nicht

[43]Dekoratoren: `http://uswaretech.com/blog/2009/06/understanding-decorators/`, 6.12.2010

ganz so elegante Lösung.

Als nächstes wird die Boardinstanz abgefragt, die über die URL an die View geschickt worden ist. Falls das Board noch frei ist, wird es reserviert und dem Template mitgeteilt. Während des Reservierungsvorgangs wird eine Funktion aufgerufen, die ein zufälliges Passwort generiert. Mit diesem Passwort ist es dann möglich, dass ein User sich mittels SSH und dem Boardnamen an dem Server, der die Boards verwaltet anzumelden.

Versucht ein dritter User ein bereits reserviertes Board zu übernehmen, wird ihm der Zugang verwehrt. Die Information, ob das Board reserviert worden ist, geschieht mittels der Variablen *info*, die das Template auswertet, wobei das Attribut *isReservedBy* für die Überprüfung abgefragt wird. Stimmt der aktuelle User und der Name des Users, welches das Board reserviert hat überein, stehen ihm alle Funktionen 1.1 für den Umgang mit dem Board zur Verfügung. Zur Veranschaulichung, hier die View, die für die Reservierung der Mikrocontroller zuständig ist.

C.5

```
1  @login_required
2  def boardReservation(request, board):
3      ok = boardReservationStatus(request, board)
4
5      board = Board.objects.get(name=board)
6
7      if ok:
8          if board.isReservedBy is None:
9              size = 8
10             password = generatePassword(size)
11             #password =  ''.join([choice(string.letters + string.digits) for i in range
                   (size)])
12
13             returncode = changePassword(board.name, password)
14
15             board = Board.objects.get(name=board)
16             board.currentPasswort = password
17             board.isReservedBy = request.user
18             board.reservedDate = datetime.now()
19             board.save()
20
21             info = 'reserved'
22         elif board.isReservedBy == request.user :
23             info = 'reserved'
24         else:
25             info = 'free'
26
27         return render_to_response('mikrocontrollertemplate/reservation.html', {'board':
               board, 'request':request, 'info':info})
28     else:
```

```
29          info = 'reserved'
30          return render_to_response('mikrocontrollertemplate/reservation.html', {'board':
                board, 'request':request, 'info':info})
```

Listing 3.28: View zur Reservierung der Mikrocontroller

3.9.4 Besonderheiten

Die Erstellung von *normalen* Views ist etwas einfacher und ein gewohnterer Weg für den Einsteiger. Für den Umgang mit Generic-Views ist auch zusätzliche Einarbeitung nötig, da an manchen Stellen nicht genug Beispiele für den Umgang in der Dokumentation zur Verfügung stehen. Bestimmte Funktionalitäten, die diese Views bieten, müssen an bestimmten Stellen, wie bei der Verschlagwortung in der *Object-Detail*-View[44] (Slug) durch *Try and Error* erst einmal getestet und konfiguriert werden. Gelingt der Umgang mit diesem Konzept, kann sich der Entwickler bei bloßer Darstellung von Objekten viel Programmierarbeit sparen und dem DRY-Prinzip treu bleiben.

Da Views im Grunde genauso geschrieben werden, wie normale Funktionen unter Python, kann der Python-Entwickler ohne vielem Umdenken sofort mit der Erstellung der Kernapplikation beginnen. Dabei ist hervorzuheben, dass Django z.B. bei der Authentifizierung und der Verwaltung von Usern, dem Entwickler viel Arbeit abnimmt und eine bereits funktionsfähige Administrationsoberfläche für die Änderung von Objekten als App beilegt.

3.10 Templatesystem

Das Templatesystem von Django bildet die Schnittstelle zwischen dem User und dem darunterliegendem System. Es dient zur Generierung von HTML und ist somit der Teil, mit dem sich der Webdesigner hauptsächlich beschäftigt. Wie viele Teile in Django ist sein Templatesystem lose gekoppelt und kann durch z.B. andere Templatesysteme wie Smarty[45], Cheetah[46] oder Mako[47] ausgetauscht werden.

[44] Slug: `http://docs.djangoproject.com/en/dev/ref/generic-views/#django-views-generic-date-based-object-detail`, 6.12.2010
[45] Smarty: `http://www.smarty.net/`, 6.12.2010
[46] Cheetah: `http://www.cheetahtemplate.org/`, 6.12.2010
[47] Mako: `http://www.makotemplates.org/`, 6.12.2010

3.10.1 Einsatz

Django's Templatesystem reicht für viele Webanwendungen vollkommen aus und kann auch für die Generierung anderer Dokumente wie CSV[48] oder XML verwendet werden. Somit ist dieses System sehr flexibel einsetzbar und nicht nur auf HTML beschränkt. Das Templatesystem besitzt, wie viele andere auch, bestimmte Anweisungen, um z.B. Text auszugeben, Blöcke zu markieren oder Logik auszudrücken. Die logischen Anweisungen sind aber nicht mit der Mächtigkeit von Python und seiner Verwendung in den Views zu vergleichen, da bei der Entwicklung darauf geachtet wurde, dass Logik und Darstellung voneinander getrennt werden. So dienen die Anweisungen in Templates meist zur Iteration von Dictionaries und der Verbesserung der Lesbarkeit von Werten bei ihrer Ausgabe.

Es existieren z.B. Daten-Funktionen, die die Darstellung eines Datums so verändern können, wie die Benutzer der Anwendung sie gewohnt sind. Solche Funktionen werden Built-in-Filter[49] genannt und erleichtern die Manipulation des Darstellung, bevor HTML an den Client zurückgeschickt wird.

3.10.2 Bibliotheken

Über die Tag- und Filter Library hinaus existieren weitere Bibliotheken zur weiteren Manipulation der Ausgabe.

- django.contrib.humanize
- django.contrib.markup
- django.contrib.webdesign
- i18n

Diese zusätzlich Bibliotheken ermöglichen es z.B. die Lesbarkeit von Informationen zu verbessern, aus HTML z.B. reST (reStructured Text) zu machen, dem Webdesigner bei der Generierung von sogenanntem *lorem ipsum*-Text[50] zu unterstützen oder auch bei der Internationalisierung der Anwendung.

[48] CSV: `http://docs.djangoproject.com/en/1.2/howto/outputting-csv/`, 6.12.2010
[49] Built-in-Filter: `http://docs.djangoproject.com/en/1.2/ref/templates/builtins/#built-in-filter-reference`, 6.12.2010
[50] Lorem ipsum: `http://www.lipsum.com/`, 6.12.2010

3.10.3 Templates in der Mikroadmin Applikation

Für die Erstellung von Mikroadmin sind 13 Templates entstanden, die Vererbungsstruktur ist in Abbildung 3.3 zu sehen. Alle Templates können vollständig im Repository[51] eingesehen werden.

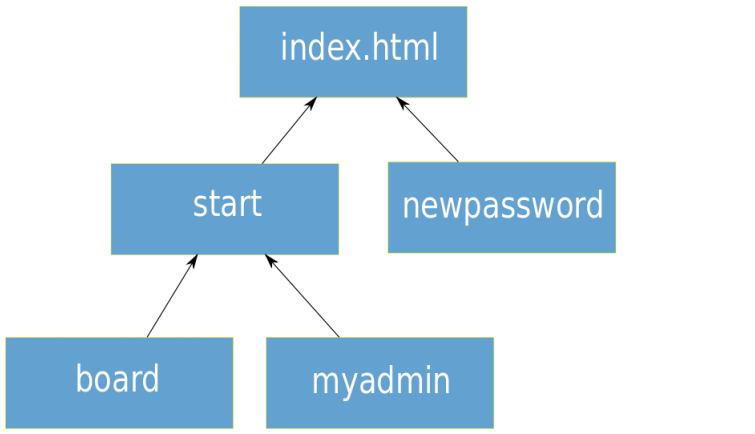

Abbildung 3.3: Templates Vererbungsstruktur

Die einzelnen Teile, die von abgeleiteten Templates überschrieben werden dürfen, werden mittels Block-Tags markiert. Diese Markierungen werden in den Eltern-Templates vorgenommen. Wenn nun auf der abgeleiteten Seite bestimmter Inhalt mit neuem ausgetauscht werden soll, wird dieser in dem abgeleiteten Teil wiederum in einem Block-Tag eingebettet. Um mehrere Block-Tags zu erstellen, müssen diese benannt sein, um somit angesprochen werden zu können. Werden Bereiche, die in den Eltern-Templates markiert sind, nicht von Kind-Templates überschrieben, bleibt der Inhalt des Eltern-Templates erhalten.

[51] Mikroadmin Templates: `http://www.gitorious.com/webframeworks/mikroadmin/trees/master/templates/mikrocontrollertemplate`, 6.12.2010

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN
    " "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head>
 <title>
 {% block title %}
 MikroAdmin
 {% endblock %}
 </title>
</head>
<body>
{% block body %}
Inhalt des Eltern-Templates
{% endblock %}
</body>
</html>
```

Listing 3.29: Beispiel für ein Eltern-Template index.html

Alle Blöcke, die das Kind-Template ersetzen soll, müssen wiederum innerhalb von Block-Tags eingegeben werden. Die restlichen Teile des Eltern-Templates werden so übernommen, wie ursprünglich angegeben.

```
# Befehl zur Kennzeichnung der Vererbung
{% extends "mikrocontrollertemplate/index.html" %}

{% block title %}
Startseite nach dem Login
{% endblock %}

{% block body %}
Inhalt des Kindtemplates
{% endblock %}
```

Listing 3.30: Beispiel für ein Kind-Template start.html

Mit Hilfe der Block-Tags ist es sehr einfach eine Struktur aufzubauen und dem DRY-Prinzip treu zu bleiben. Durch die Vererbungshierarchie kann die Struktur und das Aussehen der Seite leicht an zentraler Stelle mittels CSS verändert werden. Die Auswirkungen sind nach dem Aufruf über die View auch am Kind-Template zu sehen. Dies erspart dem Designer viel Zeit und der Überblick über den HTML-Code wird erleichtert, da dieser auf ein Minimum reduziert wird.

3.10.4 Einbindung weiteren Codes

Wie in anderen Templateengines auch, kann Code eingebunden werden, der nicht unbedingt zum Vererbungsmuster passt. Zum Beispiel bestimmte Informationen, wie eine Adresse, die im Impressum oder der Kontaktseite auftauchen soll. Diese beiden Seiten aber in keinem Vererbungsverhältnis stehen. Für solche Fälle ist die Anweisung {% include 'adresse.html' %} sehr nützlich. Die aktuelle Adresse steht dann allen Templates, die dieses einbinden zur Verfügung. Ein weiterer Vorteil mittels eines Includes ist, dass auf dem Context, des einbindenden Templates, zugegriffen werden kann. Somit auch auf alle Variablen, die über die View an das Template übergeben werden.

3.10.5 Darstellungslogik

Weiterhin gibt es Tags, die es dem Entwickler ermöglichen, Objekte in einem Template über eine Schleife auszugeben. Innerhalb der Schleife kann auf alle Eigenschaften, die das ermittelte Objekt hat, zugegriffen werden. Um eine Objektliste auszugeben, muss diese, ähnlich wie in Python, in eine For-Schleife eingebunden sein. Innerhalb dieser wird über ein Bezeichner auf das einzelne Element zugegriffen. Ein Auszug aus dem Template *start.html*, erläutert diesen Vorgang.

```
1  <ul id="boardoverview">
2  {% for board in boards %}
3  <li class="boardunit">
4      <a href="/mikroadmin/board/{{ board.name }}">
5              {% if board.isReservedBy == request.user %}
6                  <img class="boardimage" src="/site_media/mikrocontrollertemplate/images
                       /standardmikrocontroller-user.jpg" alt="{{ board.name }}" />
7              {% else %}
8              {% if board.isReservedBy %}
9                   <img class="boardimage" src="/site_media/mikrocontrollertemplate/
                        images/standardmikrocontroller-reserviert.jpg" alt="{{ board.name
                        }}" />
10             {% else %}
11                  <img class="boardimage" src="/site_media/mikrocontrollertemplate/
                        images/{{ board.image }}" alt="{{ board.name }}" />
12             {% endif %}
13             {% endif %}
14     </a>
15     <div class="boardname">{{ board.name }}</div>
16 </li>
17 {% endfor %}
```

```
18 </ul>
```

Listing 3.31: Auszug aus start.html für die Ausgabe der Mikrocontroller

Über den *Context* in der View, werden die Boards mittels der Variablen *boards* dem Template zugänglich gemacht. Die Boards werden mittels *<ul>* als Listenpunkte ausgegeben. Die einzelnen Objekte werden in der Schleife {% for board in boards %} angesprochen. Abhängig davon, ob ein Board vom aktuellen User reserviert ist, das Board bereits anderweitig vergeben oder noch frei ist, wird ein anderes Bild ausgewählt und dem User angezeigt. Die Abfrage der jeweiligen Zustände geschieht jeweils über verschachtelte {% if %} {% else %} Konstrukte. In diesem Fall konnte die Auswahl des Bildes ins Template ausgelagert werden, da es sich um keine sehr komplexe Abfrage handelt.

Da Python viel mächtiger als die mitgelieferte Template-Sprache ist, sollte so viel Programmcode wie möglich in die View ausgelagert werden. Und nur Abfragen oder Kontrollstrukturen, die im Grunde nur der Darstellung dienlich sind, im Template stehen. So kann Programmlogik und Darstellung gut voneinander getrennt werden.

3.11 Formulare

Um den Dialog mit dem User über das Internet zu führen ist es notwendig, dass dieser Daten eingeben kann bzw. mit Hilfe einer Navigation durch die Anwendung geführt wird. Durch das Klicken von Links ist es möglich die Seite zu steuern und Informationen zu senden. Meistens geschieht das durch die Übermittlung von Zusatzinformationen in der URL, die Skripte auf der Serverseite entgegennehmen und verarbeiten. Normalerweise geschieht das in einer Form durch eine Mischung aus Skriptname und übergebenen Informationen *index.php?vorame=Max&nachname=Mustermann*. Manchmal reicht dieser Ansatz nicht aus, da die Menge der Daten, die über die URL übermittelt werden kann, begrenzt ist. Auch Sicherheitsprobleme können dabei auftreten. Vor allem weil diese per GET abgesetzt werden und z.B. Passwörter durch Dritte dadurch leicht lesbar sind. Um personenbezogene Daten oder größere Informationen an den Server zu übermitteln, ist es besser auf Formulare zurückzugreifen. Dies vereinfacht es dem User, Daten in einer geordneten Form einzutragen und durch die Betätigung eines Buttons, diese an den Server zu übermitteln. Ein weiterer Vorteil ist, dass dann mittels JavaScript auf der Clientseite Eingaben verifiziert werden können, bevor diese an den Server übertragen werden.

3.11.1 Erzeugung von Formularen

Bei Django gibt es die Möglichkeit Formulare wie gewohnt per HTML in Templates einzubetten und einzelne Felder per Hand zu definieren. Darüber hinaus existiert die sogenannte *forms*-Library[52], mit der es möglich ist, Formulare objektorientiert zu deklarieren. Bei der Deklaration können Bedingungen vermerkt werden, wie z.B. ob das Feld leer sein darf, ob es ein Email-Feld sein soll oder welche Länge der Eintrag maximal haben darf. Bei der Erstellung der Formulare als Klasse, müssen diese von *from django import forms* abgeleitet sein. Die *forms*-Klasse bietet eingebaute Methoden zur Verifizierung und nimmt dem Entwickler diese wiederkehrende Arbeit bei Formularen ab. Hier ein Auszug aus der Formularklasse, welche die Registrierung für die Mikroadmin-Applikation handhabt.

C.2

```
1 class RegisterForm(forms.Form):
2     username = forms.CharField(max_length=30, label='Username:')
3     email = forms.EmailField(label='Email:')
4     password1 = forms.CharField(widget=forms.PasswordInput, label='Passwort:')
5     password2 = forms.CharField(widget=forms.PasswordInput, label='Passwort wiederholen
          :')
```

Listing 3.32: Formulare als Klasse deklariert

Es wird standardmäßig angenommen, dass alle Felder benötigt werden. Ist ein Formularfeld doch nicht zwingend notwendig, kann dieses mit der Anweisung *required=False* entsprechend markiert werden. Darüber hinaus existieren sogenannte *Widget*-Klassen[53], mit denen sich die genaue Art der Formularfelder definieren lassen und auch ihre Darstellung als HTML. Innerhalb der View, in der das Formular erzeugt wird, kann vor der Darstellung im Template, das Formular auf Validität hin überprüft werden. Das bedeutet, dass alle Felder in der Klasse, den Deklarationen bei einer Validierung folgen müssen. Werden diese Regeln veletzt, wird im ausgegebenen Template darauf hingewiesen und die noch nicht validen Felder mit einem Hinweis versehen. Die Validierung wird mit dem Befehl *formularname.is_valid()* in der View angestoßen. Bereits in die Felder eingebebene Informationen werden weitergereicht, da der RequestContext von der View an das Template übertragen wird. Somit müssen in die Felder, die den Test bestanden haben, Daten nicht noch einmal eingegeben werden. Die Validierung in der View findet

[52]Forms Library: `http://docs.djangoproject.com/en/1.2/topics/forms/`, 6.12.2010
[53]Widget: `http://docs.djangoproject.com/en/1.2/ref/forms/widgets/`, 6.12.2010

folgendermaßen statt.

C.5

```
1   if request.method == 'POST':
2       regform = RegisterForm(request.POST)
3
4       if regform.is_valid():
5           cleanedUsername = regform.cleaned_data['username']
6           cleanedEmail = regform.cleaned_data['email']
7           cleanedPassword1 = regform.cleaned_data['password1']
8           cleanedPassword2 = regform.cleaned_data['password2']
9
10          if not regform.samePassword(cleanedPassword1, cleanedPassword2):
11              registinfo = 'Passwort1 und Passwort2 sind nicht gleich'
12
13          elif regform.isRegisterd(cleanedUsername):
14              registinfo= 'User existiert bereits'
```

Listing 3.33: Formularvalidierung

Ist der Test bestanden, wird ein Dictionary namens *cleaned_data* erzeugt, worüber auf die einzelnen Werte der Felder des Formulars in der View Zugriff genommen werden kann. Mit Hilfe selbst definierter Methoden wird im folgenden überprüft, ob die Werte der Passwortfelder übereinstimmen und ob der User bereits in der Datenbank existiert. In der Variablen *registinfo* wird die jeweilige Fehlermeldung festgehalten und dann dem Template für die Fehlerausgabe zur Verfügung gestellt. Treten bei der Überprüfung keine weiteren Fehler auf, wird anschließend ein neuer User erstellt und in die Datenbank überführt.

3.11.2 Formulare einbinden

Auf der Template Seite können Formulare[54] sehr einfach eingebunden werden. Dazu reicht der Name des erzeugten Formulars und eine der folgenden Anweisungen.

- **formular.as_p** in   eingeschlossen
- **formular.as_table** in <tr> <td> </td> </tr> eingeschlossen
- **formular.as_ul** in <ul>   eingeschlossen

[54] Formulareinbindung: `http://docs.djangoproject.com/en/dev/topics/forms/#displaying-a-form-using-a-template` 6.12.2010

Es ist darauf zu achten, dass im Template um die verschiedenen Feld Anweisungen herum, jeweils ein Formular-Tag und ein Submit-Button eingebunden werden muss, da nur die einzelnen Felder und ihre jeweilige Bezeichnung ausgegeben werden. Im Folgenden ein Beispiel, wie die Einbindung mittels *formular.as_p* generiert werden kann.

D.3

```
1  <form id="registerformular" method="post" action="/mikroadmin/">
2     {% csrf_token %}
3     <fieldset id="registerformularfieldset">
4     <legend class="legend">Registrierung:</legend>
5
6     {{ regform.as_p }}
7
8     <input type="submit" id="submitregister" value="Registrieren"/>
9     </fieldset>
10 </form>
```

Listing 3.34: Einbindung eines Klassenformulars in ein Template

Im Beispiel dienen *fieldset* und *legend*, der Umrandung des Formulars und der Erzeugung einer passenden Überschrift. Für die Einbindung des Formulars, sind diese jedoch nicht nötig. Um Missbrauch vorzubeugen wird in jedem Formular eine *{% csrf_token %}* Anweisung eingesetzt, die den Schutz vor CSFR[55] Attacken erhöht. Näheres unter Sicherheitsaspekte im Abschnitt 3.13.

3.11.3 Weitere Hinweise

Es ist durchaus möglich die Ausgabe der Formulare selbst zu gestalten[56], falls die oben genannten Methoden nicht ausreichen sollten. So sind verschachtelte Konstruktionen mit * *-Tags möglich. Auf die einzelnen Formularfelder bzw. den Fehlermeldungen kann mit *formularname.feldname* oder *formularname.feldname.erros* zugegriffen werden, um ein verbessertes Fehlerfeedback zu ermöglichen.

[55] Cross-Site Request Forgery: `http://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29`, 6.12.2010

[56] Formulare selbst gestalten: `http://docs.djangoproject.com/en/dev/topics/forms/#customizing-the-form-template`, 6.12.2010

3.12 JavaScript und Ajax

Soll die Userinteraktion mit einer Anwendung gut funktionieren, reicht es heutzutage nicht mehr aus, nur Links und Formulare zur Verfügung zu stellen.

Im Desktop Bereich gibt es eine vielfältige Auswahl an GUI-Frameworks wie z.B. QT[57], GTK[58], AWT und Swing für verschiedene Plattformen. So hat sich im Laufe der Zeit, seit dem Entwurf von ECMA-Script - der Sprachkern von JavaScript - und dem sogenannten 1. Browserkrieg[59] der 1995 bis 1998 zwischen Netscape und Microsoft stattfand, bei der clientseitigen Programmierung im Browser sehr viel getan. So kann die Verbindung aus HTML und JavaScript als eine eigenständige GUI angesehen werden.

Mit Ajax[60] ist es möglich HTTPRequests abzusetzen und mittels JavaScript per DOM-Manipulation die Ergebnisse des Requests an beliebiger Stelle auf einer HTML-Seite einzubinden. Und das ohne einen vollständigen Page-Refresh abzusetzen.

3.12.1 JavaScript Frameworks

In diesem Zusammenhang gibt es mittlerweile auch verschiedene JavaScript-Framewoks, die Ajax-Funktionalität anbieten und eine einfache Benutzung der JavaScript Funktionalität versprechen. Folgende Auflistung zeigt bekannte Vertreter dieser Frameworks.

- **Dojo** (`http://dojotoolkit.org/`)
- **jQuery** (`http://jquery.com/`)
- **MochiKit** (`http://mochi.github.com/mochikit/`)
- **MooTools** (`http://mootools.net/`)
- **Prototype** (`http://www.prototypejs.org/`)
- **Yahoo! User Interface** (`http://developer.yahoo.com/yui/`)

[57] QT: `http://qt.nokia.com/products/`, 7.12.2010
[58] GTK: `http://www.gtk.org/`
[59] Browserkrieg: `http://server02.is.uni-sb.de/courses/ident/kontroverses/browserkrieg/`, 7.12.2010
[60] Ajax: `http://openbook.galileocomputing.de/javascript_ajax/18_ajax_002.htm`, 2.12.2010

All diese Frameworks bieten die Möglichkeit den DOM-Baum zu manipulieren, Ajax Interaktionen, Animationen oder Drag and Drop in Projekte einzubinden. Die Wahl eines bestimmten Frameworks ist nicht leicht, da diese von der Erfahrung und Geschmacks des Entwicklers abhängt.

Auch Django lässt dem Entwickler die freie Wahl. Hier kann ein beliebiges Framework eingebunden beziehungsweise mit anderen gemischt werden. Bei der Entwicklung der Mikroadmin-Applikation ist die Wahl auf jQuery gefallen, aufgrund bereits gemachter Erfahrungen mit diesem Framework.

3.12.2 Einbindung von jQuery

Um jQuery auf der Seite einzubinden gibt es verschiedene Methoden. Die gewöhnlichste ist, den jQuery Code auf den eigenen Server zu laden und ihn so direkt per *<script type="text/javascript" src="/js/jquery.js"></script>* einzubinden. Darüber hinaus existiert die Methode, verschiedene Frameworks über *googles* CDN[61], den Usern zur Verfügung zu stellen. Vorteil[62] dieser Methode ist, dass jQuery und andere Frameworks so schneller den Benutzern zur Verfügung gestellt werden können. Dies ist darauf zu führen, dass *google* den schnellsten zur Verfügung stehenden Server für den User ermittelt. Manche Browser haben eine Begrenzung in der Anzahl der Verbindungen[63], die sie zu einem Server aufbauen dürfen. Dadurch, dass eine Verbindung weniger mit dem Zielserver aufgebaut werden muss, kann diese Verbindung dazu genutzt werden weitere Daten wie Bilder usw. für den User vom Server aus zu laden. Ein weiterer Vorteil liegt darin, dass wenn der User bereits andere Seiten besucht hat, die auch den jQuery Code in gleicher Version vom CDN beziehen, dieser bereits im Cache des Browsers vorhanden ist und dieser gar nicht erst runtergeladen werden muss. Um die schnellste Verbindung zum CDN zu erhalten und jQuery gewohnt zu benutzen, wird das Framework in der Mikroadmin Applikation deswegen folgendermaßen im *index.html* Template eingebunden. So steht jQuery allen weiteren Templates, die von diesem abhängen auch zur Verfügung.

D.3

[61] Content Distribution Network `http://code.google.com/intl/de-DE/apis/libraries/devguide.html`, 7.12.2010

[62] Gründe für Frameworks über CDN: `http://encosia.com/2008/12/10/3-reasons-why-you-should-let-google-host-jquery-for-you/`, 7.12.2010

[63] Gleichzeitige Verbindungen zu einem Server: `http://www.heise.de/ct/artikel/Surf-Triathlon-291708.html`, 7.12.2010

```
<script
type="text/javascript"
src="http://code.jquery.com/jquery-latest.min.js">
</script>
```

Listing 3.35: Einbinden von jQuery

3.12.3 jQuery im Einsatz

In Mikroadmin wird jQuery hauptsächlich dafür benutzt, die Funktionalität anzustoßen, die auf der Verwaltungsseite eines Boards angeboten wird. Unter anderem die Reservierung und somit die Bereitstellung eines Passworts für den SSH-Zugang bzw. den PowerOn/Off oder den Reset-Vorgang eines Mikrocontrollers.

D.16

```
$(document).ready(function(){

$('.boardlink').click(function(){
        var link = $(this).attr('alt')
        /*alert(link);*/
        $("#navigationfeedback").empty().html('<div id="information"><img src="/
            site_media/mikrocontrollertemplate/images/loading.gif" /><span>Loading ...
            </span></div>');

        $('#navigationfeedback').load(link)
    });

});
```

Listing 3.36: Board Links

Mit diesem Codestück können alle Links angesprochen werden, die im Class-Attribut mit *boardLink* markiert sind. Nach einem Klick auf den Link wird aus dem *alt*-Attribut ausgelesen, welche URL und somit welche View nach dem Klick angestoßen werden soll. Für die Unterscheidung, welcher Link aktiviert wurde, wird der *$(this)*-Selector in jQuery eingesetzt. Mit diesem Selektor ist es möglich den aktuellen Link anzusprechen, somit ist nur eine einzige jQuery Funktion nötig, um alle mit *.boardlink* markierten Links auszuführen. Ohne die Markierung der Links mit einem *class*-Attribut und dem *$(this)* Selektor hätte für jeden Link eine neue Funktion geschrieben werden müssen.

Solange die aufgerufene View noch nicht geladen wurde, kann ein Lade-Icon an der Stelle gesetzt werden, an der die Ausgabe erfolgen soll. Diese Möglichkeit bietet jQery

mit Hilfe der *emty()*-Funktion an. Die Ajax-Funktionalität kann dann mittels *load()* aufgerufen werden, um den Inhalt des markierten DIV-Tags mit *navigationfeedbak* zu überschreiben. Dies geschieht aber erst, wenn die View mit ihrer Verarbeitung fertig ist.

Weiterhin wird in der Applikation JavaScript eingesetzt, um Fehlermeldungen auszublenden oder beim Klick auf Formularfelder diese zu leeren. Der Einsatz von JavaScript führt bei der Mikroadmin-Applikation zu einer besseren Benutzerführung und beschleunigt das Feedback von Aktionen, da nur einzelne Teile nachgeladen werden müssen.

3.12.4 Warum ein JavaScript Framework

Zwar wurde in der Applikation wenig JavaScript eingesetzt und für dieses Maß wäre ein Framework nicht unbedingt nötig gewesen. Der Vorteil liegt aber darin, dass mit jQuery die Auswahl von HTML-Tags sehr einfach gelingt und Ajax mittels der *load()*-Funktion einfach zu handhaben ist. Ein weiterer Vorteil ist, dass der entwickelte Code browserübergreifend funktioniert. Somit ist mit keinen bzw. wenig Problemen mit verschiedenen Browsern zu rechnen, da die Frameworkentwickler Anpassungen für diese vornehmen. In der Praxis stellt sich die Anpassung von purem JavaScript an verschiedene Browser als eine mühsame Aufgabe heraus, da jeweilige JavaScript-Engines[64] den gleichen Code teilweise anders interpretieren.

3.13 Sicherheitsaspekte

Usereingaben sind das Einfallstor von Angriffen vieler Webapplikationen, deswegen sollten sie aus Sicherheitsgründen besonders unter die Lupe genommen werden. Es kann nie sichergestellt werden, dass derjenige am Browserende gute oder böse Absichten verfolgt. Dabei kann es sich um einen normalen User handeln oder um jemanden, der die Applikation angreifen möchte. Die Entwickler des Django Frameworks verfolgen nicht die Absicht alle möglichen Zero-Day-Exploits, die es gibt, abzuwehren. Ihr Ziel in Sachen Sicherheit ist viel mehr einen guten grundlegenden Schutz vor den bekanntesten Angriffen im Web zu bieten und Eingaben, die zur weiteren Verarbeitung an Django geschickt werden, ungefährlich zu machen. Dies geschieht normalerweise automatisch und der Entwickler muss sich im Grunde nicht darum kümmern. Trotzdem ist es wichtig zu

[64] Webdesign mit JavaScript [9, Kap. 6]

wissen, wie dieser Schutz in Django funktioniert.

3.13.1 SQL Injection

Werden beispielsweise Anfragen an den Server gestellt, Informationen aus der Datenbank in einem Template anzuzeigen, kann es passieren, dass schädlicher SQL-Code in die Anfrage eingeschleust[65] wird. Vor allem dann, wenn Eingaben nicht nach bestimmten Zeichen vor der Ausführung geparst werden, um sie per \-Backslash zu escapen und mögliche Angriffe abzumildern.

Dieses Problem taucht bei Django nur auf, wenn der Entwickler spezielle SQL-Anfragen *händisch* formuliert, also bei Benutzung der Low-Level SQL-API und somit ohne den OR-Mapper. Sind Low-Level SQL-Anfragen doch mal nötig können diese durch die Methode *cursor.execute(sql, [parameter])* sicherer gemacht werden. Dabei wird String Interpolation vermieden, also die direkte Einbindung von Usereingaben in den SQL-Befehl. Mit Hilfe dieser Methode werden *bind parameters*[66] genutzt und vor der Übergabe an die SQL-Anweisung schädliche Zeichen *escaped* und dann erst an das Statement zur weiteren Verarbeitung übergeben.

Alle Abfragen der Applikation wurden ausschließlich mittels OR-Mapping durchgeführt. Somit kümmert sich das Framework um die Bereinigung von Daten, die an Mikroadmin geschickt werden und eventuell Schaden anrichten können.

3.13.2 XSS

Ein weiterer oft genutzter Angriff ist Cross-Site Scripting. Bei diesem Angriff kann HTML in eine fremde Seite eingeschleust werden. Das Problem entsteht hauptsächlich, wenn Eingaben auf der Clientseite vor weiterer Verarbeitung nicht so umgewandelt werden, dass kein fremder HTML Code in die Seite eingebettet werden kann. Schlimmer noch, wenn fremder Code es bis zur Datenbank schafft und permanent Teil der angegriffenen Seite wird.

In Django wird dieser Angriff durch das Templatesystem abgefedert. In HTML-Tags vorkommende Zeichen, die von einer View und somit einem Request stammen, werden

[65] SQL-Injection: `http://www.owasp.org/index.php/SQL_Injection`, 7.12.2010

[66] Bind Parameter/Variablen: `http://www.adp-gmbh.ch/ora/concepts/bind_variables.html`, 7.12.2010

vor der HTML Erzeugung durch das Templatesystem umgewandelt. Betroffen von der Umwandlung sind ', ", < und >. Diese Zeichen werden in HTML für die Attributwerte beziehungsweise für die Einleitung und Beendigung von Tags benötigt. Die genannten Zeichen werden in *", ', < und >*, also ihren entsprechenden Entities[67] übersetzt.

Alle Views in der Mikroadmin Applikation, die eine Antwort an der Browser zurückgeben, sind mittels Django vor XSS Angriffen geschützt. Für die Rückgabe von HTML-Code an den Browser wird die Methode *render_to_response()*[68] eingesetzt, welche ein Template als Parameter benötigt und für die Übergabe der Daten an das Template optional ein Dictionary aufnehmen kann.

3.13.3 CSRF

Eine CSFR[69]-Attacke kann zustande kommen, wenn einem User eine URL untergeschoben wird, die der Browser oder der User selbst ausführt. Das Austricksen des Users, kann zum Beispiel per Email geschehen, indem ihm ein Link zur Ausführung offeriert wird. Voraussetzung für einen solchen Angriff ist, dass das Opfer auf der Zielseite angemeldet ist. Die einfachste Variante wäre z.B. die Ausführung eines getarnten Links, das einen Logout des Users hervorruft.

Um diesen Angriffstyp unter Django abzumildern, kann das Middleware Paket *django.contrib.csrf* genutzt werden. Dieses wird in der *setting.py*-Datei im *MIDDLEWARE_CLASSES*-Bereich eingebunden. Mit diesem Paket können dann POST-Anfragen an den Server sicherer gemacht werden. Dabei werden Formulare in allen Templates mit einem {% csrf_token %} versehen.

```
1 <form id="loginformular" method="post"
2 action="/mikroadmin/accounts/login/">
3     {% csrf_token %}
4     <label for="username">Username:</label>
5     <input type="text" id="username" name="username" />
6     <label for="password">Passwort:</label>
7     <input type="password" id="password" name="password" />
8     <input type="submit" id="submitlogin" value="Login"/>
```

[67] Entities: http://www.w3schools.com/tags/ref_entities.asp, 7.12.2010
[68] Response Shortcuts: http://docs.djangoproject.com/en/1.2/topics/http/shortcuts/, 7.12.2010
[69] CSRF: http://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29, 7.12.2010

```
9       <input type="hidden" name="next" value="/start" />
10  </form>
```

Listing 3.37: Schutz gegen CSRF mittels Token

Dieses Token bewirkt, dass ein *hidden*-Field im geschützten Formular eingebettet wird. Der Wert des versteckten Feldes, wird aus der Session-ID mittels Hashing erzeugt. Gleichzeitig wird im Browser des Users ein Cookie gesetzt. Fehlt dieser Hashwert beim Absetzen eines POST-Requests oder stimmt dieser nicht mit dem Wert des Cookies überein, wird eine 403 HTTP-Fehlermeldung ausgegeben und der Befehl und die Aktion werden nicht ausgeführt. Damit das CSRF-Feld mit dem Hashwert im Template gefüllt werden kann, muss die View entsprechend, den *Context* an das Template weitergeben können. Die Weitergabe erfolgt durch das Einbinden von *RequestContext(request)* innerhalb des *render_to_response()* Shortcuts. Im Folgenden ein Beispiel für die View *index()*, die das Template der Startseite zurückgibt. Bevor die Seite an den Aufrufer zurückgegeben wird, fügt das Middelware Paket das CSRF-Token dem RequestContext hinzu.

C.5

```
1   def index(request, loginerror=None):
2
3       .
4       .
5       .
6
7       return render_to_response('mikrocontrollertemplate/index.html',
8                                 {'regform': regform,
9                                  'loginerror': loginerror,
10                                 'registinfo': registinfo},
11                                context_instance=RequestContext(request))
```

Listing 3.38: Weitergabe des CSRF-Tokens an das Formular

Dieser Schutz wirkt wie angesprochen nur auf Formulare, die mittels POST abgesendet werden. Alle anderen Anfragen, die mittels GET abgesetzt werden, sollte der Entwickler mittels Escaping vor weiterer Verarbeitung selbst bereinigen.

3.13.4 Weitere Angriffsarten

Natürlich gibt es noch weitere Arten von Angriffen, die hier nicht weiter im Detail besprochen werden. Diese sollte der Entwickler einer Webapplikation trotzdem kennen.

Im *Djangobook*[70] werden weitere mögliche Angriffsvektoren aufgezeigt und mögliche Lösungen zur Härtung vorgeschlagen.

3.14 Testen mit Django

Django untertützt unter anderem zwei bekannte Testumgebungen[71] mit Doctests und Unittests, die in der *tests.py*-Datei erstellt werden und mittels *python manage.py tests appName* ausführbar sind.

3.14.1 Doctests

Für Doctests werden innerhalb einer Funktion Docstrings, normalerweise zur Dokumentation genutzt, verwendet. Die Beschreibung der Tests werden so geschrieben, wie Aufrufe innerhalb eines Pythoninterpreters. Dabei werden z.B. Funktionsaufrufe für die zu testende Funktion erstellt und jeweilige Parameter, welche die zu testende Funktion für den Aufruf benötigt, übergeben. Damit ein Vergleich und somit eine Verifizierung stattfinden kann, müssen auch die möglichen Ergebnisse beziehungsweise ihre Returnwerte auf gleicher Weise deklariert sein.

```
1  def meineFunktion(liste, stelle):
2    """
3    >>> a = ['django', 'web', 'framework']
4    >>> meineFunktion(a, 0)
5    'django'
6    >>> meineFunktion(a, 1)
7    'web'
8    """
9    return liste[stelle]
```

Listing 3.39: Syntax eines doctests

Bei der Ausführung eines Tests werden die Dateien *models.py* und *tests.py* automatisch für mögliche Tests berücksichtigt. Um den Test in der Funktion, die z.B. in der *views.py* liegt, auszuführen, werden Ergänzungen in der *tests.py*-Datei benötigt. Zuerst wird die Funktion mittels *import*-Anweisung bekannt gemacht, um den Zugriff auf die Views zu ermöglichen. Damit die gewünschte Funktion und somit der Test berücksich-

[70] Django Security: `http://www.djangobook.com/en/2.0/chapter20/`, 7.12.2010
[71] Testumgebung: `http://docs.djangoproject.com/en/1.2/topics/testing/`, 7.12.2010

tigt wird, muss in dem __test__-Dictionary der Funktionsname aufgeführt sein. Das Dictionary wird in *tests.py* bekannt gemacht.

```
1  from views import meineFunktion
2
3  __test__ = {
4      'meineFunktion': meineFunktion
5  }
```

Listing 3.40: Doctests für Funktionen in der tests.py-Datei

Natürlich ist es auch möglich Klassen zu testen. Das folgende Listing zeigt die Erzeugung einer Boardinstanz und die Überprüfung des dazugehörigen Namens. Dabei wird in der *models.py*-Datei automatisch nach Tests gesucht und falls vorhanden ausgeführt. Anders als bei den Funktionen aus *views.py*, müssen keine weiteren Einstellungen vorgenommen werden. Die Testbeschreibung wird wie in Funktionen, mit Hilfe eines Kommentars definiert.

```
1  # Create your models here.
2  class Board(models.Model):
3      """
4      Class description for microcontroller which can be managed
5
6      # Testing for creation of a board instance
7      >>> myBoard = Board.objects.create(name='Atmel AT91SAM7',
8      ... description='http://www.atmel.com/', linuxUsername='',
9      ... image='standardimage.jpg', currentPasswort='', isReservedBy=None,
10     ... reservedDate=None)
11     >>> myBoard.save()
12     >>> myBoard.name
13     'Atmel AT91SAM7'
14     """
15     name = models.CharField(max_length=30)
16     # weitere Klassenattribute werden aus Platzgruenden weggelassen
17
18     def __unicode__(self):
19         return self.name
```

Listing 3.41: Auszug aus der models.py-Datei für Doctests

3.14.2 Unittest

Wer einen objektorientierten Ansatz bevorzugt, kann auf das Paket *unittest* zurückgreifen. Hiermit ist es genauso einfach möglich Tests für die Applikation zu erstellen. Auch hier werden beide Dateien für die Tests genutzt, da der *Testrunner* in beiden Dateien wie gewohnt, nach Unittests sucht. Im Folgenden Beispiel wird ein neues Objekt erstellt. Es wird das gleiche Beispiel gezeigt, um die Syntax vergleichen zu können. Im Listing wird ein Test für die Funktion *meineFunktion()*, in der Datei *tests.py* gezeigt.

```
1  from views import meineFunktion
2  import unittest
3
4  class MyFuncTestCase(unittest.TestCase):
5      def setUp(self):
6          self.liste = ['django', 'web', 'framework']
7
8      def testmeineFuktion(self):
9          self.assertEqual(meineFunktion(self.liste, 0), 'django')
10         self.assertEqual(meineFunktion(self.liste, 1), 'web')
```

Listing 3.42: Funktionstest mittels Unittest

Dabei werden anders als bei Doctests, der *assertEqual*-Methode der Funktionsname und der mögliche Returnwert mitgeteilt. Wichtig dabei ist, dass die Methode *setUp(self)* vor allen anderen Methoden aufgerufen wird. Darin befinden sich Einstellungen und Variablendefinitonen. Auf diese Definitionen können alle weiteren Testmethoden zugreifen. Dabei gilt durch Konvention, dass diesen Methoden *test* im Namen vorangestellt werden muss. So kann der *Testrunner* erkennen welche Methoden getestet werden sollen und welche nicht. Auf ähnliche Weise können auch Klassen und die Erzeugung von Objekten getestet werden. Zur Veranschaulichung ein weiteres Beispiel.

```
1  import unittest
2
3  class MyFuncTestCase(unittest.TestCase):
4      def setUp(self):
5          self.myBoard= Board.objects.create(name='Atmel AT91SAM7',
6                      description='http://www.atmel.com/',
7                      linuxUsername='',
8                      image='standardimage.jpg',
9                      currentPasswort='', isReservedBy=None,
10                     reservedDate=None)
11
```

```
12      def testmeineFuktion(self):
13          self.assertEqual(self.myBoard.name, 'Atmel AT91SAM7')
```

Listing 3.43: Klassen testen mittels Unittest

Im *unittest*-Paket von Python 2.7 wird eine Backport-Version für Python 2.4 mitgeliefert. Diese Version genannt *unittest2* enthält erweiterte Funktionalitäten, wie z.B. *discovery*. Damit ist es möglich Tests zu lokalisieren und nur bestimmte Tests auszuführen. Eingebunden wird dieses Paket mittels *from django.utils import unittest* welches als Alias für *unittest2* fungiert.

3.14.3 Welche Testumgebung einsetzen?

Die Auswahl zwischen den beiden Testframeworks bleibt dem Entwickler überlassen und hängt von der Erfahrung mit dem Pythoninterpreter ab. Wenn diese vorhanden ist, ist Doctest sicher die erste Wahl. Falls der Entwickler z.B. aus der Java-Welt stammt, dann ist das Paket Unittest der einfachere Weg Tests zu schreiben. Es soll aber dennoch erwähnt bleiben, dass beide parallel benutzt werden können und der Entwickler von Test zu Test situationsabhängig ein Framework einsetzen kann, welches ihm in dem Augenblick die größte Flexibilität bietet.

3.15 Bereitstellung von Django-Applikationen

Im Entwicklungsstadium einer Webapplikation kann unter Django auf den eingebauten Entwicklungsserver[72] zugegriffen werden. Dieser wird mit dem Befehl *python manage runserver* auf der Entwicklungsmaschine standardmäßig auf 127.0.0.1:8000 gestartet. Bei jedem Start validiert dieser alle vorhandenen Modelle. Auch der entwickelte Pythoncode wird immer neu geladen, was während der Erstellung der Entwicklung Vorteile mit sich bringt. So muss der Server bei Codeänderungen nicht jedes Mal neu angestoßen werden. Während der Entwicklungsphase ist dieser Server ein gutes Werkzeug, welches aber nicht für Produktionszwecke eingesetzt werden sollte, da dieser keinen Sicherheits- oder Geschwindigkeitstests unterworfen wird. Dies wird in Zukunft auch so bleiben.

We're in the business of making Web frameworks, not Web servers, so im-

[72] Runserver: `http://docs.djangoproject.com/en/1.2/ref/django-admin/#runserver-port-or-ipaddr-port` 7.12.2010

proving this server to be able to handle a production environment is outside the scope of Django[73].

3.15.1 Verbindung zwischen Django und Server

Django Applikationen können mittels FastCGI, SCGI, AJP, mod_python oder mod_wsgi produktiv eingesetzt werden. Dabei wird bei den aufgezählten Techniken Pythoncode beim Start von Apache in den Arbeitsspeicher des Servers geladen, was den Zugriff auf die Applikation beschleunigt.

Um Django produktiv einzusetzen, wurde bisher sehr oft *mod_python* genutzt. Es wird aber empfohlen *mod_wsgi*[74] als Schnittstelle zum Server zu benutzen. Als Gründe[75] werden die einfachere Anbindung, bessere Performance, geringere Auslastung des Arbeitsspeichers und größere Sicherheit im Vergleich zu *mod_python* angegeben. Django wird ab Version 1.5 *mod_python*[76] nicht mehr offiziell unterstützen. So wird der Entwickler deswegen angehalten, in Zukunft auf andere Techniken zur Verbindung von Django und Webserver zu setzen.

3.15.2 Django und Apache

Ein installierter[77] und lauffähiger Python-Interpreter wird für weitere Schritte vorausgesetzt. Zur Anbindung der Applikation kommen bei Mikroadmin Apache und mod_wsgi zum Einsatz. Apache, weil der Server gut dokumentiert ist und oft bei Webapplikationen beziehungsweise im Web zum Einsatz[78] kommt. Bei mod_wsgi wird eine höhere Performance und Sicherheit versprochen, auch weil die Applikation nicht unter dem Standard Web-User laufen muss. Sondern ein eigener User für die Applikation angelegt werden kann. So haben parallel auf dem Server laufende Django Applikationen, keinen Zugriff

[73] Einsatz in der Entwicklungsphase: `http://docs.djangoproject.com/en/1.2/ref/django-admin/#django-admin-runserver`, 7.12.2010

[74] mod_wsgi: `http://code.google.com/p/modwsgi/`, 7.12.2010

[75] Django and mod_wsgi: `http://www.technobabble.dk/2008/aug/25/django-mod-wsgi-perfect-match/`, 7.12.2010

[76] Mod Python wird nicht mehr offiziell unterstützt: `http://docs.djangoproject.com/en/dev/releases/1.3/#mod-python-support`, 7.12.2010

[77] Installation von Python unter Debian GNU/Linux: `http://diveintopython.org/installing_python/debian.html`, 7.12.2010

[78] Server Einsatz im Vergleich: `http://news.netcraft.com/archives/category/web-server-survey/`, 7.12.2010

auf die jeweils anderen und sind somit vor gegenseitigem Zugriff besser geschützt. Im Vergleich dazu hat der Standard Apache-Server User, unter Debian *www-data*[79] genannt, auf alle Dateien in seinem Verzeichnis Zugriffsrechte und somit auch auf die verschiedenen Unterordner auf denen Applikationen verwaltet werden. Wenn bei Skripten Sicherheitsprobleme[79] auftreten, können somit alle Applikationen unter *www-data* betroffen sein. Eine Trennung ist aus Sicherheitsgründen sinnvoll.

Um Mikroadmin mit Apache zu verbinden, muss erst einmal *mod_wsgi* installiert werden. Dafür kann unter Debian auf der Konsole der Befehl `apt-get install libapache2-mod-wsgi` unter dem User *root* benutzt werden. Dabei wird das WSGI-Modul installiert und benötigte Dateien landen unter */etc/apache2/* in den Ordnern für Module *mods-available* und *mods-enabled*. Dabei wird Apache automatisch konfiguriert und *mod_wsgi* zu den aktiven Modulen hinzugefügt. Sollte das Modul nicht geladen werden, kann ein Neustart des Webservers helfen. Sobald das Modul installiert und aktiviert ist, muss die *httpd.conf* folgendermaßen editiert werden.

```
1 WSGIScriptAlias /mikroadmin /pfad/zu/mikroadmin/apache/django.wsgi
2
3 <Directory /pfad/zu/mikroadmin/apache>
4 Order deny,allow
5 Allow from all
6 </Directory>
```

Listing 3.44: Alias zu Mikroadmin in httpd.conf

WSGIScriptAlias[80] verhält sich ähnlich wie die Apache Alias[81] Direktive, mit dem Unterschied, dass sie die Verknüpfung zwischen einer Server-URL und einem WSGI-Skript herstellt. Das WSGI-Skript sollte aus Sicherheitsgründen[82] nicht im gleichen Verzeichnis wie die *settings.py*-Datei liegen, dieses sollte besser in einem Unterverzeichnis z.B. mit dem Namen *apache* gespeichert werden. Falls unabsichtlich auf einem übergeordneten Verzeichnis eine URL in der *httpd.conf* gemapped wird, kann es passieren, dass der Sourcecode der Anwendung und somit Datenbank Passwörter öffentlich zugänglich sind.

Sobald die angegebene URL aufgerufen wird, handhabt *mod_wsgi* den Request und

[79] www-data unter Debian: `http://wiki.debian.org/Apache` , 7.12.2010
[80] WSGIScriptAlias Funktionsweise: `http://code.google.com/p/modwsgi/wiki/ConfigurationDirectives#WSGIScriptAlias`, 7.12.2010
[81] Alias:`http://httpd.apache.org/docs/2.2/mod/mod_alias.html#alias`, 7.12.2010
[82] django.wsgi in ein Unterverzeichnis legen: `http://code.google.com/p/modwsgi/wiki/IntegrationWithDjango`, 7.12.2010

die im WSGI-Skript definierte Webanwendung wird ausgeführt. Ein Skript beinhaltet folgende Anweisungen für die Verbindung und ist in Python geschrieben.

B.1

```
import os
import sys

# Django-Installationsverzeichnis (optional)
sys.path.append('/usr/lib/python2.5/site-packages/django')

# Falls die Pfade noch nicht bereits in der PATH-Anweisung
# aufgefuehrt sind (Optional)
sys.path.append('/home/jac/webframeworks/django')
sys.path.append('/home/jac/webframeworks/django/mikroadmin')

# Angaben zur settings.py Datei und Name der Anwendung
os.environ['DJANGO_SETTINGS_MODULE'] = 'mikroadmin.settings'


import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
```

Listing 3.45: WSGI-Skript mit Informationen zur Webapplikation

Mit diesen Einstellungen werden alle Python-Skripte ausgeführt und Views mit entsprechendem HTML-Inhalt an den Aufrufer zurückgegeben. Damit auch Bilder und Stylesheet Anweisungen von der Anwendung benutzt werden können, muss dieser Inhalt in *httpd.conf* bekannt gemacht werden. So kann Apache als Server für statischen Inhalt dienen. Dies geschieht mittels *Alias* Direktive.

```
Alias /site_media/ /home/mikroadmin/django/mikroadmin/static/media/
```

Listing 3.46: Alias für statischen Inhalt

Wenn Templates auf statischen Inhalt zugreifen sollen, wird in den betreffenden HTML-Tags im Attribut *src* allen Dateien */site-media/* vorangesetzt.

Es soll drauf hingewiesen werden, dass die Empfehlung[83] lautet, einen anderen Server, auf dem Django nicht ausgeführt wird, zur Bereitstellung statischer Dateien zu benutzen. Empfohlen werden folgende Server.

- lighttpd
- Nginx

[83] Statischen Inhalt einbinden: `http://docs.djangoproject.com/en/1.2/howto/deployment/modwsgi/#serving-media-files`, 7.12.2010

- TUX
- Abgespeckte Apache Version
- Cherokee

Die Mikroadmin Applikation nutzt den gleichen Server für die Django Instanz und die Auslieferung statischer Inhalte. Der Grund ist, dass mit keiner zu großen Serverlast im produktiven Einsatz zu rechnen ist.

3.16 Fazit: Implementierung der Anwendung mit Django

Wenn der Entwickler aus einem Python-Hintergrund stammt gelingt der Umgang mit Django leicht. Auch deswegen, weil die Erstellung von Klassen und Funktionen, genauso wie in Python schnell gelingen. Entwickler aus anderen Bereichen wie PHP oder Ruby können schnell Erfolge mit Django erzielen. Auch vor dem Hintergrund, dass Django ähnlich wie in anderen Frameworks das MVC-Pattern nutzt und Python einfach zu erlernen ist.

Die weiteren Vorteile von Django sind unter anderen das *manage.py*-Skript, mit dem z.B. der Testserver gestartet wird und die Anwendungsmodelle verifiziert werden. Wenn Änderungen am Code gemacht werden, ist die Änderung beim Aufruf entsprechender Views sofort zu erkennen, da eine automatische Prüfung vor Ausführung stattfindet. So ist es nicht nötig den Server immer wieder neu zu starten. Ein weiteres Plus ist die Adminoberfläche, die einfach als Modul geladen werden kann. Eine Adminoberfläche wird sehr oft bei Webanwendungen benötigt, um schnell Änderungen an Tabelleninhalten zu ermöglichen oder zur allgemeinen Verwaltung durch einen Administrator. So sind Änderungen an Modellen oder die Vergabe von Rechten sehr leicht möglich. Es kann sofort mit der Entwicklung der Kernapplikation angefangen werden ohne parallel eine Administrationsoberfläche aufbauen zu müssen.

Die Erfahrung zeigt, dass die Konfiguration und die Bereitstellung der Applikation viel Zeit in Anspruch nehmen kann, wenn dieser Vorgang zum ersten Mal ausgeführt werden muss. Ist diese Hürde einmal genommen, können weitere Anwendungen leicht konfiguriert werden.

Die Arbeit mit Django geht leicht von der Hand. Aber die Einarbeitung in ein neues Framework ist mit etwas Zeit verbunden. Mit guter Dokumentation Seitens des Django-Projekts, Online-Tutorials und guter englischer Literatur kann in relativ kurzer Zeit ein Projekt entwickelt werden.

Kapitel 4.0

Pylons

4.1 Einführung

Pylons[1] leichtgewichtiges MVC Webframework, das auf Python basiert und unter der BSD Lizenz steht. Momentan liegt das Framework in der Version 1.0 vor. Das Framework ist im Gegensatz zu Django lose gekoppelt. Dies bedeutet, dass die meisten Teile leicht austauschbar sind und von einfachem Gluecode zusammengehalten werden. Dabei werden low-level APIs und Methoden genutzt, um die benötigten Teile zu verbinden. Für die einzelnen Teile werden keine Vorgaben gemacht, obwohl die Entwickler Mako als Templateengine und SQLAlchemy als OR-Mapper empfehlen. Pylons verspricht die Entwicklung von Webanwendungen schnell, flexibel und einfach zu gestalten. Convention over configuration ist ein zentrales Design Paradigma, das Pylons verfolgt, mit dem auch an manchen Stellen gebrochen werden kann. Für Neueinsteiger kann es am Anfang zur Hürde werden, aus den losen Teilen ein harmonierendes Ganzes zu gestalten.

4.2 Dokumentation

Die erste Anlaufstelle, um den Umgang mit Pylons zu lernen ist sicherlich die Pylons Projektseite[2], auf der neben der Dokumentation auch ein Blog und ein Wiki bereitgestellt werden. Die Dokumentation[3] selbst ist in Kapitel unterteilt und deckt die Bereiche von den ersten Schritten bis hin zur endgültigen Bereitstellung der entwickelten Applikation ab.

Darüber hinaus existiert ein Buch[4] vom Entwickler des Python Webframeworks James Gardner. Das Buch spricht den Neuling an und führt ihn bis zum Experten in Sachen Webentwicklung mit Pylons heran.

Bis auf ein paar Artikel über die Beschreibung des Frameworks[5] und ein Vergleich[6]

[1] The Definitive Guide to Pylons [10, Kap. 1]
[2] PylonsHQ: `http://pylonshq.com/`, 8.12.2010
[3] Pylons Dokumentation: `http://pylonshq.com/docs/en/0.9.7/`, 8.12.2010
[4] The Definitive Guide to Pylons [10]
[5] What Makes Pylons Stand Out As a Web Framework: `http://farmdev.com/thoughts/72/what-makes-pylons-stand-out-as-a-web-framework-/`, 8.12.2010
[6] TurboGears and Pylons: `http://blog.ianbicking.org/turbogears-and-pylons.html`, 8.12.2010

zwischen Pylons und TurboGears gibt es wenige Seiten, die sich mit Pylons befassen.

Trotz alledem existieren viele Webseiten[7] und Webapplikationen, die mit Hilfe von Pylons entwickelt wurden. Darunter bekannte, wie *reddit.com* und *bittorrent.com*.

4.3 Installation

Für die erste Installation[8] wird empfohlen, Pylons unter einer isolierten Python Umgebung zu installieren. Damit kann Pylons getestet werden, ohne die bestehende Python Installation und Pakete zu beeinflussen. Ein weiterer Vorteil besteht darin, dass wenn keine ausreichenden *root*-Rechte auf einem Server für den Zugriff auf oder die Installation von weiteren Python Paketen zur Verfügung stehen, Pylons trotzdem installiert und getestet werden kann.

4.3.1 Virtualenv

Mittels *virtualenv* kann eine virtuelle Umgebung für die Ausführung von Python geschaffen werden. Eine *.tar.gz* Datei steht auf der Pylons Seite oder unter pypi.python.org[9] zum Download zur Verfügung. Virtualenv kann folgendermaßen unter Linux bezogen und installiert werden.

```
1 $ wget http://pypi.python.org/packages/source/v/virtualenv/virtualenv-1.5.1.tar.gz
2 $ tar -xvf virtualenv-1.5.1.tar.gz cd virtualenv-1.5.1/
```

Listing 4.1: Download und entpacken von virtualenv

Nach erfolgreicher Installation, kann die virtuelle Python Umgebung generiert werden. Dazu ist unter Linux folgender Befehl nötig. Natürlich sollten die Pfade gegebenenfalls angepasst werden.

```
1 $ ./virtualenv.py /home/myhome/virtualpythonenv
```

Listing 4.2: Einrichtung einer virtualenv Umgebung

[7] Sites Using Pylons: `http://wiki.pylonshq.com/display/pylonscommunity/Sites+Using+Pylons`, 8.12.2010

[8] Pylons Installation[10, Kap.2]

[9] Pypi virtualenv: `http://pypi.python.org/pypi/virtualenv#downloads`, 9.12.2010

Sobald die Python Umgebung[10] eingerichtet ist, kann auf zusätzliche Skripte zugegriffen werden, die den Umgang mit *virtualenv* vereinfachen können. Ein sehr nützliches Tool ist *activate*, welches im *bin/* Verzeichnis zu finden ist. Damit ist es möglich, für die momentane Shell, alle Skripte in der Umgebung bereitzustellen. So kann z.B. der *python* Befehl aus der virtuellen Umgebung, ohne relative oder absolute Pfadangabe ausgeführt werden. Die virtuelle Umgebung wird mittels *source activate* aktiviert. Vor der Eingabeaufforderung erscheint in Klammern der Name der virtuellen Umgebung. In diesem Fall *(virtualpythonenv) user@rechner:~$*. Die Aktivierung kann mittels *which python* überprüft werden, dabei sollte der Pfad zu Python in der virtuellen Umgebung zu sehen sein.

Jetzt kann per *easy_install* Pylons ohne Pfadzusätze installiert werden. Dazu ist nur folgender Befehl nötig.

```
1 $ easy_install Pylons
```

Listing 4.3: Installation von Pylons

Der Befehl installiert Pylons in der neuesten Version und löst automatisch alle nötigen Abhängigkeiten auf. Dabei wird unter anderem auch die Templateengine Mako und der PasterServer installiert. Jedoch nicht SQLAlchemy, dieses Paket muss bei Bedarf nachinstalliert werden. Für spätere Updates kann der Befehl *./easy_install -U Pylons* nützlich sein. Um sicher zu gehen, dass Pylons eingerichtet wurde, kann in der Python Konsole per *import pylons*, das Laden von Pylons getestet werden.

Alle nachstehenden Befehle zur Ausführung von Skripten setzen der Einfachheit halber eine aktivierte Umgebung voraus. Die Umgebung kann durch Eingabe von *deactivate* wieder in der Shell ausgeschaltet werden. Natürlich ist es möglich die Befehle ohne aktivierte Umgebung auszuführen. Um Fehler bei der Ausführung zu vermeiden, muss darauf geachtet werden, dass die Pfade zu den Skripten richtig sind und der richtige Python-Interpreter ausgeführt wird.

4.4 Erste Schritte

Nach erfolgreicher Installation von Pylons kann das erste Projekt ins Leben gerufen werden. Dazu kann in ein beliebiges Verzeichnis gewechselt werden. Um ein neues Pro-

[10] virtualenv: `http://pypi.python.org/pypi/virtualenv`, 9.12.2010

jekt zu erzeugen wird das *PasterSkript* benötigt. Das neue Projekt wird folgendermaßen erzeugt.

```
1 $ paster create -t pylons MikroAdmin
```

Listing 4.4: Ein neues Pylons Projekt erzeugen

Mit der Option *-t* wird das Template ausgesucht, mit dem das Mikroadmin Projekt erzeugt werden soll. Mit dem Befehl *paster create –list-templates* können bereitgestellte Templates ausgegeben werden. Folgende Templates stehen am Anfang zur Auswahl. Eigene Templates können der Liste hinzugefügt werden.

- **basic_package:** Paket mit setuptools
- **paste_deploy:** Paket aus paste.deploy
- **pylons:** Pylons Vorlage
- **pylons_minimal:** Pylons minimal Vorlage

Um die Applikation zu starten kann der Paste HTTP Server aus dem Paste[11] Paket benutzt werden. Der Vorteil des HTTP Servers ist, dass er so eingestellt werden kann, dass bei Änderungen in der Codebasis oder der Konfigurationsdatei, diese automatisch während dem Betrieb neu geladen werden. Eine nützliche Einstellung während der Entwicklungsphase. Darüber hinaus kann er die Konfigurationsdateien auslesen, die Pylons benutzt.

Die erste Bereitstellung des Projektes gelingt recht einfach. Dazu muss lediglich das *PasterSkript* mit ein paar Optionen ausgeführt werden.

```
1 $ cd MikroAdmin
2 $ paster serve --reload development.ini
```

Listing 4.5: Laden einer Pylons-Applikation, während der Entwicklungsphase

Mit der *–reload* Option werden veränderte Dateien im Projekt neu geladen. Mit Hilfe einer *ini* Datei kann Pylons konfiguriert werden. Standardmäßig wir *development.ini* bei der Projekterstellung automatisch generiert. Für Unittests ist die *test.ini* Konfigurationsdatei gedacht. Wenn die Applikation einen produktiven Stand erreicht hat, kann

[11] Python Paste: `http://pythonpaste.org/`, 10.12.2010

der Entwickler eine *production.ini* erstellen, in der z.B. der Debug Modus ausgeschaltet wird.

Standardmäßig wird der Port *:5000* für die Auslieferung der Applikation in Pylons genutzt. Dieser kann jederzeit, vor Bereitstellung, in einer der gewünschten Konfigurationsdateien, geändert werden.

Der Aufruf der Applikation kann im Browser entweder mit `http://localhost:5000` oder `http://127.0.0.1:5000` erfolgen. Abbildung 4.1 wird, wenn der Aufruf klappt, angezeigt.

Abbildung 4.1: Pylons: Erfolgsmeldung

4.5 URL-Dispatcher

Unter Pylons werden ausführbare Methoden, die dem User zur Ausführung zur Verfügung stehen, Actions genannt. Die Verbindung zwischen Controllern und den im Browser eingegebenen URLs geschieht bei Pylons mit Hilfe von *Routes*[12]. Dieses Paket wird bei der Pylons Installation mitgeliefert, kann jedoch gegen einen anderen WSGI URL-Dispatcher ausgetauscht werden.

Routes ist eine Python Reimplementierung des Dispatching Systems von Ruby on Rails und verbindet in Pylons URLs mit Actions[13]. Die Routing Definitionen sind innerhalb eines Projekts im Beispielprojekt unter *Mikroadmin/mikroadmin/config/routing.py* zu finden.

Für die Verbindung wird die Mapper-Klasse benötigt, die bei Initialisierung einer Mapperinstanz den Pfad zugewiesen bekommt, an dem sich alle *Controller* befinden. Mit der *connect*-Methode werden alle nötigen Verbindungen zwischen URLs und Controllern zur erstellten Mapperinstanz hinzugefügt.

```
1  def make_map(config):
2      # Initialisierung
3      map = Mapper(directory=config['pylons.paths']['controllers'],
4                   always_scan=config['debug'])
5      map.minimization = False
6      map.explicit = False
7
8      # Fehlerbehandlung werden am Anfang hinzugefuegt, um sicherzustellen, dass
9      # sie auch aufgeloest werden
10     map.connect('/error/{action}', controller='error')
11     map.connect('/error/{action}/{id}', controller='error')
12
13     # Routen festlegen
14     map.connect('/', controller='mikroadmin', action='index')
15     map.connect('/{controller}/{action}')
16     map.connect('/{controller}/{action}/{id}')
17
18     return map
```

Listing 4.6: Routes Verbindungen in der routing.py Datei

Wie bei der ersten Verbindungsanweisung zu sehen, wird einer URL, in diesem Fall die Startseite der Applikation, mit dem *mikroadmin* Controller und somit der *index*

[12]Routes: `http://routes.groovie.org/`, 11.12.2010
[13]Actions: Controller Methoden innerhalb einer Controllerklasse in Pylons

Action verbunden. Wie Controller aussehen und was sie zurückgeben können, wird später erläutert. Bei der zweiten Verbindung wird der Name eines Controllers und eine beliebige Action erwartet. Diese wird innerhalb der angegebenen Controllerklasse gesucht und ausgeführt. Existiert die Controllerklasse aber nicht die Action, wird ein Traceback im Debug Mode angezeigt. In anderen Fällen eine *404 Not Found* Seite.

Wichtig im Zusammenhang mit Pylons zu wissen ist, dass zuerst im *public* Ordner nach passenden Dateien für die Rückgabe von HTML gesucht wird. Falls sich im *mikroadmin/public* Ordner eine Datei namens *index.html* befindet, wird diese zurückgegeben. Soll aber die Action ausgeführt werden, muss diese *index.html* Datei entweder gelöscht oder umbenannt werden.

Mit Hilfe von Routes ist es möglich gut leserliche und flexible Verbindungen zu Actions herzustellen. Auch können weitere Argumente, wie z.B. *ids* an die entsprechende Action zur Verarbeitung übergeben werden. So ist Routes ein zentrales Tool in Pylons, um zielgerichtet Applikationen auszuführen.

4.6 Geschäftslogik

Bei der Erstellung des *MikroAdmin* Projekts wurde ein Unterordner namens *mikroadmin* generiert, der weitere Unterordner beherbergt. Darunter auch ein Ordner mit dem Namen *controllers*. Mit dem PasterScript, mit dem auch schon das Projekt erstellt wurde, ist es möglich Controller zu erstellen.

```
1 $ paster controller new
```

Listing 4.7: Erstellung eines Controllers unter Pylons

Der Befehl kann innerhalb des Projekts, egal in welchem Verzeichnis, ausgeführt werden. Dabei wird im *controllers* Ordner eine Skelettdatei *new.py* erstellt erstellt, in dieser wird der grundsätzliche Aufbau einer Controllerklasse abgebildet und alle nötigen Pakete importiert. Zusätzlich wird für spätere Testzwecke, unter *tests/functional/*, die Datei *test_new.py* generiert. Hier der Aufbau der Controllerklasse *NewController*.

```
1 class NewController(BaseController):
2     def index(self):
3         # Return a rendered template
4         # return render('/new.mako')
```

```
5         # or, return a string
6         return 'Hello World'
```

Listing 4.8: Aufbau einer ControllerKlasse

Dem Namen der Klasse wird der Zusatz *Controller* gegeben und diese Klasse erbt Eigenschaften und Methoden von der BaseController Klasse. Standardmäßig wird eine Action mit dem Namen *index()* generiert, die einen einfachen String zurückgibt. Nach diesem Muster können weitere Actions innerhalb der Klasse geschrieben werden. Weiterhin kann, anstatt von HTML bzw. eines einfachen Strings, der Action auch eine Funktion zum Rendern eines Mako Templates für die Generierung von HTML übergeben werden. Diese Action kann wie bereits im Abschnitt 4.5 gezeigt, vom Benutzer einer Applikation ausgeführt werden. Sollen Actions nicht von außen über eine URL angesprochen werden können, sondern nur über Klassenmethoden, ist es möglich diese als *private* zu kennzeichnen. Damit verschiedene Controller auf oft benutzte Funktionen Zugriff erhalten, ist es sinnvoll diese unter *mikroadmin/lib* in der Datei *helpers.py* abzulegen.

4.6.1 Request und Response

Um mit den Daten aus dem Request in einer Action arbeiten zu können, wird in Pylons auf das *request* Objekt zurückgegriffen, das bei jeder Anfrage aus der Pylons *from pylons import request* Klasse neu erzeugt wird. Die Pylons Requestklasse ist eine Unterklasse von *webob.Request*[14] aus dem WebOb Package des Paste Servers. Die Pylons Requestklasse wird in jeder Controllerdatei, die vom PasterScript generiert wird, importiert. Das erzeugte Objekt gewährt den Zugang auf alle möglichen Informationen eines Requests. Der Zugriff gelingt am einfachsten in einer Action über ein Dictionary mittels *request.params*.

Auch die Antwort, also der Response zum Server, kann manipuliert werden. Das geschieht auf ähnliche Weise, wie beim Request Objekt. Der Content Type[15] kann z.B. angepasst werden, wenn die Antwort nur aus Text bestehen soll. Der Befehl in einer Action dazu lautet *response.content_type = 'text/plain'*. Dies ändert den standardmäßig auf *text/html* gestellten Typen auf den gewünschten um. Darüber hinaus können auch Cookies damit gesetzt oder gelöscht werden.

[14] WebOb: `http://pythonpaste.org/webob/`, 13.12.2010
[15] Content Types: `http://www.w3.org/TR/html4/types.html`, 13.12.2010

4.7 View

Standardmäßig wird Pylons mit Mako[16] für die View[17] ausgeliefert. Mako ist eine Templatebibliothek, basierend auf Python. Dabei bedient sie sich von anderen Templatebibliotheken, wie Djangos Templates, Cheetah oder Myghty. So können innerhalb von HTML-Code unter anderem Python Anweisungen eingebettet werden. Ähnlich wie in Python-Klassen, können Mako Templates von anderen Templates erben und für dynamische Ausgaben markierte Blöcke überschrieben werden. Laut der Mako Seite, kann die Library von der Geschwindigkeit her gut mit anderen mithalten.

Da in jeder Controllerdatei die *render()* Funktion bereits standardmäßig importiert wird, können Templates sofort für die HTML Ausgabe benutzt werden. Der Import geschieht aus *mikroadmin/lib/* in der Datei *base.py*, die wiederum die eigentliche Funktion aus dem Pylons Package lädt.

```
1 from pylons.templating import render_mako as render
```

Listing 4.9: Ursprung der Funktion zur Verarbeitung der Mako Templates

Die benutzten Mako Templates werden im Projekt unter *mikroadmin/templates/* abgelegt und von den Actions, nur durch Angabe des Templatenamens, automatisch an dieser Stelle gefunden. Befinden sich selbst erstellte Templates an anderer Stelle, kann die *paths* Liste in der *config.py* Datei entsprechend erweitert werden.

4.7.1 Umgang mit Mako Templates

Mako Templates können folgendermaßen aus einer Action aufgerufen und Argumente an diese übergeben werden.

```
1 def template(self):
2     name = 'Pylons Developer'
3     c.language = 'python'
4     return render('/greeting.html', extra_vars={'name': name})
```

Listing 4.10: Action mit Übergabe eines Mako Templates und Parametern

[16] Mako: `http://www.makotemplates.org/`, 14.12.2010
[17] Templates mit Pylons [10, Kap. 5, S. 63]

Der führende / vor dem Templatenamen sollte nicht vergessen werden, sonst wird dieser nicht gefunden.

```
<html>
<head>
<title>First Template</title>
</head>
<body>
Hello ${name}, do you speak ${c.language}?
</body>
</html>
```

Listing 4.11: Mako Template mit verschiedenen Arten zur Parameterübergabe

Durch den *extra_ vars* Parameter[18] können Daten an das Template weitergeleitet werden. Dabei kann der Name des Parameters vor der Übergabe geändert werden, dies funktioniert über einen Dictionary Eintrag. Eine weitere Möglichkeit Daten weiterzureichen, ist über den Template Context die Daten zu übermitteln. Dieser Context wird bei jedem Aufruf und somit jedem Request erzeugt und wird an globaler Stelle gesetzt. Somit können Templates auf diese Daten, ohne explizite Übergabe zugreifen. Damit das Template die Daten finden kann, muss folgender Import im Controller erfolgen.

```
from pylons import tmpl_context as c
```

Listing 4.12: Einbindung des Template Contexts

Der Vorteil ist, dass die Wahrscheinlichkeit der Kollision mit Variablen aus dem globalen Kontext so reduziert wird. Ein weiterer Vorteil ist, dass diese Daten während eines Requests an anderer Stelle der Applikation zur Verfügung stehen.

Für die Steuerung der Darstellung der Inhalte steht in Mako eine Python ähnliche Syntax zur Verfügung. Wobei die Einrückungstiefe einzelner Anweisungen keine Rolle spielt. Natürlich sollte, aus Gründen der Lesbarkeit, der Code strukturiert werden. Kontrollstrukturen und Anweisungen werden mit % eingeleitet und sollten, da wo sie auch in Python verlangt werden, mit einem *:* enden. Darüber hinaus wird das Ende eines Anweisungsblocks mit dem Einleitungswort *end* und dem Namen der entsprechenden Anweisung z.B. *for* mit *endfor* beendet. Hier ein Beispiel für den Einsatz von Kontrollstrukturen und der entsprechenden Ausgabe mit Mako.

[18] Templates Parameter: `http://pylonshq.com/docs/en/0.9.7/views/#templates`, 15.12.2010

```
# c.kontakt stammt hierbei aus dem Request Context einer Action
c.kontakt = [
             ('Guido van Rossum','guido@python.org')
             ('James Gardner','')
             ('Ben Bangert','ben@groovie.org')
            ]


## Mako Template
## (Kommentare werden in Mako mit doppeltem Doppelkreuz eingeleitet)

<ul>
% for item in c.kontakt
    <li>\
% if item[1]:
        <a href='mailto:${item[1]}'>${item[0]}</a>\
% else:
        ${item[0]}\
% endif
    </li>
% endfor
</ul>


# HTML Ausgabe:
<ul>
<li><a href='mailto:guido@python.org'>Guido van Rossum</a></li>
<li>James Gardner</li>
<li><a href='mailto:ben@groovie.org'>Ben Bangert</a></li>
</ul>
```

Listing 4.13: Kontrollstrukturen in Pylons

Durch die Ähnlichkeit der Syntax mit Python können dynamische Ausgaben sehr einfach in Mako erzeugt werden. Im Beispiel ist darauf zu achten, dass am Ende einiger Anweisungen der \ steht, dies verhindert bei der HTML Ausgabe einen Textumbruch. Obwohl es möglich ist mit Mako viel Logik ins Template auszulagern, sollte dies nicht extrem betrieben werden. Eine gute Praxis ist es, die Templatesprache nur zum Zweck der Darstellung von Daten und HTML zu verwenden und nicht als Ersatz für die Actions.

4.7.2 Weitere Mako Features

Mako bietet es an, innerhalb von Templates Funktionen[19] zu erzeugen und aufzurufen. Es ist auch möglich Funktionen aus anderen Bibliotheken zu importieren, die allerdings als ersten Parameter *context* aufnehmen müssen. Zusätzlich ist es möglich Returnwerte für Templatefunktionen zu simulieren, indem *capture*[20] den Funktionsoutput im Template zwischenspeichert und an definierter Stelle wieder ausgibt. Wenn innerhalb von Templates Funktionen anderer Templates importiert werden, kann für diese ein eigener Namespace definiert werden, um z.B. Namenskollisionen zu vermeiden. Das wichtigste Feature in Mako ist allerdings, eine Templatehierarchie aufbauen zu können. Dadurch kann der Aufwand HTML Code zu verwalten, auf ein Mininum reduziert werden.

Es ist darüber hinaus möglich eine andere Templateengine in Pylons einzusetzen, und auch ein paralleler[21] Betrieb verschiedener Engines kann realisiert werden.

Um weitere Features und Eigenschaften kennenzulernen, sind die Pylons View[22] Seite und die Mako Dokumentation[23] eine gute Anlaufstelle.

4.8 SQLAlchemy und Modelle

Durch den flexiblen Charakter von Pylons ist es möglich verschiedene Arten[24] der Speicherung von Daten zu benutzen. Die Entscheidung ist abhängig davon, mit welchen Art von Daten die Webapplikation umgehen soll.

Sollen viele binäre Daten, wie Bilder oder Videos verwaltet werden und der Erfolg der Applikation ist nicht genau vorauszusehen, kann Amazon S3[25] eine gute Option sein. Eine Menge von Python Bibliotheken, wie z.B. *boto*[26], ermöglichen den Zugriff auf Anbieter mit solchen Cloud Services.

Bestehen die Daten der Anwendung hauptsächlich aus XML Daten sind XML Datenbanken eine weitere Option unter Pylons. So müssen die Daten nicht erst in ein anderes Format umgewandelt werden und ein Zugriff mit z.B. XPath oder XQuery ist dann sofort

[19] Funktionen in Mako [10, Kap. 5, S. 76]
[20] Capture Feature: [10, Kap. 5, S. 79]
[21] Paralleler Betrieb verschiedener Template Engines: [10, Kap. 5, S. 89]
[22] Pylons Views: `http://pylonshq.com/docs/en/0.9.7/views/#templating-with-mako`, 15.12.2010
[23] Mako Dokumentation: `http://www.makotemplates.org/docs/`, 15.12.2010
[24] Pylons und Datenbanken [10, Kap. 7, S. 129-133]
[25] Amazon S3: `http://aws.amazon.com/de/s3/`, 17.12.2010
[26] boto `http://code.google.com/p/boto/`, 17.12.2010

möglich. eXist[27] und Berkeley DB XML[28] werden von Pylons unterstützt.

Ist die Datenbasis von vielen Objekten geprägt, können diese in Objekt Datenbanken untergebracht werden. Die Objekte werden in Dateien abgelegt und eine Portierung der Datenbasis auf andere Server kann so leicht gelingen. Bekannte Vertreter on Objekt Datenbanken sind Durus und ZODB. Diese beiden Datenbanken können nur Objekte aufnehmen, die durch das *pickle*[29] Modul serialisiert werden können. So ist das Auslesen von Daten aus Dateien oder RDBMS nicht direkt möglich. Das Aufrufen von Daten ist nur mit Hilfe von Python und *for*-Schleifen möglich, im Gegensatz zu Aufrufen in z.B. PostgreSQL mit optimierten *C* Suchroutinen.

Die meisten Webapplikationen unter Pylons werden mit RDBMS z.B. mit MySQl oder PostgreSQL für die persistente Haltung von Daten verwendet. Da normalerweise Objekte in diesen Applikationen miteinander interagieren, werden Objekt-Relationale-Mapper für die Verbindung der Logik mit der Datenbank benutzt. In Pylons werden bekannte ORMs, wie Storm[30] von Cannonical, SQLObject oder SQLAlchemy unterstützt. Da SQLAlchemy als ein modernes und flexibles Tool angesehen wird und eine große Community besitzt, wird es von den Pylons Entwicklern empfohlen.

4.8.1 Installation von SQLAlchemy und pysqlite2

Am Einfachsten kann der Umgang mit SQLAlchemy mit Hilfe von SQLite3 gezeigt werden. Für die Verbindung zwischen Python, SQLAlchemy und der Datenbank ist der *pysqlite2* Connector nötig. Normalerweise ist dieser bei Python standardmäßig dabei. Unter Ubuntu ist das nicht der Fall und er muss deswegen nachinstalliert werden. Dies gelingt durch den Befehl *sudo apt-get install python-pysqlite2 python-sqlalchemy*, wobei auch gleichzeitig SQLAlchemy für Python zur Verfügung steht und importiert werden kann. Auch in der virtuellen Python Umgebung.

Soll die Installation von *pysqlite2* per *easy_ install* erfolgen, muss vorher gewährleistet sein, dass die SQLite3 Entwicklungsdateien bzw. die Headerdateien vorhanden sind. Dafür sind folgende Schritte in der virtuellen Python Umgebung unter Ubuntu nötig.

[27] eXist: `http://exist-db.org/index.html`, 17.12.2010
[28] Berkeley DB XML: `http://www.oracle.com/us/products/database/berkeley-db/xml/index.html`, ,17.2.2010
[29] Pickle Modul: `http://docs.python.org/library/pickle.html`, 17.12.2010
[30] Storm ORM: `https://storm.canonical.com/`, 17.12.2010

```
# Globlale Installation unter Ubuntu
$ sudo apt-get install libsqlite3-dev

# Installation in der virtuellen Python Umgebung
$ easy_install pysqlite # obwohl pysqlite2 installiert wird
$ easy_install SQLAlchemy # Installiert die neuste Version
```

Listing 4.14: Installation von pyqlite2 per easy_install

Nach Installation beider Pakete, kann Pylons SQLAlchemy mit darunterliegender SQLite3 Datenbank nutzen. Wenn gewünscht, kann noch SQLite3 für die Kommandozeile installiert werden. Dieser Schritt ist für den Betrieb nicht notwendig und daher optional.

4.8.2 Vorteile von SQLAlchemy

Mittels der Python DB API 2.0[31] können SQL Statements an die gewünschte Datenbank gerichtet werden. Dies geschieht aber datenbankspezifisch und die Statements können bei einem Wechsel auf eine andere Datenbank eventuell wieder angepasst werden. SQLAlchemy überwindet diesen Nachteil und stellt eine Abstraktionsschicht bereit, um Abfragen, unabhängig von der darunterliegenden Datenbank, erstellen zu können.

So muss der Entwickler keine direkten SQL Statements schreiben, sondern SQLAlchemy Funktionen aufrufen und pythonische Statements generieren. Ein weiterer Vorteil ist, dass die abstrakte API SQL-Injection Angriffe[32] abschwächen kann, indem Usereingaben vor der Erzeugung der spezifischen SQL-Statements escaped werden. Durch den Aufbau eines *Connection Pool* werden Anfragen beschleunigt an die ausgewählte Datenbank weitergeleitet. Dieser Pool wird beim Start der Pylonsanwendung durch SQLAlchemy befüllt. Wenn anschließend eine Anfrage ankommt, wird im Pool nach bereits offenen Verbindungen zur Datenbank gesucht. Nach Beendigung der Anfrage wird die Verbindung nicht wieder geschlossen, sondern lediglich für zukünftige Anfragen freigegeben und in den Pool zurückgeworfen. So ist ein ständiger Verbindungsauf- und -abbau obsolet.

Wenn der Entwickler sich dennoch dafür entscheidet, RAW-SQL Statements für eine spezifische Datenbank zu schreiben, ist das mit Hilfe von SQLAlchemy möglich und zu empfehlen. Denn dadurch kann er auf alle Features[33] zugreifen, die SQLAlchemy bietet, auch auf den Verbindungspool.

[31] Python DB API 2.0 Specification: `http://www.python.org/dev/peps/pep-0249/`, 20.12.2010
[32] SQLAlchemy und SQL-Injection [10, Kap. 7, S. 142]
[33] SQLAlchemy Features: `http://www.sqlalchemy.org/features.html`, 20.12.2010

4.8.3 SQLAlchemy als OR-Mapper

Sobald SQLAlchemy und der *pysqlite* Datenbanktreiber in der virtuellen Umgebung betriebsbereit sind, können Klassen mit Tabellen verknüpft werden. Die Bereitstellung der Pakete kann mittels *import sqlalchemy* und *import pysqlite2* getestet werden, dabei dürfen keine Fehlermeldungen auftreten.

Soll ein Pylons Projekt SQLAlchemy unterstützen, kann dies bereits bei der Erstellung dem Framework mitgeteilt werden. Dabei sollte wie bei der Generierung des *MikroAdmin* Projekts Mako als Templateengine ausgewählt sein und unbedingt die Unterstützung von SQLAlchemy mit *True* bestätigt werden.

```
1  $ paster create --template=pylons Mikroadmin
2  Selected and implied templates:
3    Pylons#pylons  Pylons application template
4
5  Variables:
6    egg:      Mikroadmin
7    package:  mikroadmin
8    project:  Mikroadmin
9  Enter template_engine (mako/genshi/jinja2/etc: Template language) ['mako']: mako
10 Enter sqlalchemy (True/False: Include SQLAlchemy 0.5 configuration) [False]: True
```

Listing 4.15: Mikroadmin Testprojekt unter Pylons

Nach der Grundkonfiguration des Projekts können die Modelle definiert werden. Bis zur SQLAlchemy 0.4[34] Version gab es eine Möglichkeit Tabellen über Metainformationen zu Definieren und sie dann anschließend mittels des SQLAlchemy Mappers einer Klasse zuzuweisen. Dies ist immer noch möglich und verspricht mehr Flexibilität beim Mapping, was aber nicht immer benötigt wird. Ab der Version 0.5[35] kann dies in einem einzigen Schritt durchgeführt werden. Unter Pylons 1.0 wird empfohlen Modelle im Verzeichnis *mikroadmin/model/* als eigene Dateien auszulagern, dies soll eine bessere Übersicht ermöglichen, falls viele Modelle definiert werden. Anschließend sollten sie in die *__init__.py* Datei der *models* Verzeichnis, für spätere Verarbeitung der Modelle in den Actions, am Anfang der Datei importiert werden. Das folgende Listing zeigt, die Definition eines Modells.

[34] Ursprüngliche Mapping Methode: `http://pylonshq.com/docs/en/0.9.7/models/#defining-tables-and-orm-classes`, 21.12.2010

[35] Empfohlenes Klassenmapping: `http://pylonshq.com/docs/en/1.0/models/#creating-a-model`, 21.12.2010

```
1  """ Board model """
2  from sqlalchemy import Column, ForeignKey, DateTime, text
3
4  from sqlalchemy.types import Integer, String
5
6  from mikroadmin.model.meta import Base
7  #from meta import Base
8
9
10 class Board(Base):
11     __tablename__ = 'board'
12
13     id = Column(Integer, primary_key=True)
14     name = Column(String(30))
15     description = Column(String(30), nullable=True)
16     linuxUsername = Column(String(30), nullable=True)
17     image = Column(String(30), nullable=True)
18     currentPassword = Column(String(30), nullable=True)
19     isReservedBy = Column(Integer, ForeignKey('user.id'), nullable=True)
20     reservedDate = Column(DateTime, server_default=text('CURRENT_TIMESTAMP'),
21     nullable=True)
22
23     # Textuelle Darstellung einer Boardinstanz
24     def __repr__(self):
25         return "<Board('%s')" % self.name
```

Listing 4.16: Modell erstellen unter Pylons

Die Variable Base ist mit der *declarative_Base()* Funktion verbunden, die aus dem *from sqlalchemy.ext.declarative* Paket stammt und in der *meta.py* Datei im *model* Verzeichnis importiert wird. Diese Funktion führt automatisch das OR-Mapping mit den Eigenschaften der definierten Klasse durch. Neben den im Listing genannten Datentypen[36] gibt es noch viele weitere und zusätzliche Einstellungen[37] in SQLAlchemy, die noch genauere Deklarationen der Klassen und die Beziehungen untereinander erlauben.

Um SQLite3 mit Pylons und SQLAlchemy zu verbinden, wird noch die Position der Datenbank auf dem Server in der genutzten **.ini* z.B. *development.ini* benötigt. Diese Datei liegt direkt im Projektverzeichnis. Folgende Information sollte in der *[app:main]* Sektion eingefügt werden, wenn sie nicht bereits vorhanden ist.

[36]SQLAlchemy Datentypen: `http://www.sqlalchemy.org/docs/core/types.html`, 21.12.2010
[37]SQlAlchemy Schema: `http://www.sqlalchemy.org/docs/core/schema.html`, 21.12.2010

```
[app:main]
sqlalchemy.url = sqlite:///%(here)s/mikroadmin.db
```

Listing 4.17: Information über die benutzte Datenbank in Pylons

Die Anweisung *%(here)s* bezeichnet dabei das Verzeichnis in der die **.ini* Datei liegt und *mikroadmin.db* der Name der SQLite3 Datei. Wenn der Pfad zur Datenbankdatei absolut sein soll, muss ein weiterer / hinzugefügt werden.

Der nächste Schritt ist es, die Klasse als Tabelle in die Datenbank zu mappen. Dafür bietet das Paster Skript eine Option an. Um das Skript erfolgreich ausführen zu können, muss im Unterverzeichnis *mikroadmin* die Datei *websetup.py* editiert werden. Um die Datenbank bei jeder Ausführung zu bereinigen und darauf hin Standardeinträge z.B. zu Testzwecken einzufügen, sollten folgenden Anweisung eingefügt werden.

```
"""Setup the Mikroadmin application"""
import logging

import pylons.test

from mikroadmin.config.environment import load_environment
from mikroadmin.model.meta import Session, Base
from mikroadmin.model.Board import Board

log = logging.getLogger(__name__)

def setup_app(command, conf, vars):
    """Place any commands to setup mikroadmin here"""
    # Don't reload the app if it was loaded under the testing environment
    if not pylons.test.pylonsapp:
        load_environment(conf.global_conf, conf.local_conf)

    # Create the tables if they don't already exist
    log.info("Creating tables")
    Base.metadata.drop_all(checkfirst=True, bind=Session.bind)
    Base.metadata.create_all(bind=Session.bind)

    log.info("Creating new entries")

    board1 = Board(name='AVR Dragon')
    board2 = Board(name='Atmel AT91SAM7')
    board3 = Board(name='MSP430')
    Session.add(board1)
    Session.add(board2)
    Session.add(board3)
    Session.commit()
```

```
32
33      log.info("Successfully setup")
```

Listing 4.18: Erstellung von Datenbanktabellen und Standardeinträgen in websetup.py

Der wichtigste Import im Listing ist das Einbinden der Board Klasse, damit auch Boardelemente in die Datenbank eingefügt werden können. Die *drop_all()* Funktion überprüft, ob bereits Tabellen existieren und löscht alle Tabellen der Modelle die im *model* Verzeichnis definiert wurden. Andere Tabellen bleiben hiervon unberührt. *create_all()* erstellt anschließend aus allen Klassenmodellen Tabellen in der Datenbank. Damit die einzelnen Einträge in die Datenbank überführt werden, sollten diese in die Datenbanksession[38] mit *add()* eingebunden sein, welche über *commit()* endgültig in die Datenbank überführt werden. Entsteht innerhalb einer Datenbanksession ein Problem, wird automatisch ein Rollback gefahren und Datenbankeinträge innerhalb dieser Session wieder gelöscht. So kann sichergestellt werden, dass zusammenhängende Anweisungen als Block in die Datenbank übertragen werden und die Datenbank somit konsistent bleibt.

Um das editierte *websetup.py* Skript auszuführen, wird folgende PasterSkript Anweisung in der Shell ausgeführt.

```
1 $ paster setup-app development.ini
```

Listing 4.19: Ausführung der websetup.py über das Paster Skript

Wenn das Skript ohne Fehler durchläuft, wird dem Aufrufer über das Log Paket die Nachricht *INFO [mikroadmin.websetup] [MainThread]Successfully setup* zurückgegeben. Ist das gegeben, kann die Manipulation der Einträge über Actions erfolgen.

4.8.4 CRUD über Actions und SQLAlchemy

Für Webapplikationen ist es wichtig den Datenbankbestand zu manipulieren. Die Manipulation der Daten geschieht in Pylons, so wie in anderen Frameworks, über die Controller. Dafür bietet SQLAlchemy die nötige Funktionalität. Um das zu verdeutlichen, werden verschiedene Actions erstellt, welche die wichtigsten Möglichkeiten der Datenmanipulation zeigen.

[38] SQLAlchemy und Sessions: `http://www.sqlalchemy.org/docs/orm/session.html`, 22.12.2010

```
1  # coding: utf-8
2
3  import logging
4
5  from pylons import request, response, session, tmpl_context as c, url
6  from pylons.controllers.util import abort, redirect
7
8  from mikroadmin.lib.base import BaseController, render
9  from mikroadmin.model.Board import Board
10 from mikroadmin.model.meta import Session
11
12 log = logging.getLogger(__name__)
13
14 class CrudController(BaseController):
15
16     def index(self):
17         # Return a rendered template
18         #return render('/crud.mako')
19         # or, return a string
20         return 'Hello World'
21
22     def create(self, boardname):
23         board = Board(name=boardname)
24         Session.add(board)
25         Session.commit()
26
27         return 'A new board with name %s was created' % boardname
28
29     def read(self):
30         boardnames = []
31         boardnames.append('<ul>')
32         [boardnames.append('<li>' + board.name + '</li>') for board in Session.query(
               Board).all()]
33         boardnames.append('</ul>')
34         return boardnames
35
36     def update(self, boardname, feld, inhalt):
37         board = Session.query(Board).filter_by(name=boardname).first()
38
39         befehl = 'board.' + feld + ' = ' + '"'+inhalt+'"'
40         exec(befehl)
41         Session.commit()
42
43         return eval('board.'+feld)
44
45     def delete(self, boardname):
46         board = Session.query(Board).filter_by(name=boardname).first()
47         Session.delete(board)
48         Session.commit()
49
```

```
50          return u'Das Board ' + boardname + u' wurde geloescht'
```

Listing 4.20: CRUD mit Pylons und SQLAlchemy

Diese Beispiele verstehen sich nur zur Veranschaulichung und sollen zeigen, wie Datenbankeinträge manipuliert werden können. Auf Sicherheit der Anwendung wurde nicht geachtet, bei Interesse kann der Pylons[39] Eintrag bei Python Security durchgelesen werden. Auch hätten hier bei den Returnangaben, auf Mako Templates im Abschnitt 4.7 zur Ausgabe weitergeleitet werden können. Worauf hier der Einfachheit halber verzichtet wurde.

Die wichtigen Anweisungen bei allen Actions, die die Datenbasis manipulieren sollen ist die *Session* Anweisung gefolgt von weiteren SQLAlchemy Ausdrücken zur Filterung bzw. Auswahl einzelner oder mehreren Boardinstanzen. Bis auf das reine Auslesen von Informationen müssen alle Sessions mittels *commit()* beendet werden, damit die Änderungen in der Datenbank erfolgen.

Der URL-Dispatcher sollte folgendermaßen ergänzt werden, um die Informationen an die Actions weiterleiten zu können.

```
1 map.connect('/{controller}/{action}/{boardname}')
2 map.connect('/{controller}/{action}/{boardname}/{feld}/{inhalt}')
```

Listing 4.21: URL-Dispatcher Konfiguration für CRUD Actions

Um z.B. ein Attribut einer Boardinstanz zu verändern, können mittels einer GET-Abfrage im Browser die Anweisung an die Actions geschickt werden. Für die die Ergänzung z.B. des *description* Feldes einer Boardinstanz, kann folgender Aufruf auf dem Entwicklungsserver erfolgen.

```
1 http://localhost:5000/crud/update/AVR Dragon/description/Super Board
```

Listing 4.22: Übergabe von GET Parameter und Daten an Actions

Die erste Information nach dem Aufruf der Action *update* ist der Name des jeweiligen Boards, *description* ist das zu manipulierende Attribut und das letzte Datum der Eintrag, welches ins ausgewählte Feld überführt werden soll. Die Übertragung der Daten zu den Actions kann auch über Formulare[40] in Pylons erfolgen. Die in Formu-

[39] Pylons Security: `http://www.pythonsecurity.org/wiki/pylons/`, 23.12.2010
[40] Pylons Formulare [10, Kap. 6, S. 91]

laren im Request übertragenen Daten sind daraufhin in der Action entweder mit *request.params['formularfeld']* oder auch mittels *request.POST['formularfeld']* für die Weiterverarbeitung zugänglich.

Pylons bietet zusätzliche Funktionalität, um den Umgang mit Formularen zu erleichtern. Darunter das *webhelpers.html.tags* Paket, um automatisch Formulare zu generieren ohne dabei HTML direkt zu benutzen. Oder mittels *formencode* einzelne Felder zu validieren bevor Daten abgespeichert werden.

4.9 JavaScript

Zur Anfangzeit von Pylons wurden die JavaScript[41] Frameworks *Script.aculo.us* und *Prototype* offiziell unterstützt. Heutzutage kann jedes Framework eingesetzt werden, welches der Entwickler bevorzugt. Unter Pylons Entwicklern sind *jQuery*, *YUI* und *ExtJS* sehr beliebt.

All die genannten Frameworks können direkt eingebunden werden, dabei wird das ausgewählte Framework heruntergeladen und ins *public* Verzeichnis der Pylons Applikation kopiert. Eine andere Möglichkeit ist, das gewählte Framework über das *CDN*[42] z.B. von Google oder Yahoo! in die Applikation in einem zentralen Template einzubinden. Wie jQuery unter HTML eingebunden werden kann, ist unter 3.12.2 nachzulesen.

4.10 Sicherheit

Das Framework schützt standardmäßig vor *XSS*, indem im Konfigurationsverzeichnis unter *mikroadmin/config/* in der Datei *einvironment.py*, der *escape* Filter aktiviert wird. Dabei werden alle Usereingaben so gefiltert, dass HTML Anweisungen in ihre Entities umgewandelt werden. Soll an bestimmten Stellen HTML Code erlaubt sein, kann diese Option mittels der *literal* Funktion aus dem Webhelpers Package aktiviert werden.

Um Formulare gegen *CSRF*[43] zu schützen, damit Formulare mittels des *webhelpers* Paketes in Templates benutzt werden können, müssen diese in der *helpers.py* Datei im *lib* Verzeichnis der Webapplikation eingebunden werden. Folgender Aufruf gilt für die

[41] Pylons and JavaScript [10, Kap. 15, S. 325]
[42] Content Delivery Framework: `http://www.cdncatalog.com/`, 27.12.2010
[43] CSRF Schutz unter Pylons `http://pylonshq.com/docs/en/1.0/helpers/#secure-form-tag-helpers`, 27.12

aktuelle Pylons Version.

```
1 from webhelpers.html.tags import *
2 from webhelpers.pylonslib import secure_form
```

Listing 4.23: Schutz vor CSRF mittels secure_form

Innerhalb eines Templates können folgende Zeilen eingebunden werden, um ein Formular und zusätzliche Felder einzubinden. Die Funktion *auth_token_hidden_field()* fügt dem Formular ein verstecktes Feld hinzu, welches im Value Attribut einen Schlüssel zur Identifizierung mit sich trägt. Dieser Wert wird bei POST Requests zur Authentifizierung der Anfrage benötigt. Ist dieser Wert nicht richtig oder nicht vorhanden, wird eine weitere Ausführung verweigert.

```
1 <html>
2 <h1>Ihre Email Adresse</h1>
3 ${h.secure_form('/test/')}
4 ${h.auth_token_hidden_field()}
5 Email Address: ${h.text('email')}
6 ${h.submit('submit', 'Absenden')}
7 ${h.end_form()}
8 </html>
```

Listing 4.24: Einbinden eines sicheren Formulars in Pylons

Um es nochmal zu verdeutlichen, der größte Schutz vor Angriffen liegt darin, Usereingaben vor weiterer Verarbeitung zu *escapen*. Unabhängig davon, ob sie per GET- oder POST-Request an den Server übermittelt werden.

4.11 Bereitstellung

Während der Entwicklungsphase wurde die Applikation in einer virtuellen Python Umgebung erstellt. Soll die erstellte Anwendung produktiv eingesetzt werden, kann das auch in der virtuellen Umgebung geschehen. Aus Sicherheitsgründen sollte die Applikation unter einem User laufen, der keine Rootrechte auf das System besitzt. Dafür kann ein neuer Useraccount erstellt werden. Dazu wird das Tool *adduser*[44] in diesem Fall genutzt.

[44] Adduser unter Linux/Ubuntu: `http://wiki.ubuntu-forum.de/index.php/Benutzer_/_Gruppen_verwalten`, 28.12.2010

Nach Erstellung eines neuen Useraccounts z.B. */home/pylonsapp* - alle weiteren Schritte beziehen sich auf diesen Account - wird *virtualenv* über *easy_install* installiert, wie im Abschnitt 4.3.1 bereits erklärt. Auch sollten weitere Schritte, wie z.B. die Installation von *pysqlite* und weitere Software, falls benötigt unter dem neuen Account durchgeführt werden. Der Einfachheit halber wird im Folgenden nur die Bereitstellung einer Pylons Applikation ohne Datenbank gezeigt.

Nach Installation und Erstellung einer virtuellen Python Umgebung z.B. unter */home/pylonsapp/pylonsenv* kann die erstellte Applikation, als Python Egg, in ein Verzeichnis unterhalb des neuen Accounts kopiert werden.

Ein Python Egg[45] der zu installierenden Applikation wird folgendermaßen generiert. Dabei wurden alle Metainformationen und Abhängigkeiten bereits von *Paste* bei der Erstellung der Webapplikation erstellt und müssen deshalb nicht selbst definiert werden.

```
1 $ cd MikroAdmin
2 $ python setup.py bdist_egg
```

Listing 4.25: Erstellung eines Python Eggs

Bei Erfolg wird innerhalb des Applikationsordners ein Verzeichnis mit dem Namen *dist* erzeugt, in diesem Ordner befindet sich eine Datei mit der Endung *.egg*. Für dieses Beispiel wird folgender Name *MikroAdmin-0.1dev-py2.6.egg* angenommen. Der Name kann je nach benutzter Python Version variieren.

Unter dem neu erstellten Useraccount kann die MikroAdmin Applikation installiert werden. Dazu kann jetzt einfach *easy_install* folgendermaßen ausgeführt werden.

```
1 $ cd /home/pylonsapp/eggs/
2 $ ~/pylonsapp/pylonsenv/bin/easy_install MikroAdmin-0.1dev-py2.6.egg
```

Listing 4.26: Installation eines Pylons Webapplikation Eggs

Wenn keine zusätzlichen Angaben gemacht werden, wird das *Egg* im *site-packages* Verzeichnis der virtuellen Python Umgebung in *MikroAdmin-0.1dev-py2.6.egg* entpackt und zusätzliche Pakete, von der die Applikation abhängt, installiert. Unter anderem Pylons und Paste.

[45] A small introduction to Python Eggs: `http://mrtopf.de/blog/en/a-small-introduction-to-python-eggs/`, 28.12.2010

Damit die Applikation mit dem *paste* Server gestartet werden kann, muss eine **.ini* Datei erzeugt werden. Da sie für Produktionszwecke genutzt werden soll, kann sie *production.ini* genannt werden. Sie sollte konsequenterweise im Verzeichnis *MikroAdmin-0.1dev-py2.6.egg* erzeugt werden, was aber nicht zwinged notwendig ist. Die **.ini* Datei wird folgendermaßen erstellt.

```
1 $ cd
2 $ ~/pylonsapp/pylonsenv/lib/python2.6/site-packages/MikroAdmin-0.1dev-py2.6.egg
3 $ ~/pylonsenv/bin/paster make-config MikroAdmin production.ini
```

Listing 4.27: Erstellung einer production.ini Datei mit paster

Da die *production.ini* für den produktiven Einsatz genutzt wird, sollte aus Sicherheitsgründen in der Datei unter *[DEFAULT]* die Variable *debug* auf *false* gestellt werden. Um festzustellen, ob die Applikation ausgeliefert werden kann, sollte sie mit *paste* gestartet werden.

```
1 $ ~/pylonsenv/bin/paster serve production.ini
```

Listing 4.28: Test der MikroAdmin App für die Bereitstellung

Wenn die Applikation erfolgreich geladen wird und im Browser aufrufbar ist, kann der Apache Webserver mittels *mod_wsgi* für die Bereitstellung sorgen. Die nächsten Schritte setzen eine erfolgreiche Installation dieser beiden Pakete voraus.

4.11.1 Pylons unter Apache

Es gibt verschiedene Arten Pylons und Apache zu verbinden, eine Möglichkeit ist Apache als Proxy[46] zum Paste Server zu nutzen. Dabei wird Apache dazu genutzt Anfragen an einen Python Server wie Paste weiterzuleiten. So kann auf bewährte Weise Paster-Skript für die Einstellungen und somit für eine bessere Kontrolle über die Webapplikation genutzt werden. Gleichzeitig erhöht sich die Sicherheit, da Apache als Schleuse benutzt wird, der sich jahrelang als bewährter Webserver ausgezeichnet hat. Der Nachteil dabei ist, dass verschiedene Server gewartet werden müssen und somit nicht immer klar ist, wo genau bei Problemen die Fehler liegen können.

[46] Apache als Proxy für Paste [10, Kap. 21, S.494]

Eine neuere und elegantere Variante ist es, wie bereits erwähnt und auch unter Django demonstriert, eine Pylons Applikation durch Apache und mod_wsgi auszuliefern. Dafür ist es nötig in der Apache Konfiguration einen *Virtual Host* zu erzeugen, der ein *WSGI* Skript ausführt. Das *WSGI* Skript dient somit als Verbindung zwischen Apache und Pylons. Ein solches Skript für die MikroAdmin Applikation kann folgendermaßen aussehen.

```
1 # Virtual env site-packges
2 import site
3 site.addsitedir('/home/pylonsapp/pylonsenv/lib/python2.6/site-packages')
4
5 # Laden der Pylons Applikation
6 from paste.deploy import loadapp
7 application =
8 loadapp('config:/home/pylonsapp/pylonsenv/lib/python2.6/site-packages/MikroAdmin-0.1dev
      -py2.6.egg/production.ini')
```

Listing 4.29: WSGI Skript für Pylons und Apache

Durch *addsetdir()* wird sichergestellt, das auch *Eggs*, die in den **.pth* aufgelistet sind auch in den Python PATH hinzugefügt werden. Um schließlich die Webapplikation zu starten, wird die Applikation mit den Einstellungen in der *production.ini* geladen. Damit Requests vom Client an die Actions weitergegeben werden und diese ausgeführt werden können, beziehungsweise statischer Code zurückgegeben wird, muss Apache entsprechend konfiguriert sein. Die Zeilen im Listing 4.30 können dafür im Apache eingetragen werden. Unter Ubuntu wird die Datei *default* im */etc/apache2/sites-available/* Verzeichnis benutzt. Dateien in diesem Verzeichnis enthalten Konfigurationsanweisungen für Virtual Hosts. Es kann auch eine neue Virtual Host Datei in *sites-available* manuell erstellt werden, diese sollte anschließend mit *sudo a2ensite name*[47] im *sites-enabled* Verzeichnis eingebunden sein. Nur Virtual Hosts, die hier verlinkt sind, werden von Apache berücksichtigt.

```
1 <VirtualHost *:80>
2 WSGIScriptAlias / /home/pylonsapp/apache/mikroadmin.wsgi
3
4     <Directory '/home/pylonsapp/apache'>
5         Order deny,allow
6         Allow from all
7     </Directory>
```

[47] Virtual Hosts in Apache aktivieren: `http://www.debian-administration.org/articles/207`, 28.12.2010

```
8
9 </VirtualHost>
```

Listing 4.30: Virtual Host für eine Pylons Applikation

Der *WSGIScriptAlias* Direktive werden zwei Informationen mitgeteilt. Dabei steht an erster Stelle die URL, die vom Client aufgerufen werden muss, um die Pylons Applikation anzusprechen. In diesem Fall wird die Applikation an oberster Stelle eingehängt, symbolisiert durch den /. Die zweite Information ist der Ort, an dem das *WSGI* Skript abgelegt wurde. In der *Directory* Direktive wird zusätzlich das *WSGI* Skript Verzeichnis sichtbar gemacht.

Die eben durchgeführten Schritte sollten dazu führen, dass die Applikation bereitgestellt wird. Damit der Apache User *www-data* auf die Verzeichnisse der Applikation zugreifen darf, sollte die *pylonsapp* Gruppe dem www-data User hinzugefügt werden. Mit dem Befehl *sudo usermod -G pylonsapp www-data* wird das unter Ubuntu ermöglicht.

Da bei der Bereitstellung Daten von der Anwendung generiert werden, z.B. werden Mako Templates in Python Code umgewandelt, ist es nötig sicherzustellen, dass der *data* Ordner in der Applikation gelesen und beschrieben werden kann. Das sollte auch der *www-data* User dürfen, weil die Applikation unter diesem User läuft. Mit *chmod -R g+rw data* kann das durchgeführt werden.

4.12 Fazit

Pylons ist ein sehr flexibles Webframework. Vor allem dadurch, dass bereits bewährte Softwarepakete durch den Glue Code verbunden werden können. So haben Entwickler, die einzelne unterstützte Softwarepakete kennen, einen großen Vorteil. Zwar bietet das Buch *The Definitive Guide to Pylons*[48] einen sehr guten Einstieg in das Framework und vermittelt auch Expertenwissen, ist aber an manchen Stellen teilweise veraltet. Zu nennen ist hierbei z.B. das Kapitel 7 über das Mapping von Klassen und Tabellen. Da sollte das Online Manual in der aktuellen Version 1.0[49] gegengelesen werden.

Da die vielfältigen Einzelkomponenten erst verstanden werden müssen, und darüber hinaus auch wie sie miteinander durch Pylons zu verbinden sind, ist Pylons kein Framework für Python Neulinge.

[48] The Definitive Guide to Pylons [10]
[49] Pylons Online Dokumentation: `http://pylonshq.com/docs/en/1.0/`, 29.12.2010

Dafür ist die Bereitstellung von Applikationen in einer Produktivumgebung etwas einfacher als bei Django. Zwar ist Convention over Configuration gut, wenn es darum geht schnell eine Applikation aufzubauen. Aber wenn Probleme auftauchen, kann diese *Magie* hinderlich sein.

Im Grunde ist Pylons ein solides Framework, mit dem Anwendungen schnell entwickelt werden können und das dem Entwickler keine Vorgaben macht, welche Pakete für die Webentwicklung zu benutzen sind. Erfahrungen mit Python und Paketen zur Erstellung von Applikationen erleichtern den Einstieg. Dies macht auch eine gezielte Auswahl von Bibliotheken für die Erstellung von Webapplikationen einfacher.

Kapitel 5.0

Python Webframeworks im Vergleich

5.1 Vergleich

Wie eingangs erwähnt, existieren in der Python Welt eine Menge an Webframeworks, die kleinen und auch sehr großen Ansprüchen gerecht werden. Django und Pylons wurden im Detail besprochen, weil sie die wichtigsten Zweige bei der Entwicklung von Webapplikationen unter Python darstellen. Django mit seinen selbst entwickelten Tools die out of the box funktionieren und Pylons mit dem mächtigen Paste Server und Konfigurationsskript PasterSkript, auf die weitere Frameworks aufbauen.

Hier in Abbildung 5.1 wird dargestellt, wie verschiedene Python Webframeworks im Zusammenhang stehen.

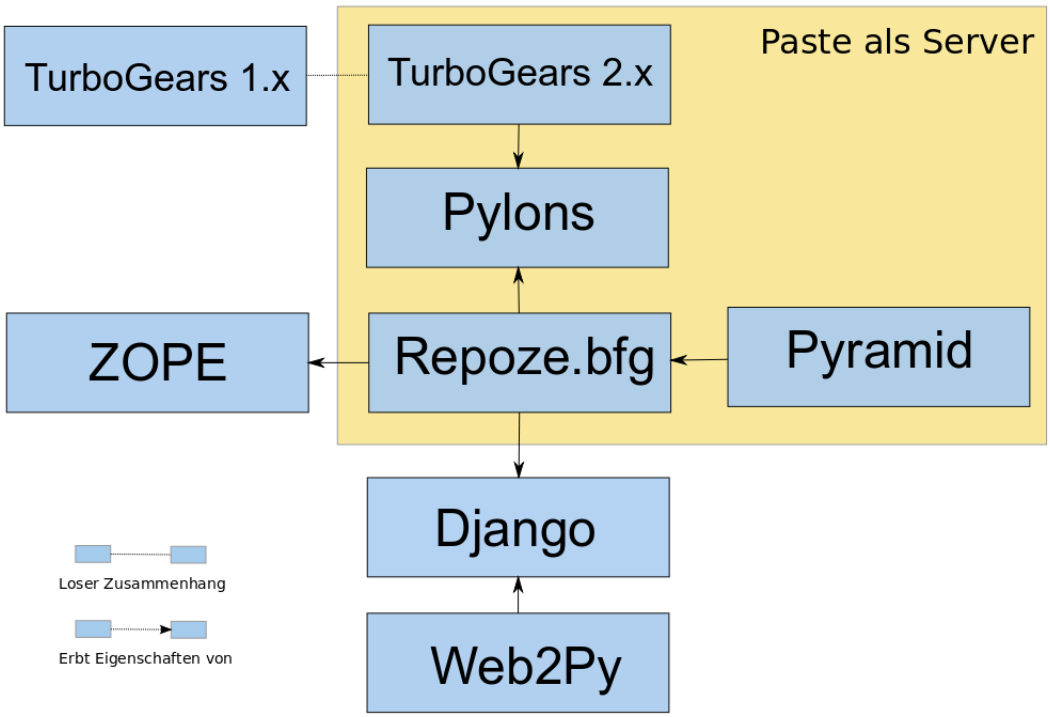

Abbildung 5.1: Verbindung verschiedener Frameworks untereinander

Noch zu ergänzen ist, dass Repoze.bfg an vielen Stellen Django ähnelt. Herauszustellen ist, dass Controller in diesen Frameworks Views genannt werden und Views Templates. Das kann an manchen Stellen verwirrend sein.

Der Vollständigkeit halber soll an dieser Stelle erwähnt werden, dass natürlich nicht alle Frameworks aus der Python Welt in weiterer Tiefe in dieser Arbeit besprochen werden können.

Zope[1] wird momentan hauptsächlich für CMS[2] genutzt und liegt in der Version 3 vor und wird BlueBream[3] genannt. In einem Atemzug ist Plone[4], als bekannteste Aplikation aus Zope zu nennen, das eine große Verbreitung genießt.

Web2py[5] als *Zero Configuration* Framework ist hauptsächlich durch Ruby on Rails und Django inspiriert worden. Es bringt alle nötigen Pakete mit und kann sogar von einem USB-Stick aus, ohne weitere Konfiguration, ausgeführt werden. Es ist laut Entwicklermeinung einfacher zu lernen als Django und bringt auch eine eigene Administrationsoberfläche mit. So können Anpassungen vollständig über den Browser stattfinden und somit müssen auf der Shell keine weiteren Einstellungen vorgenommen werden, wenn dies nicht gewünscht ist.

Für ausführlichere Informationen und einen Überblick über Webframeworks in der Python Welt, wird zur Abrundung dieser Arbeit auf das PythonWiki[6] verwiesen.

5.1.1 Django

Da der Fokus der Mikroadmin Applikation auch im Zentrum dieser Arbeit steht, ist der Django Teil im Abschnitt 3 sehr ausführlich und zeigt die gängigsten Techniken, die unter diesem Framework eingesetzt werden. Da Django ein Full-Stack Framework ist und die wesentlichen Teile, wie die Django ORM oder der URL-Dispatcher, unter dem Schirm von Django entwickelt werden, ist eine reibungslose Arbeitsweise der Komponenten fast garantiert.

Ein Entwickler, der in die Welt von Python und Web eintauchen will, bekommt ein Framework mit guter Dokumentation und ein System, welches out of the box funktio-

[1] Nutzung von Zope: `http://wiki.python.org/moin/ContentManagementSystems`, 10.01.2011
[2] CMS: Content Management System
[3] BlueBream: `http://bluebream.zope.org/`, 10.01.2011
[4] Plone: `http://plone.org/`, 10.01.2011
[5] web2py: `http://web2py.com/`, 10.01.2011
[6] Python Webframeworks: `http://wiki.python.org/moin/WebFrameworks`, 10.01.2011

niert. Die einzelnen Komponenten dienen zur schnellen Entwicklung von Webapplikationen, die darauf fokussiert sind, viele CRUD Anfragen konsistent und zuverlässig zu bewältigen und Ergebnisse dem Benutzer zu präsentieren. Dies reicht für die meisten Webanwendungen. Sollen speziellere Wünsche erfüllt werden, kann Django an seine Grenzen stoßen, muss es aber nicht. Vor allem dadurch, da auch Fremdpakete, wie SQLAlchemy, mit Django benutzt werden können. Ein weiterer großer Vorteil des Frameworks ist die breite Community Basis die es besitzt, vor allem auch wegen der sehr guten Dokumentation.

Django im Überblick	
Art	MTV Full-Stack
OR Mapper	Django ORM
Template	Django Templateengine
Controller	Views genannt: Aufrufbare Funktionen
URL Dispatching	Django URL Dispatcher
Server	runserver (einfacher Entwicklungsserver) / Anbindung an Apache unter anderem über WSGI möglich
Dokumentation	Sehr gut
Setup Tool	django-admin.py and manage.py
Geeignet für	Kleine, mittel bis große Webanwendungen
Hauptentwickler	Lawrence Journal-World
Aktuelle stabile Version	1.2.4 / 22.12.2010
Website	`http://www.djangoproject.com`
Lizenz	BSD
JavaScript Unterstützung	jQuery für die Adminoberfläche / Alle gängigen
Besonderheiten	Administrationsoberfläche, Autorisierung, Security Features als einfach zuschaltbare Apps. Sehr gut bei Applikationen, die wie CMS Systeme funktionieren sollen.

Tabelle 5.1: Django im Überblick

5.1.2 Pylons

Als Vertreter eines Glue Code Frameworks wurde Pylons im Abschnitt 4 für einen tieferen Einblick ausgesucht. Vor allem deswegen, weil andere Webframeworks dieser Art auf Pylons basieren. Darunter fallen TurboGears 2 und Pyramid.

Pylons ist sehr flexibel, weil es viele Komponenten zur gemeinsamen Arbeit verbinden kann. So werden dem Entwickler keine Fesseln angelegt und die Freiheit gegeben, aus bewährten Paketen die passenden auszusuchen. Mit den Empfehlungen für Pakete, die von den Entwicklern für den Einsatz von Pylons gemacht werden, kann nichts falsch gemacht werden. Weil diese in der Python Welt zur *state of the art* gehören oder gut mit dem Framework zusammenarbeiten. Vor allem SQLAlchemy und Mako werden bei Pylons Applikationen oft eingesetzt und können auch in anderen Projekten - auch Web fremde - eingesetzt werden. Wer aus der *Ruby on Rails* Welt kommt und sich mit CoC[7] anfreunden kann, ist bei Pylons gut aufgehoben. Zwar ist die Einarbeitung in Pylons nicht schwieriger als bei Django, es ist aber wesentlich einfacher, wenn der angehende Webentwickler Erfahrung in Python einbringen kann. Die Dokumentation ist zweckdienlich, bei Spezialaufgaben oder Fremdpaketen wird auf fremde Dokumentation verwiesen, vor allem beim Gebrauch von SQLAlchemy, welches vorbildlich dokumentiert ist.

Pylons im Überblick	
Art	MVC Glue Code zur Verbindung der Einzelteile
OR Mapper	SQLAlchemy / SQLObject
Template	Mako / Genshi / Jinja (alle out of the box)
Controller	Actions genannt: Aufrufbare Methoden
URL Dispatching	Routes (RoR entlehnt)
Server	Paste / Anbindung an Apache unter anderem über WSGI möglich
Dokumentation	Zweckmäßig / SQLAlchemy hervorragend
Setup Tool	Paste Script (paster)
Geeignet für	Mittel bis große Webanwendungen

[7] CoC: Convention over Configuration

Pylons im Überblick	
Hauptentwickler	Ben Bangert, James Gardner
Aktuelle stabile Version	1.0 / 28.05.2010
Website	`http://pylonshq.com`
Lizenz	BSD
JavaScript Unterstützung	Alle gängigen
Besonderheiten	Flexibilität bei der Auswahl der Komponenten / Empfohlene sind state of the art

Tabelle 5.2: Pylons im Überblick

5.1.3 TurboGears 1.x

In der neuesten Version benutzt TurboGears 1.1 als MVC[8] Framework bewährte Komponenten. Der zentrale Knoten ist hierbei CherryPy[9], der alle ankommenden Requests abfängt und an Controller weiterleitet. Dabei kann CherryPy als eigenständiger Server arbeiten oder über einen vorgelagerten Server z.B. Apache oder lighttpd über WSGI aufgerufen werden.

Die entsprechenden Requests werden an Controller übergeben, die Modelle manipulieren können. Weiterhin können die Controller Daten aus den Modellen anfordern und diese zur Darstellung an eine Templateengine zur Darstellung weiterleiten. Daraufhin wird der zurückgelieferte HTML-Code von den Controllern an den Client zurückgeliefert.

Für die Persistenz können alle Datenbanken genutzt werden, die auch SQLAlchemy unterstützt, unter anderem auch SQLite oder PostgreSQL.

Zur Erstellung von HTML oder anderem angeforderten textuellen Content für das Web, wie z.B. XML, wird Genshi[10] genutzt. Die asynchrone Kommunikation mittels Ajax[11] und Manipulation des DOM Baums übernimmt Mochikit, welches out of the box

[8] Model View Controller Pattern in TurboGears 1.x: `http://docs.turbogears.org/1.1/BigPicture`, 4.11.2011
[9] CherryPy: `http://cherrypy.org/`, 3.01.2011
[10] Genshi: `http://genshi.edgewall.org/`, 3.01.2011
[11] Defining Ajax: `http://www.adaptivepath.com/ideas/essays/archives/000385.php`, 4.01.2011

funktioniert. Durch die Kompatibilität von Mochikit zu anderen Bibliotheken können weitere parallel eingebunden werden.

Die empfohlene Art TurboGears 1.1 zu installieren, ist es eine *virtualenv* Umgebung zu erstellen[12] und mittels *easy_install* TurboGears in diese nachzuladen. Die Erstellung einer virtuellen Python Umgebung wird im Abschnitt 4.3.1 gezeigt. Durch die Installation per *easy_install TurboGears* wird die aktuellste Version der 1.x Reihe heruntergeladen und Abhängigkeiten aufgelöst.

Für das Mapping benötigter Klassen auf Datenbanktabellen kann mit *easy_install SQLAlchemy* zusätzlich installiert werden.

Um ein neues Projekt zu erstellen, wird das Tool tg-admin benötigt, welches der TurboGears Installation beiliegt. Um das Mapping der Klassen und Tabellen einfacher zu gestalten, empfiehlt die Dokumentation *Elixir*[13]. Dabei handelt es sich um einen Wrapper für SQLAlchemy mit dem Ziel die *ActiveMapper* Erweiterung und das *TurboEntity* Projekt in SQLAlchemy zu ersetzen.

Durch den Befehl *tg-admin quickstart* wird im ausgewählten Verzeichnis anschließend ein neues Projekt erstellt, wobei nach der Ausführung ein passender Name und weitere Einstellungen erfragt werden. Um das so entstandene Projekt zu starten, wird automatisch ein Skript *start-projektname.py* generiert. Nach dem Starten des Python Skriptes kann nun unter *http://localhost:8080* die Startseite des Projekts betrachtet werden.

Controller werden in der Datei *controllers.py* entwickelt. Dabei werden Klassen erstellt, die in der URL automatisch als Verzeichnisse übersetzt werden. Klassenmethoden die als Controller fungieren, werden vom URL-Dispatcher wiederum als jeweilige Unterverzeichnisse gemapped, die mit dem Controllernamen angesprochen werden müssen. Requestdaten werden an die Controller weitergegeben, solange die Anzahl der übermittelten Daten übereinstimmt. Ansonsten wird eine Fehlermeldung ausgegeben. Mit Hilfe einer *default* Methode können nicht vorhandene Controller abgefangen werden, um z.B. eine individuelle Fehlermeldung auszugeben. Durch ein Slash getrennt, können bei GET-Requests, der Reihenfolge nach, Daten an die Controller übergeben werden. Bei POST-Requests gewohnt über benannte Formularfelder.

TurboGears 1.x ist mittels *virtualenv* und *easy_install* schnell installiert und durch das Konfigurationstool *tg-admin* erfolgt das Setup sehr leicht.

[12]TurboGears Installation: `http://docs.turbogears.org/1.1/Install`, 4.01.2011
[13]Elixir: `http://elixir.ematia.de/trac/wiki`, 4.11.2011

TurboGears 1.x im Überblick	
Art	MVC Full-Stack mit Glue Code zur Verbindung der Einzelteile
OR Mapper	SQLAlchemy / SQLObject
Template	Genshi
Controller	Aufrufbare Methoden die, mittels Decorator nach außen freigegeben werden
URL Dispatching	Object Dispatching via CherryPy durch Klassen und Methoden
Server	CherryPy
Setup Tool	tg-admin
Dokumentation	Zweckmäßig. Wenig deutschsprachige Literatur
Geeignet für	Kleine, mittel bis große Webanwendungen
Hauptentwickler	Florent Aide
Aktuelle stabile Version	1.1 / 4.10.2009
Website	`http://turbogears.org/about/`
Lizenz	MIT
JavaScript Unterstützung	Mochikit (out of the box) / Einbindung anderer Bibliotheken möglich
Besonderheiten	Schnelle Entwicklung von Applikationen. Das Framework bietet Defaulteinstellungen und bietet eine Auswahl von Komponenten an, die reibungslos miteinander arbeiten. Diese können bei Bedarf durch andere Teile ausgetauscht werden. JavaScript und Ajax Unterstützung out of the box. CherryPy ist das Herz des Frameworks.

Tabelle 5.3: TurboGears 1.x im Überblick

5.1.4 TurboGears 2.x

Das TurboGears 2.x Framework wirkt frischer im Vergleich zu seinem Vorgänger. Als Persistenzschicht wurde SQLAlchemy beibehalten, welche das Framework weiterhin befähigt eine Vielzahl von Datenbanken zu unterstützen und ein OR-Mapping ermöglicht. TurboGears 2.x setzt nicht mehr auf CherryPy, da die Integration[14] von CherryPy 3 sich als zu kompliziert erwiesen hat, weil sich viele Änderungen zwischen CherryPy 2 und 3 ergeben haben. Auch ist TurboGears 1.x zu sehr verdrahtet mit der CherryPy 2 Konfiguration, was einen Austausch des Servers sehr schwierig macht.

Ein zentraler Grund auf Pylons aufzusetzen, ist die gute WSGI Integration, die dieses Framework bietet. Auch die Möglichkeit der Benutzung von *Routes* zum URL-Dispatching spielte eine wichtige Rolle. Der Hauptunterschied zwischen TurboGears 2.x und Pylons liegt aber darin, dass TurboGears ein Full-Stack mit gegebenen Standardkomponenten anbieten möchte. Die Argumentation liegt darin, dass mit vorgegebenen Komponenten der Glue Code und somit die Integration des gesamten Frameworks und darüber hinaus weitere Tools besser auf das Framework abgestimmt werden können. Im Gegensatz zu Pylons, welches so agnostisch wie möglich bleiben will, um so viele Komponenten wie möglich einbinden zu können.

Für die View wird weiterhin auf Genshi gesetzt, als Nachfolger von Kid. Auch deswegen, weil die Community von Genshi größer ist und somit eine stetige Entwicklung gewährleistet wird. Darüber hinaus können sich die TurboGears Entwickler sich so besser auf die Integration des Frameworks konzentrieren ohne viel an Genshi arbeiten zu müssen.

Für die Kommunikation zwischen Client und Server über Formulare, wird ToscaWidgets[15] standardmäßig benutzt. Über diese Library können Formulare, die als Klassen definiert werden, an z.B. Genshi Templates, durch einen Dictionary Eintrag weitergegeben werden. Dabei können Formularfelder mit benutzerfreundlichen JavaScript Widgets für die Datumseingabe definiert sein. Darüber hinaus können Formulare, vor der Speicherung von Daten auf der Serverseite, auf Validität hin überprüft werden. Über ToscaWidgets[16] können verschiedene JavaScript Bibliotheken eingebunden werden, die

[14] Antworten auf das neue TurboGears Konzept: `http://www.turbogears.org/2.1/docs/main/WhatsNew.html#why-not-merge-with-pylons`, 5.01.2011

[15] ToscaWidgets: `http://toscawidgets.org/`, 7.01.2011

[16] Einbindung verschiedener JavaScript Bibliotheken in ToscaWidgets: `http://turbogears.org/2.1/docs/main/GlobalJSLib.html`, 7.01.2011

als Default für das ganze Projekt gelten, oder nur für bestimmte Templates.

Zu Testzwecken oder auch für den produktiven Einsatz empfiehlt es sich *virtualenv* zu benutzen. Informationen zur Installation können im Abschnitt 4.3.1 nachgelesen werden. Nach der Ausführung des *activate* Skriptes kann in der Shell folgender Befehl für die Installation von TurboGears 2.1 eingegeben werden.

```
1 $ easy_install -i http://www.turbogears.org/2.1/downloads/current/index tg.devtools
```

Listing 5.1: Installation von TurboGears 2.1

Die *-i* Option von *easy_install* wird benutzt, um mittels einer URL den Ort anzugeben, wo sich das zu installierende Paket befindet. Um ein einfaches Projekt zu erzeugen kann *paster quickstart mikroadmin* benutzt werden, wobei die letzte Anweisung des Befehls den Namen des neu erzeugten Projekts darstellt. Danach können alle Einstellungen mit Default übernommen werden. Mit folgender Anweisung wird das neu erstellte Projekt unter *site-packages* als *egg-link* aufgenommen und zusätzliche Pakete installiert, unter anderem ToscaWidgets.

```
1 $ cd mikroadmin
2 $ python setup.py develop
```

Listing 5.2: Nachinstallation zusätzlicher TurboGears 2.x Pakete

Danach kann *nosetests* im Verzeichnis ausgeführt werden, um festzustellen, ob das Projekt richtig initialisiert wurde.

Bei mehrmaliger Generierung des Projekts und anschließendem *nosetest* erfolgte die gleiche Fehlermeldung. Diese meldete, dass *Genshi* nicht installiert sei. Mit *easy_install genshi* wird das Paket nachinstalliert und *nosetests* läuft ohne Fehlermeldung durch.

TurboGears 2.x im Überblick	
Art	MVC Full-Stack mit Glue Code zur Verbindung der Einzelteile
OR Mapper	SQLAlchemy
Template	Genshi
Controller	Aufrufbare Methoden in verschiedenen Klassen gruppiert. Weiterleitung auf Fremdmethoden innerhalb der Klassen möglich

TurboGears 2.x im Überblick	
URL Dispatching	Object Dispatching (nicht mehr über CherryPy) / Routes (Pylons) / RESTful Controller Dispatching[17]
Server	Paste
Setup Tool	PasterSkript (paster)[18]
Dokumentation	Zweckmäßig. Wenig deutschsprachige Literatur
Geeignet für	Kleine, mittel bis große Webanwendungen
Hauptentwickler	Mark Ramm[19]
Aktuelle stabile Version	2.1 / 19.11.2010
Website	`http://turbogears.org/2.1/`
Lizenz	MIT
JavaScript Unterstützung	Alle gängigen / Einbindung von Bibliotheken auch über ToscaWidgets möglich
Besonderheiten	TurboGears 2.x bietet aktuellere Komponenten als seine Vorgängerversion und basiert nicht mehr auf CherryPy, sondern auf Pylons. Das Framework bietet ein Default Komponentenstack, der reibungslos arbeitet und von TurboGears zusammengehalten wird. Teile können weiterhin gegen andere ausgetauscht werden.

Tabelle 5.4: TurboGears 2.x im Überblick

[17] Controller Dispatching TurboGears 2.x: `http://turbogears.org/2.1/docs/modules/tgcontroller.html` 5.01.2011

[18] PasterSkript in TurboGears 2.x `http://www.turbogears.org/2.1/docs/main/WhatsNew.html#command-changes`, 5.01.2011

[19] TurboGears Projektleiter: `http://docs.turbogears.org/TurboGearsTeam`, 5.11.2011

5.1.5 Repoze.bfg

Repoze.bfg ist ein minimalistisches Webframework. Es ist inspiriert durch Pylons, Django und Zope und bringt out of the box das nötigste mit, um Webapplikationen zu schreiben. Die Entwickler kategorisieren das Framework als MVC, obwohl es eigentlich nur Models und Views zur Erstellung von Applikationen bereitstellt und keine Controller im eigentlichen Sinne benutzt. Selbst bezeichnen die Entwickler das Framework als MV[20] und betrachten die View so, wei die Django Entwickler.

Da das Projekt an sich nicht sehr groß ist und jede undokumentierte Funktion als Bug unter Repoze.bfg gilt, ist die Dokumentation[21] auf dem Niveau von Django. An Literatur existiert nur ein einziges englischsprachiges Buch[22] für die Version 1.2 vom Hauptentwickler Chris McDonough zum Framework und die Dokumentation auf der Projektseite.

Repoze.bfg wird von Googles App Engine[23] unterstützt und kann unter Jython[24] betrieben werden. Eine Warnung vorweg, Chameleon, die Standard Template Engine von Repoze.bfg funktioniert unter Jython nicht. Hier kann z.B. auf Jinja[25] ausgewichen werden.

Um Funktionen bei Repoze.bfg für die Ausführung anzumelden existieren verschiedene Möglichkeiten[26]. Die imperative Art die Funktionen anzumelden ist pythonischer, da nur Methoden aufgerufen werden müssen, damit eine Anmeldung erfolgt. Die deklarative Art ist XML entlehnt und benutzt ZCML, wie es auch unter Zope benutzt wird. Dabei werden in einer ausgelagerten Datei Anweisungen geschrieben, die Makros darstellen und in Funktionsaufrufe übersetzt werden. Der Vorteil an der deklarativen Methode ist, dass keine Dateien, die Geschäftslogik beinhalten angefasst werden müssen und Drittanbieter die ZCML Dateien einfach erweitern können. Daneben gibt es noch die Möglichkeit Dekoratoren zu benutzen und eine Scan Funktion auszuführen, die automatisch die Anmeldung der Funktionen für den Entwickler übernimmt. Repoze.bfg stellt es frei, die für

[20] MV: Model View
[21] Repoze.bfg Dokumentation: `http://bfg.repoze.org/documentation`, 10.01.2011
[22] The repoze.bfg Web Application Framework: `https://www.createspace.com/3422488`, 10.01.2011
[23] Repoze und App Engine: `http://docs.repoze.org/bfg/current/tutorials/gae/index.html#appengine-tutorial`, 10.01.2011
[24] Jython: `http://www.jython.org/`, 10.01.2011
[25] Jinja Template Engine: `http://wsgiarea.pocoo.org/jinja/`, 10.01.2011
[26] Repoze.bfg Konfiguration: `http://docs.repoze.org/bfg/current/narr/configuration.html`, 10.01.2011

den Entwickler geeignetste Methode für die Anmeldung der Views in die Applikation zu benutzen und auch eine Mischung der genannten Varianten ist möglich.

Die Installation[27] kann auch hier innerhalb einer Python Umgebung. Die Vorgehensweise wird im Abschnitt 4.3.1 gezeigt. Dabei muss lediglich folgender Befehl ausgeführt werden *easy_install -i http://dist.repoze.org/bfg/current/simple repoze.bfg*. Es werden alle nötigen Pakete für die Arbeit aus dem aktuellen Zweig geladen und installiert.

Ein Projekt zu erstellen ist sehr einfach. Innerhalb eines Verzeichnisses wird z.B. die Datei *mikroadmin.py* erzeugt und folgender Inhalt eingebunden.

```
1  from webob import Response
2  from paste.httpserver import serve
3  from repoze.bfg.configuration import Configurator
4
5  def mikroadmin(request):
6      return Response('Willkommen bei Mikroadmin')
7
8  def fehlermeldung(request):
9      return Response('Fehler 404')
10
11 if __name__ == '__main__':
12     config = Configurator()
13     config.begin()
14     config.add_view(mikroadmin)
15     config.add_view(fehlermeldung, name='404')
16     config.end()
17     app = config.make_wsgi_app()
18     serve(app, host='0.0.0.0')
```

Listing 5.3: Die einfachste Art eine Repoze.bfg Applikaton zu erstellen

Die *webob*[28] Bibliothek wird als Grundlage für Requests und Responses bei Repoze.bfg genutzt. Dabei steht die *Response* Klasse im Mittelpunkt, die eine WSGI Antwort erzeugen kann. Die Funktionen können ein Request-Objekt für die Weiterverarbeitung aufnehmen, in der alle Requestinformationen aufgeführt sind. Das Skript kann einfach per *python mikroadmin.py* ausgeführt werden. Dabei wird eine Paste Server Instanz aufgerufen, welche die zuvor imperativ definierten Konfigurationen beim Start aufruft. Standardmäßig wird der Port 8080 genutzt und die Applikation kann anschließend über localhost:8080/ aufgerufen werden. Wenn diese URL aufgerufen wird, erscheint die zuletzt mit *config.add_view(mikroadmin)* eingebundene Funktion. Alle anderen müssen

[27] Repoze.bfg Installation: `http://docs.repoze.org/bfg/current/narr/install`, 10.01.2011
[28] Webob Bibliothek: `http://pythonpaste.org/webob/`, 10.01.2011

einen Bezeichner *name* tragen, um nicht beim Aufruf auf oberste Ebene angezeigt zu werden. Der Bezeichner ist gleichzeitig der Verbindungsname zwischen URL und Funktion.

Diese aufrufbaren Funktionen werden bei Repoze.bfg View Callables gennant, dabei können diese Views auch Methoden innerhalb von Klassen sein. Weitere Einstellungen, die in der *add_view()* Funktion gemacht werden, werden *predicates* genannt. Je mehr predicates als Eigenschaften definiert werden, desto spezifischer kann der Aufruf einer Funktion bzw. Methode konfiguriert werden. So wird die View, die am ehesten zum Aufruf in der URL passt, aufgerufen. So spielt die Reihenfolge in der Konfiguration keine Rolle, außer wenn der Aufruf auf verschiedene Views passen würde, wird die zuletzt hinzugefügte View aufgerufen.

Nachdem Paste als Server benutzt wird und somit auch PasterScript zur Verfügung steht, kann ähnlich wie bei Pylons oder TurboGears 2 *paster* zur Generierung eines Projekts benutzt werden. Natürlich ist das bei einfachen Projekten nicht nötig, wird aber empfohlen, da weitere Einstellungen vorgenommen werden und die Benutzung des PasterSkripts während der Entwicklung ermöglicht wird.

Um ein einfaches Projekt zu generieren, das nur *traversing* und kein *routes* verwendet und ohne Datenbankanbindung auskommen soll, kann das *bfg_starter* Template verwendet werden. Mit folgendem Befehl wird das Projekt erzeugt.

```
1 $ paster create -t bfg_starter
```

Listing 5.4: Erstellung eines einfachen Repoze.bfg Projekts mit PasterSkript

Es stehen weitere Templates zur Verfügung, die auch die Benutzung von ZODB oder von SQLAlchemy unterstützte Datenbanken für die Persistenz ermöglichen. Dabei werden abhängig vom Template verschiedene Voreinstellungen in den von PasterSkript erzeugten Dateien vorgenommen, um die gewünschte Funktionalität zu gewährleisten.

Repoze.bfg benutzt PasterSkript für seine Konfiguration und Ausführung und *setuptools* für die Erstellung von *Eggs* und somit für die Auslieferung von Projekten. Daher ähneln sich weitere Schritte zur Bereitstellung und Auslieferung von Projekten mit Hilfe von WSGI und Apache. Diese Schritte sind im Abschnitt 4.4 erklärt.

Das Framework eignet sich out of the box vor allem für kleine Projekte und Anfänger. Auch für Entwickler, die schnell produktiv sein möchten und sich nicht unbedingt in verschiedene Webtechniken vor der Benutzung eines Webframeworks einarbeiten wollen.

Aber durch die Möglichkeit des Einsatzes von Paste, SQLAlchemy, Routes und ZODB werden auch die Wünsche erfahrener Entwickler bedient und die Entwicklung anspruchsvoller Anwendungen ermöglicht.

Repoze.bfg im Überblick	
Art	MV Full-Stack mit Glue Code zur Verbindung der Einzelteile
OR Mapper	keine out of the box / Einbindung von SQLAlchemy möglich
Template	Eigene: Chameleon / Andere Template Engines möglich
Controller	Als View Callable bezeichnet
URL Dispatching	Traversal (automatisch über Methoden und Funktionen) / Routes
Server	Paste
Setup Tool	PasterSkript (paster)
Dokumentation	Sehr gut. Wenige weitere Literatur
Geeignet für	Kleine, mittel bis große Webanwendungen
Hauptentwickler	Chris McDonough
Aktuelle stabile Version	1.2 / 10.02.2010
Website	`http://bfg.repoze.org/`
Lizenz	Eigene[29] BSD like
JavaScript Unterstützung	Alle gängigen
Besonderheiten	Repoze.bfg bietet ähnlich zu Django ein Full-Stack Framework an, das eigene Tools für die Entwicklung von Webapplikationen mitbringt und sehr einfach zu installieren ist. Soll die Applikation wachsen, steht das Framework auf den Schultern von Pylons und kann mit all seinen Tools benutzt werden

Tabelle 5.5: TurboGears 2.x im Überblick

[29] Repoze.bfg Lizenz: `http://repoze.org/license.html`, 10.02.2011

5.2 Welches Framework ist die richtige Wahl?

Die genannten Webframeworks sind mittlerweile die bekanntesten in der Python Welt. Sie ermöglichen alle die Erstellung kleiner bis großer Anwendungen mit geringem Konfigurationsaufwand. Sie haben alle gemein, dass sie auf WSGI für die Verbindung nach außen aufsetzen und somit eine stabile Basis besitzen.

Langsam zeichnet sich der Trend ab, dass in Zukunft Django als das bekannteste der gezeigten Frameworks Konkurrenz bekommt. Pylons und Repoze.bfg haben sich zusammengetan, um ein neues Webframework zu erstellen, dass die Erfahrung der Entwickler beider Projekte konzentriert und diese ablösen wird. *Pyramid*[30], der offizielle Nachfolger von Repoze.bfg, wird vom Pylons Projekt weiterentwickelt und wird folgendermaßen beworben.

> Pyramid is a small, fast, down-to-earth Python web application development framework. It is developed as part of the Pylons Project. It is licensed under a BSD-like license.

Die Entscheidung zwischen den Webframeworks ist nicht leicht. Sollen Applikationen geschrieben werden, die viele Inhalte verwalten sollen, ist Django mit sehr guter Dokumentation und weiteren zuschaltbaren Features die richtige Wahl.

Liegen kleine Anwendungen mit oder ohne Datenbankanbindung im Fokus, oder ist vorauszusehen, dass die Anwendung hohen Ansprüchen bei der Verwaltung von Klassen und Datenbanktabellen gerecht werden soll, ist Pylons und darauf aufbauend TurboGears 2.x in Verbindung mit SQLAlchemy die richtige Wahl. TurboGears bietet hier eine Vorauswahl an gängigen Tools out of the box an, wohingegen Pylons dem Entwickler bei der Auswahl mehr Freiheiten gibt. Dies erfordert aber eine größere Erfahrung mit Python und der Anzahl von Paketen, die diese Sprache für die Webentwicklung anbietet.

Sollen die Möglichkeiten aus der Welt von Django und Pylons in einem Framework vereint sein und auch noch ZODB genutzt werden können, sind Repoze.bfg und zukunftsträchtiger noch Pyramid die geeignete Wahl bei der Erstellung eines neuen Projekts.

[30] Pyramid Webframework: `http://docs.pylonshq.com/pyramid/dev/`, 10.01.2011

Literaturverzeichnis

[1] Farid Hajji:, *Das Python Praxisbuch - Der große Profi-Leitfaden fürProgrammierer*, Addison-Wesley, Deutsch, ISBN-13: 9783827325433, 2008

[2] Johannes Ernesti, *Python 3 Das umfassende Handbuch*, Galileo Press, Deutsch, ISBN-10: 3836214121, 2008

[3] Adrian Holovaty and Jacob Kaplan-Moss, *The Definitive Guide to Django: Web Development Done Right*, Apress, English, ISBN-10: 1590597257, 2007

[4] Ayman Hourieh, *Django 1.0 Website Development*, Packt Publishing, English, ISBN-10: 1847196780, 2009

[5] Jeff Forcier, Paul Bissex and Wesley Chun, *Python Web Development with Django*, Addison-Wesley Professional, Englisch, ISBN-10: 0132356139, 2008

[6] Oliver Bartosch, *Einstieg in SQL*, Galileo Press, Deutsch, ISBN-10: 3898424979, 2004

[7] Dave Thomas and David Heinemeier Hansson, *Agile Web Development with Rails*, The Pragmatic Programmers LLC, Englisch, ISBN-10: 1934356166, 2008

[8] Christian Wenz, *JavaScript and AJAX*, `http://openbook.galileocomputing.de/javascript_ajax/`, Galileo Press, Deutsch, ISBN-13: 9783898428590, 2007

[9] Nick Heinle and Bill Pena, *Webdesign mit JavaScript*, `http://www.oreilly.de/catalog/designjs2ger/chapter/f_Kapitel01IVZ.html`, O'Reilly, Deutsch, ISBN-10: 3897212757, 2002

[10] James Gardner, *The Definitive Guide to Pylons*, `http://pylonsbook.com/en/1.1/index.html`, 28.07.2010 Apress, Englisch, ISBN-10: 1590599349, 2009

[11] Ramm, Dangoor and Sayfan, *Rapid Web Applications with TurboGears*, `http://pylonsbook.com/en/1.1/index.html`, 28.07.2010 Prentice Hall, Englisch, ISBN-10: 0132433885, 2006

[12] Python Beschreibung, `http://www.python.org/about/`, 28.07.2010

[13] Django, `http://www.djangoproject.com/`, 29.07.2010

[14] phpwact.org, `http://www.phpwact.org`, 28.07.2010

[15] Python Webframeworks Übersicht , `http://wiki.python.org/moin/WebFrameworks`, 28.07.2010

[16] Pylons, `http://pylonshq.com/`, 29.07.2010

[17] zope, `http://www.zope.de/uber-zope`, 29.07.2010

[18] web2py, `http://www.web2py.com/examples/default/what`, 29.07.2010

[19] zend, `http://framework.zend.com/about/overview`, 30.07.2010

[20] Ruby on Rails, `http://rubyonrails.org/`, 30.07.2010

[21] perl.net.au, `http://perl.net.au/wiki/Catalyst`, 30.07.2010

[22] Catalystframework, `http://catalystframework.org/`, 30.07.2010

[23] Wicket Webframework, `http://wicket.apache.org/`, 30.07.2010

[24] Dojo, `http://dojotoolkit.org/`, 16.11.2010

[25] jQuery, `http://jquery.com/`, 16.11.2010

[26] MochiKit, `http://mochi.github.com/mochikit/`, 16.11.2010

[27] MooTools, `http://mootools.net/`, 16.11.2010

[28] Prototype, `http://www.prototypejs.org/`, 16.11.2010

[29] Yahoo! User Interface, `http://developer.yahoo.com/yui/`, 16.11.2010

[30] TurboGears, `http://turbogears.org/`, 7.01.2011

[31] ToscaWidgets, `http://toscawidgets.org/`, 7.01.2011

Index

T

U

V

W

X

Z

mikroadmin

```
#!/usr/bin/env python
from django.core.management import execute_manager
try:
    import settings # Assumed to be in the same directory.
except ImportError:
    import sys
    sys.stderr.write("Error: Can't find the file 'settings.py' in the directory
        containing %r. It appears you've customized things.\nYou'll have to run django-
        admin.py, passing it your settings module.\n(If the file settings.py does
        indeed exist, it's causing an ImportError somehow.)\n" % __file__)
    sys.exit(1)

if __name__ == "__main__":
    execute_manager(settings)
```

Listing A.1: manage.py

```
# Django settings for mikroadmin project.

DEBUG = True
TEMPLATE_DEBUG = DEBUG

ADMINS = (
    # ('Your Name', 'your_email@domain.com'),
)

MANAGERS = ADMINS

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3', # Add 'postgresql_psycopg2', '
            postgresql', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': '/home/jac/webframeworks/database/sqlite-mikroadmin/mikroadmin.db',   #
             Or path to database file if using sqlite3.
        'USER': '',                      # Not used with sqlite3.
        'PASSWORD': '',                  # Not used with sqlite3.
        'HOST': '',                      # Set to empty string for localhost. Not used
            with sqlite3.
        'PORT': '',                      # Set to empty string for default. Not used
            with sqlite3.
    }
}

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# On Unix systems, a value of None will cause Django to use the same
```

```
27 # timezone as the operating system.
28 # If running in a Windows environment this must be set to the same as your
29 # system time zone.
30 TIME_ZONE = 'Europe/Berlin'
31
32 # Language code for this installation. All choices can be found here:
33 # http://www.i18nguy.com/unicode/language-identifiers.html
34 LANGUAGE_CODE = 'de-de'
35
36 SITE_ID = 1
37
38 # If you set this to False, Django will make some optimizations so as not
39 # to load the internationalization machinery.
40 USE_I18N = True
41
42 # If you set this to False, Django will not format dates, numbers and
43 # calendars according to the current locale
44 USE_L10N = True
45
46 # Absolute path to the directory that holds media.
47 # Example: "/home/media/media.lawrence.com/"
48 MEDIA_ROOT = '/home/jac/webframeworks/django/mikroadmin/templates/'
49 #MEDIA_ROOT = ''
50
51 # URL that handles the media served from MEDIA_ROOT. Make sure to use a
52 # trailing slash if there is a path component (optional in other cases).
53 # Examples: "http://media.lawrence.com", "http://example.com/media/"
54 MEDIA_URL = ''
55
56 # URL prefix for admin media -- CSS, JavaScript and images. Make sure to use a
57 # trailing slash.
58 # Examples: "http://foo.com/media/", "/media/".
59 ADMIN_MEDIA_PREFIX = '/media/'
60
61 # Make this unique, and don't share it with anybody.
62 SECRET_KEY = '%m&(@%0dlif62u9u%mg74ss3%gn0#^r85m4m7!do1+ptf=gh64'
63
64 # List of callables that know how to import templates from various sources.
65 TEMPLATE_LOADERS = (
66     'django.template.loaders.filesystem.Loader',
67     'django.template.loaders.app_directories.Loader',
68 #     'django.template.loaders.eggs.Loader',
69 )
70
71 MIDDLEWARE_CLASSES = (
72     'django.middleware.common.CommonMiddleware',
73     'django.contrib.sessions.middleware.SessionMiddleware',
74     'django.middleware.csrf.CsrfViewMiddleware',
75     'django.contrib.auth.middleware.AuthenticationMiddleware',
76     'django.contrib.messages.middleware.MessageMiddleware',
77 )
```

```
78
79 ROOT_URLCONF = 'mikroadmin.urls'
80
81 TEMPLATE_DIRS = (
82     # Put strings here, like "/home/html/django_templates" or "C:/www/django/templates
       ".
83     # Always use forward slashes, even on Windows.
84     # Don't forget to use absolute paths, not relative paths.
85     "/home/jac/webframeworks/django/mikroadmin/templates",
86 )
87
88 INSTALLED_APPS = (
89     'django.contrib.auth',
90     'django.contrib.contenttypes',
91     'django.contrib.sessions',
92     'django.contrib.sites',
93     'django.contrib.messages',
94     'django.contrib.admin',
95     'mikroadmin.appMikroadmin',
96
97     # Uncomment the next line to enable the admin:
98     # 'django.contrib.admin',
99 )
```

Listing A.2: settings.py

```
1  from django.conf.urls.defaults import *
2
3  # Uncomment the next two lines to enable the admin:
4  from django.contrib import admin
5  admin.autodiscover()
6
7
8  urlpatterns = patterns('',
9      # Example:
10     # (r'^mikroadmin/', include('mikroadmin.foo.urls')),
11
12     # Uncomment the admin/doc line below and add 'django.contrib.admindocs'
13     # to INSTALLED_APPS to enable admin documentation:
14     # (r'^admin/doc/', include('django.contrib.admindocs.urls')),
15
16     # Uncomment the next line to enable the admin:
17     (r'^admin/', include(admin.site.urls)),
18 #     (r'^$', 'django.contrib.auth.views.login', {'template_name': '
       mikrocontrollertemplate/index.html'}),
19     (r'^$', 'mikroadmin.appMikroadmin.views.index'),
20     (r'^login/(?P<loginerror>\w*)$', 'mikroadmin.appMikroadmin.views.index'),
21     (r'^start/$', 'mikroadmin.appMikroadmin.views.start'),
22
23     (r'^board/(?P<board>\w+)$', 'mikroadmin.appMikroadmin.views.board'),
24     (r'^accounts/login/$', 'mikroadmin.appMikroadmin.views.mylogin'),
```

```
25      (r'^accounts/logout/$', 'mikroadmin.appMikroadmin.views.mylogout'),
26      (r'^board/(?P<board>\w+)/statusinformation/$', 'mikroadmin.appMikroadmin.views.
            boardstatusinformation'),
27      (r'^board/(?P<board>\w+)/turnon/$', 'mikroadmin.appMikroadmin.views.boardturnon'),
28      (r'^board/(?P<board>\w+)/turnoff/$', 'mikroadmin.appMikroadmin.views.boardturnoff')
            ,
29      (r'^board/(?P<board>\w+)/reset/$', 'mikroadmin.appMikroadmin.views.boardreset'),
30      (r'^board/(?P<board>\w+)/reservate/$', 'mikroadmin.appMikroadmin.views.
            boardReservation'),
31      (r'^board/(?P<board>\w+)/release/$', 'mikroadmin.appMikroadmin.views.boardrelease')
            ,
32      (r'^myadmin/$', 'mikroadmin.appMikroadmin.views.myadmin'),
33      (r'^activate/(?P<user>\w+)/$', 'mikroadmin.appMikroadmin.views.activateUser'),
34      (r'^newpassword/$', 'mikroadmin.appMikroadmin.views.newPassword'),
35
36      (r'^site_media/(?P<path>.*)$', 'django.views.static.serve',
37              {'document_root': '/home/jac/webframeworks/django/mikroadmin/templates'}),
38
39
40  )
```

Listing A.3: urls.py

Kapitel B.0

apache

```
1  import os
2  import sys
3
4  sys.path.append('/usr/lib/python2.5/site-packages/django')
5  sys.path.append('/home/jac/webframeworks/django')
6  sys.path.append('/home/jac/webframeworks/django/mikroadmin')
7  os.environ['DJANGO_SETTINGS_MODULE'] = 'mikroadmin.settings'
8
9  import django.core.handlers.wsgi
10 application = django.core.handlers.wsgi.WSGIHandler()
```

Listing B.1: django-example.wsgi

appMikroadmin

```
"""
This dummyBoard simulates the options and reaction of a microcontroller and shows
only the functions and return values which are expected.
"""



def statusinformation(board):
    """ Returns a dictionary of status information about the board"""
    dictionary = {'led':True, 'display':True, 'RJ-45':True }

    return dictionary

def power_on(board):
    """ Returns True if the turnon-script was successful """
    return True

def power_off(board):
    """ Returns True if the turnoff-script was successful """
    return True

def reset_active(board):
    """ Returns True if the reset-script was successful """
    return True

def reset_inactive(board):
    """ Returns True if the reset-script was successful """
    return True

def light_on(board):
    """ Returns True if the light was turned on """
    return True

def light_off(board):
    """ Returns True if the light was turned off """
    return True
```

Listing C.1: dummyBoard.py

```
from django import forms
from django.contrib.auth.models import User
from django.core.exceptions import ObjectDoesNotExist
from django.core.mail import send_mail



```

```
9  class RegisterForm(forms.Form):
10     username = forms.CharField(max_length=30, label='Username:')
11     email = forms.EmailField(label='Email:')
12     password1 = forms.CharField(widget=forms.PasswordInput, label='Passwort:')
13     password2 = forms.CharField(widget=forms.PasswordInput, label='Passwort wiederholen
           :')
14
15
16     def samePassword(self, cleanedPassword1, cleanedPassword2):
17         same = False
18         #print "Passwoerter:"
19         #print cleanedPassword1
20         #print cleanedPassword2
21
22         if cleanedPassword1 == cleanedPassword2:
23             same = True
24         else:
25             same = False
26
27         return same
28
29
30     def saveNewUser(self, cleanedUsername, cleanedEmail, cleanedPassword):
31         u = User.objects.create_user(cleanedUsername, cleanedEmail, cleanedPassword)
32         u.save()
33
34         return u
35
36
37     def isRegisterd(self, cleanedUsername):
38         try:
39             u = User.objects.get(username=cleanedUsername)
40         except ObjectDoesNotExist:
41             u = False
42
43         return u
44
45     def sendEmail(self, cleanedUsername, cleanedEmail, cleanedPassword):
46         send_mail('Mikroadmin Registrierung',
47                   'Willkommen bei Mikroadmin. Sie koennen sich folgendermassen
                       einloggen. Username: %(username)s  Passwort %(password)s' % {'
                       username': cleanedUsername, 'password':cleanedPassword},
48                   'mirkoadmin@fh-augsburg.de',
49                   [cleanedEmail], fail_silently=False)
50
51         return send_mail
```

Listing C.2: forms.py

```
1  from django.db import models
2  from django.contrib.auth.models import User
```

```
3  import unittest
4
5  # Create your models here.
6  class Board(models.Model):
7      """
8      Class description for microcontroller which can be managed
9
10     # Testing for creation of a board instance
11     >>> myBoard = Board.objects.create(name='Atmel AT91SAM7',
12     ... description='http://www.atmel.com/', linuxUsername='',
13     ... image='standardimage.jpg', currentPasswort='', isReservedBy=None,
14     ... reservedDate=None)
15
16     >>> myBoard.save()
17
18     >>> myBoard.name
19     'Atmel AT91SAM7'
20     """
21     name = models.CharField(max_length=30)
22     description = models.CharField(max_length=30, blank=True, null=True)
23     linuxUsername = models.CharField(max_length=30, blank=True, null=True)
24     image = models.CharField(max_length=30, blank=True, null=True)
25     currentPasswort = models.CharField(max_length=30, blank=True, null=True)
26     isReservedBy = models.ForeignKey(User, blank=True, null=True)
27     reservedDate = models.DateTimeField(blank=True, null=True)
28
29     def __unicode__(self):
30         return self.name
31
32 # Unittest
33 class MyFuncTestCase(unittest.TestCase):
34     def setUp(self):
35         self.myBoard = Board.objects.create(name='Atmel AT91SAM7',
36                     description='http://www.atmel.com/',
37                     linuxUsername='',
38                     image='standardimage.jpg',
39                     currentPasswort='', isReservedBy=None,
40                     reservedDate=None)
41
42     def testmeineFuktion(self):
43         self.assertEqual(self.myBoard.name, 'Atmel AT91SAM7')
```

Listing C.3: models.py

```
1  """
2  This file demonstrates two different styles of tests (one doctest and one
3  unittest). These will both pass when you run "manage.py test".
4
5  Replace these with more appropriate tests for your application.
6  """
7
```

```
8  from views import meineFunktion
9
10 __test__ = {
11     'meineFunktion': meineFunktion
12 }
13
14
15 import unittest
16
17 class MyFuncTestCase(unittest.TestCase):
18     def setUp(self):
19         self.liste = ['django', 'web', 'framework']
20
21
22     def testmeineFuktion(self):
23         self.assertEqual(meineFunktion(self.liste, 0), 'django')
24         self.assertEqual(meineFunktion(self.liste, 1), 'web')
25
26
27
28 #from django.test import TestCase
29
30 #class SimpleTest(TestCase):
31
32 #     def test_basic_addition(self):
33
34 #         """
35
36 #         Tests that 1 + 1 always equals 2.
37
38 #         """
39
40 #         self.failUnlessEqual(1 + 1, 2)
41
42 #
43
44 #__test__ = {"doctest": """
45
46 #Another way to test that 1 + 1 is equal to 2.
47
48 #
49
50 #>>> 1 + 1 == 2
51
52 #True
53
54 #"""}
```

Listing C.4: tests.py

```
1  # Create your views here.
```

```
2  from django.http import HttpResponse
3  #from django.template import Context, loader
4  from django.shortcuts import render_to_response, redirect
5  from mikroadmin.appMikroadmin.models import Board, User
6  from django.contrib.auth.decorators import login_required, permission_required
7  from mikroadmin.appMikroadmin.forms import RegisterForm
8  from django.template import RequestContext
9  from django.contrib.auth import authenticate, login, logout
10 #from microcontrollertools.microtools import changePassword, generatePassword
11 from random import choice
12 import string
13 import os
14 from subprocess import *
15 import threading
16 from datetime import datetime
17 import smtplib
18 from django.core.exceptions import ObjectDoesNotExist
19 from subprocess import call
20 from random import choice
21 import string
22 import os
23
24
25
26 from dummyBoard import statusinformation, power_on, power_off, reset_active,
       reset_inactive, light_on, light_off
27
28
29 def boardReservationStatus(request, board):
30     '''
31     Checkin if User reserved a board or is free to show a specific page to
32     the User if he wants to reserve a board
33     '''
34     board = Board.objects.get(name=board)
35
36     if  board.isReservedBy == request.user or (board.isReservedBy == None):
37         return True
38     else:
39         return False
40
41
42 def index(request, loginerror=None):
43     registinfo = None
44
45     if request.method == 'POST':
46         regform = RegisterForm(request.POST)
47
48         if regform.is_valid():
49             cleanedUsername = regform.cleaned_data['username']
50             cleanedEmail = regform.cleaned_data['email']
51             cleanedPassword1 = regform.cleaned_data['password1']
```

```
52              cleanedPassword2 = regform.cleaned_data['password2']
53
54              if not regform.samePassword(cleanedPassword1, cleanedPassword2):
55                  registinfo = 'Passwort1 und Passwort2 sind nicht gleich'
56
57              elif regform.isRegisterd(cleanedUsername):
58                  registinfo= 'User existiert bereits'
59
60              elif registinfo == None:
61                  regform.saveNewUser(cleanedUsername, cleanedEmail, cleanedPassword1)
62
63                  thisuser = User.objects.get(username=cleanedUsername)
64                  thisuser.is_active = False
65                  thisuser.save()
66
67                  mikroadminMailRegistration('mikroadmin@hs-augsburg.de', cleanedEmail, '
                        Mikroadmin Registrierung', cleanedUsername, cleanedPassword1)
68                  #regform.sendEmail(cleanedUsername, cleanedEmail, cleanedPassword1)
69                  registinfo = 'ok'
70
71      else:
72          regform = RegisterForm()
73
74
75
76      return render_to_response('mikrocontrollertemplate/index.html',
77                                {'regform': regform, 'loginerror': loginerror, '
                                      registinfo': registinfo},
78                                context_instance=RequestContext(request))
79
80
81  def mylogin(request):
82      if request.method == 'POST':
83          username = request.POST['username']
84          password = request.POST['password']
85
86          user = authenticate(username=username, password=password)
87
88          if user is not None:
89              if user.is_active:
90                  login(request, user)
91                  return redirect('/mikroadmin/start/')
92              else:
93                  return redirect('/mikroadmin/')
94          else:
95              return redirect('/mikroadmin/login/error')
96      else:
97          return redirect('/mikroadmin/')
98
99
100 def mylogout(request):
```

```
101     boards = Board.objects.filter(isReservedBy=request.user)
102 
103     logoutInSecs= 10
104     size = 8
105     for board in boards:
106         password = generatePassword(size)
107         #password =  ''.join([choice(string.letters + string.digits) for i in range(
                size)])
108         returncode = changePassword(board.name, password)
109 
110         board.isReservedBy = None
111         board.save()
112 
113         os.system('echo "Sie werden in ' + str(logoutInSecs) + ' Sekunden ausgeloggt" |
                write ' + board.name + ' &')
114         os.system('sleep ' + str(logoutInSecs) + ' && ' +'sudo skill -KILL -u ' + board
                .name + ' &')
115         #pid = os.system('sudo skill -KILL -u ' + board.name + ' &')
116 
117 
118     logout(request)
119     return redirect('/mikroadmin/')
120 
121 
122 @login_required
123 def start(request):
124     boards = Board.objects.all()
125 
126     return render_to_response('mikrocontrollertemplate/start.html', {'boards': boards,
            'request':request,  'isSuperuser':request.user.is_superuser})
127 
128 @login_required
129 def board(request, board='Board1'):
130     ok = boardReservationStatus(request, board)
131 
132     board = Board.objects.get(name=board)
133 
134     if ok:
135         if board.isReservedBy == request.user:
136             info = 'reserved'
137         else:
138             info = 'free'
139 
140         return render_to_response('mikrocontrollertemplate/board.html', {'board':board,
                'request':request, 'info':info})
141     else:
142         # This try-except block is important for distinction. Otherwise an error will
                be sent to the User
143         try:
144             userWhoReservedBoard = User.objects.get(username=board.isReservedBy)
145 
```

```
146             # This statement is only necessary for empty email fields to show on
147             # Template that this field ist 'Leer'. Each user has to set an email
                    address
148             if userWhoReservedBoard.email == '':
149                 userWhoReservedBoard.email = 'Leer'
150                 userWhoReservedBoard.save()
151 
152         except:
153             # controlled Exception to do nothing
154             pass
155 
156         info = 'reserved'
157         return render_to_response('mikrocontrollertemplate/board.html', {'board': board
                , 'request':request, 'info':info})
158 
159 @login_required
160 def boardReservation(request, board):
161     ok = boardReservationStatus(request, board)
162 
163     board = Board.objects.get(name=board)
164 
165     if ok:
166         if board.isReservedBy is None:
167             size = 8
168             password = generatePassword(size)
169             #password =  ''.join([choice(string.letters + string.digits) for i in range
                    (size)])
170 
171             returncode = changePassword(board.name, password)
172 
173             board = Board.objects.get(name=board)
174             board.currentPasswort = password
175             board.isReservedBy = request.user
176             board.reservedDate = datetime.now()
177             board.save()
178 
179             info = 'reserved'
180         elif board.isReservedBy == request.user :
181             info = 'reserved'
182         else:
183             info = 'free'
184 
185         return render_to_response('mikrocontrollertemplate/reservation.html', {'board':
                board, 'request':request, 'info':info})
186     else:
187         info = 'reserved'
188         return render_to_response('mikrocontrollertemplate/reservation.html', {'board':
                board, 'request':request, 'info':info})
189 
190 
191 @login_required
```

```
192 def boardstatusinformation(request, board):
193     ok = boardReservationStatus(request, board)
194 
195     board = Board.objects.get(name=board)
196 
197     statusdict = False
198 
199     if ok:
200         if board.isReservedBy == request.user:
201             statusdict = statusinformation(board)
202             info = 'reserved'
203         else:
204             info = 'free'
205 
206         return render_to_response('mikrocontrollertemplate/statusinformation.html', {'
                request':request, 'statusdict':statusdict, 'board':board, 'info':info})
207     else:
208         info = 'reserved'
209         return render_to_response('mikrocontrollertemplate/statusinformation.html', {'
                request':request, 'statusdict':statusdict, 'board':board, 'info':info})
210 
211 @login_required
212 def boardturnon(request, board):
213     ok = boardReservationStatus(request, board)
214 
215     board = Board.objects.get(name=board)
216 
217     status = False
218 
219     if ok:
220         if board.isReservedBy == request.user:
221             status = power_on(board)
222             info = 'reserved'
223         else:
224             info = 'free'
225 
226         return render_to_response('mikrocontrollertemplate/turnon.html', {'request':
                request, 'status':status, 'board':board, 'info':info})
227     else:
228         info = 'reserved'
229         return render_to_response('mikrocontrollertemplate/turnon.html', {'request':
                request, 'status':status, 'board':board, 'info':info})
230 
231 @login_required
232 def boardturnoff(request, board):
233     ok = boardReservationStatus(request, board)
234 
235     board = Board.objects.get(name=board)
236 
237     status = False
238 
```

```
239     if ok:
240         if board.isReservedBy == request.user:
241             status = power_off(board)
242             info = 'reserved'
243         else:
244             info = 'free'
245 
246         return render_to_response('mikrocontrollertemplate/turnoff.html', {'request':
                request, 'status':status, 'board':board, 'info':info})
247     else:
248         info = 'reserved'
249         return render_to_response('mikrocontrollertemplate/turnoff.html', {'request':
                request, 'status':status, 'board':board, 'info':info})
250 
251 
252 @login_required
253 def boardreset(request, board):
254     ok = boardReservationStatus(request, board)
255 
256     board = Board.objects.get(name=board)
257 
258     status = False
259 
260     if ok:
261         if board.isReservedBy == request.user:
262             status = reset_active(board)
263             info = 'reserved'
264         else:
265             info = 'free'
266 
267         return render_to_response('mikrocontrollertemplate/reset.html', {'request':
                request, 'status':status, 'board':board, 'info':info})
268     else:
269         info = 'reserved'
270         return render_to_response('mikrocontrollertemplate/reset.html', {'request':
                request, 'status':status, 'board':board, 'info':info})
271 
272 
273 def killBoard(board):
274     myreturn = Popen('sudo skill -KILL -u ' + board, shell=True)
275 
276     return myreturn
277 
278 
279 def boardrelease(request, board):
280     board = Board.objects.get(name=board)
281 
282     if board.isReservedBy == request.user:
283         size = 8
284         logoutInSecs = 1
285         password = generatePassword(size)
```

```
286         #password =  ''.join([choice(string.letters + string.digits) for i in range(
                size)])
287         returncode = changePassword(board.name, password)
288 
289         board.isReservedBy = None
290         board.save()
291 
292         os.system('echo "Sie werden in ' + str(logoutInSecs) + ' Sekunden ausgeloggt" |
                 write ' + board.name + ' &')
293         t = threading.Timer(logoutInSecs, killBoard, [board.name])
294         t.start()
295 
296         #Popen('ls -lhisa', shell=True)
297         #Popen('sleep ' + str(logoutInSecs) + ' && ' +'sudo skill -KILL -u ' + board.
                name, shell=True ).pid
298         #os.system('sudo skill -KILL -u ' + board.name + ' &')
299 
300         return render_to_response('mikrocontrollertemplate/release.html', {'board':
                board})
301 
302     else:
303         return render_to_response('mikrocontrollertemplate/releasefailure.html', {'
                board':board})
304 
305 
306 @permission_required('request.user.is_superuser')
307 def myadmin(request):
308     myboards = Board.objects.all()
309     logoutInSecs = 5
310 
311     if request.method == 'POST':
312 
313         if request.POST['board'] == 'All':
314             boards = Board.objects.filter(isReservedBy__isnull=False)
315 
316             for board in boards:
317                 board.isReservedBy = None
318                 board.save()
319 
320                 os.system('echo "Sie werden in ' + str(logoutInSecs) + ' Sekunden
                        ausgeloggt" | write ' + board.name + ' &')
321                 t = threading.Timer(logoutInSecs, killBoard, [board.name])
322                 t.start()
323 
324         else:
325             boards = Board.objects.filter(name=request.POST['board'])
326 
327             for board in boards:
328                 board.isReservedBy = None
329                 board.save()
330 
```

```
331                 os.system('echo "Sie werden in ' + str(logoutInSecs) + ' Sekunden
                        ausgeloggt" | write ' + board.name + ' &')
332                 t = threading.Timer(logoutInSecs, killBoard, [board.name])
333                 t.start()
334
335     else:
336         myboard = "Kein Board ausgewaehlt!"
337
338     reservedBoards = Board.objects.filter(isReservedBy__isnull=False)
339
340     return render_to_response('mikrocontrollertemplate/myadmin.html', {'request':
            request, 'myboards':myboards, 'reservedBoards':reservedBoards},
            context_instance=RequestContext(request))
341
342
343
344 def mikroadminMailRegistration(sender, to, subject, user, password):
345
346     thissender = 'From:' + sender+'\r\n'
347     thisto ='To:' + to +'\r\n'
348     thissubject ='Subject:' + subject +'\r\n'
349
350     header = thissender+thisto+thissubject
351     #header = {'Subject:':thissubject, 'From:':thissender, 'To:':thisto}
352     message = 'Willkommen bei Mikroadmin\n\nSie koennen sich nach Aktivierung http://
            rabbit.informatik.fh-augsburg.de/mikroadmin/activate/' + user + ' mit Username:
             ' + user + ' und Passwort: ' + password + ' anmelden'
353
354     sendMail(sender, to, header, message, 'smtp.hs-augsburg.de')
355
356     return 0
357
358
359 def activateUser(request, user):
360     registinfo = False
361     try:
362         thisuser = User.objects.get(username=user)
363     except:
364         return redirect('/mikroadmin/')
365
366     if not thisuser.is_active:
367         thisuser.is_active = True
368         thisuser.save()
369         registinfo = 'User wurde aktiviert'
370     else:
371         registinfo = 'User ist bereits aktiv'
372
373
374     regform = RegisterForm()
375
376     return render_to_response('mikrocontrollertemplate/index.html',
```

```
377                             {'registinfo': registinfo, 'regform':regform},
                                    context_instance=RequestContext(request))
378
379
380 def newPassword(request):
381
382     info = False
383
384     if request.method == 'POST':
385         password = generatePassword(8)
386
387         useremail = request.POST['useremail']
388
389         try:
390             thisuser = User.objects.get(email=useremail)
391         except ObjectDoesNotExist:
392             info = 'ObjectDoesNotExist'
393             return render_to_response('mikrocontrollertemplate/newpassword.html',
394                                 {'info':info}, context_instance=RequestContext(
                                        request))
395
396         thisuser.set_password(password)
397         thisuser.save()
398
399         sender = 'mikroadmin@hs-augsburg.de'
400         to = useremail
401         subject = 'Mikroadmin Passwort'
402
403         thissender = 'From:' + sender+'\r\n'
404         thisto ='To:' + to +'\r\n'
405         thissubject ='Subject:' + subject +'\r\n'
406
407         header = thissender+thisto+thissubject
408         #header = {'Subject:':thissubject, 'From:':thissender, 'To:':thisto}
409         message = 'Sie koennen sich jetzt mittels Username: ' + thisuser.username + '
                und dem neuen Passwort: ' + password + ' anmelden.'
410
411         sendMail(sender, to, header, message, 'smtp.hs-augsburg.de')
412
413         info = 'send'
414
415
416     return render_to_response('mikrocontrollertemplate/newpassword.html',
417                                 {'info':info}, context_instance=RequestContext(
                                        request))
418
419
420 def sendMail(sender, to, header, message, smtpserver):
421     server = smtplib.SMTP(smtpserver)
422     server.sendmail(sender, to, header+message)
423     server.set_debuglevel(1)
```

```
424     server.quit()
425
426     return 0
427
428
429 def changePassword(board, password):
430     commandlist = ['echo "' + board + ':' + password + ':' +  ':' +  ':' + '
            Microcontroller' +  ':' + '/home/' + board + ':' + '/bin/bash' + '"', ' | ', '
            sudo newusers']
431     returncode = os.system(''.join(commandlist))
432
433     return returncode
434
435
436 def generatePassword(length):
437     newpassword =  ''.join([choice(string.letters + string.digits) for i in range(
            length)])
438
439     return newpassword
440
441
442 def meineFunktion(liste, stelle):
443     """
444     >>> a = ['django', 'web', 'framework']
445     >>> meineFunktion(a, 0)
446     'django'
447     >>> meineFunktion(a, 1)
448     'web'
449     """
450     return liste[stelle]
```

Listing C.5: views.py

templates

D.1 admin

```
{% extends "admin/base.html" %}
{% load i18n %}

{% block title %}{{ title }} | {% trans 'MikroAdmin Administration' %}{% endblock %}

{% block branding %}
<h1 id="site-name">{% trans 'Mikroadmin Administration' %}</h1>
{% endblock %}

{% block nav-global %}{% endblock %}
```

Listing D.1: `base_site.html`

D.2 mikrocontrollertemplate

```
{% extends "mikrocontrollertemplate/start.html" %}


{% block content %}
<div id="boardcontent">

<h1 id="boardhead">{{ board.name }}</h1>

<div id="oneboardcontent">
<ul id="boardnavigation">
    <li id="imagereservate" ><a class="boardlink" href="#" alt="/mikroadmin/board/{{
        board }}/reservate/">Reservieren</a></li>
    <li id="imagerelease" ><a class="boardlink" href="#" alt="/mikroadmin/board/{{
        board }}/release/">Freigeben</a></li>
    <li id="imagestatusinformation" ><a class="boardlink" href="#" alt="/mikroadmin/
        board/{{ board }}/statusinformation/">Statusinformationen</a></li>
    <li id="imageturnon" ><a class="boardlink" href="#" alt="/mikroadmin/board/{{ board
         }}/turnon/">Einschalten</a></li>
    <li id="imageturnoff" ><a class="boardlink" href="#" alt="/mikroadmin/board/{{
        board }}/turnoff/">Ausschalten</a></li>
    <li id="imagereset"><a class="boardlink" href="#" alt="/mikroadmin/board/{{ board
        }}/reset/" href="#">Reset</a> </li>
    <li id="imagehome"><a href="/mikroadmin/start/">Board Wechseln</a></li>
</ul>
<div id="navigationfeedback">
<div id="information">
```

```
{% if info == 'free' %}
Mittels "Reservieren" öknnen sie ein Board reservieren.
{% else%}
    {% if board.isReservedBy == request.user %}
    {{ board.isReservedBy }} hat {{ board.name }} am {{ board.reservedDate }}
        reserviert.
    Mit folgender Kombination Username: {{ board.name }} und Passwort: {{ board.
        currentPasswort }}ö
    knnen sie sich am Server anmelden.
    {% else %}
    <em>{{ board }}</em> ist von <em>{{ board.isReservedBy }}</em> bereits am
    <em>{{ board.reservedDate }}</em> reserviert worden!
    Wenn Sie das Board dringend brauchen öknnen sie  <em>{{ board.isReservedBy }}</em>
    unter <em>{{ request.user.email }}</em> erreichen.
    <br/><br/>
    {% endif %}
{% endif%}
</div>
</div>
</div>

<div class="clear"></div>
</div>
{% endblock %}
```

Listing D.2: board.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head>
 <title>MikroAdmin</title>
 <meta http-equiv="content-type" content="text/html; charset=utf-8">
 <link rel="stylesheet" type="text/css" href="/site_media/mikrocontrollertemplate/css/
     style.css">
 <script type="text/javascript" src="http://code.jquery.com/jquery-latest.min.js"></
     script>
 <script type="text/javascript" src="/site_media/mikrocontrollertemplate/js/script.js">
     </script>


</head>
<body>
    <div id="head">
    {% block head %}
        <div id="headcontent">
        <a href="/mikroadmin/">
        <div id="logo">
            <!-- included in css -->
        </div>
```

```
22          </a>
23          <div id="login">
24              <form id="loginformular" method="post" action="/mikroadmin/accounts/login/"
                    >
25
26              {% csrf_token %}
27               <label for="username">Username:</label>
28               <input type="text" id="username" name="username" />
29               <label for="password">Passwort:</label>
30               <input type="password" id="password" name="password" />
31               <input type="submit" id="submitlogin" value="Login"/>
32               <input type="hidden" name="next" value="/start" />
33              </form>
34              {% if loginerror %}
35              <div id="loginerror">
36                      Benutzername oder Passwort falsch, bitte loggen sie sich neu ein
37              </div>
38              {% endif %}
39              <div><a id="newpasswordlink" href="/mikroadmin/newpassword/">Passwort
                    vergessen?</a></div>
40          </div>
41
42
43
44          <div class="clear"></div>
45          </div>
46      {% endblock %}
47      </div>
48  <div id="divbody">
49      {% block content %}
50      <div id="content">
51
52          <div id="beschreibung">
53              <div id="beschreibungtext">
54              <div id="beschreibungtextcontent">
55              <fieldset id="about">
56              <legend class="legend">Info:</legend>
57
58              <div>
59                  MikroAdmin öermglicht es dem registrierten Besucher aus einer
60                  Auswahl von Mikrocontrollern einen üfr sich zu reservieren, umö
61                  ungestrt per SSH auf diese zuzugreifen. Diese Applikationö
62                  ermglicht unter anderem folgende weitere Funktionen,
63                  <ul>
64                      <li>Reservierung</li>
65                      <li>Power Off/On</li>
66                      <li>Reset</li>
67                      <li>Abrufen von Statusinformationen</li>
68                  </ul>ü
69                  fr die jeweiligen Controller.
70              </div>
```

```
71              </fieldset>
72              </div>
73              </div>
74          </div>
75          <div id="registrierung">
76
77              <form id="registerformular" method="post" action="/mikroadmin/">
78              {% csrf_token %}
79              <fieldset id="registerformularfieldset">
80              <legend class="legend">Registrierung:</legend>
81              <!--
82               <div>
83               <label for="registerusername">Username:</label>
84               <input type="text" id="registerusername" name="registerusermail" />
85              </div>
86              <div>
87               <label for="registeremail">Email:</label>
88               <input type="text" id="registeremail" name="registeremail" />
89              </div>
90              <div>
91               <label for="registerpassword">Passwort:</label>
92               <input type="password" id="registerpassword" name="registerpassword" />
93              </div>
94              <div>
95               <label for="registerpassword2">Passwort wiederholen:</label>
96               <input type="password" id="registerpassword2" name="registerpassword2" />
97              </div>
98              -->
99              {{ regform.as_p }}
100
101
102             <input type="submit" id="submitregister" value="Registrieren"/>
103             </fieldset>
104             </form>
105             <div id="registrierunginfo">
106             Die Registrierung und die Nutzung dieses Services ist kostenlos.
107             </div>
108
109             {% if registinfo %}
110             <div id="registinfo">
111                 <div class="altf4"><div class="x">x</div></div>
112                 <div class="clear"></div>
113                 <div id="registmessage">{{ registinfo }}</div>
114             </div>
115             {% endif %}
116
117         </div>
118         <div class="clear"></div>
119         <div class="push"></div>
120
121     </div>
```

```
122     {% endblock %}
123     <div id="footer">
124         <div id="footercontent">
125             <span>Ein Service der Hochschule Augsburg</span>
126 
127         </div>
128     </div>
129 
130 
131 </div>
132 </body>
133 </html>
```

Listing D.3: index.html

```
1  {% extends "mikrocontrollertemplate/start.html" %}
2  
3  
4  {% block content %}
5  <div id="boardcontent">
6  
7  <div id="boardreserved">
8  <em>{{ board }}</em> ist von <em>{{ board.isReservedBy }}</em> bereits am
9  <em>{{ board.reservedDate }}</em> reserviert worden!
10 Wenn Sie das Board dringend brauchen öknnen sie  <em>{{ board.isReservedBy }}</em>
11 unter <em>{{ userWhoReservedBoard.email }}</em> erreichen.
12 <br/><br/>
13 <a href="/mikroadmin/start/">üZurck zur Boardauswahl</a>
14 </div>
15 
16 
17 
18 <div class="clear"></div>
19 </div>
20 {% endblock %}
```

Listing D.4: isreserved.html

```
1  {% extends "mikrocontrollertemplate/start.html" %}
2  
3  
4  {% block content %}
5  <div id="boardcontent">
6  
7  <h1>{{ board.name }}</h1>
8  
9  <div id="oneboardcontent">
10 <ul id="boardnavigation">
11 
12     <li><a href="/mikroadmin/start/">Home</a></li>
13 </ul>
```

```
<div id="navigationfeedback">
    <div id="myadmin">
    <h2>Reservierung und Status</h2>
    <ul>
    {% for myboard in myboards %}
        {% if myboard.isReservedBy %}
        <li class="myadminbelegt">
        <em>{{ myboard }}</em> is reserviert von <em>{{myboard.isReservedBy}}</em>
        </li>
        {% else %}
        <li class="myadminfrei">
        <em>{{ myboard }}</em> ist frei
        </li>
        {% endif %}
    {% endfor %}
    </ul>

    <form id="myadminreleaseform" action="/mikroadmin/myadmin/" method="POST">
    {% csrf_token %}
    <select name="board" size="1">
      <option>All</option>
      {% for reservedBoard in reservedBoards %}
          <option>{{ reservedBoard }}</option>
      {% endfor %}
    </select>
    <input type="submit" value="Freigeben">
    </form>
    </div>

</div>
</div>

<div class="clear"></div>
</div>
{% endblock %}
```

Listing D.5: myadmin.html

```
{% extends "mikrocontrollertemplate/index.html" %}


    {% block content %}
    <div id="forgotpassword">
        <div class="transparent" >For the margin</div>
    {% if info == 'send'%}
    <div id="newpasswordinfo">Ihr neues Passwort wurde an Ihre Emailadresse versandt</
        div>
    {% else %}
       {% if info == 'ObjectDoesNotExist'%}
       <div id="newpasswordinfo">User existiert nicht</div>
       {% else %}
```

```
13          <form id="formforgotpassword" method="POST" action"/mikroadmin/newpassword/">
14          {% csrf_token %}
15            <input type="text" id="usermail" name="useremail" value="ihre@email.de" />
16               <input type="submit" value="Password anfordern"/>
17          </form>
18          {% endif%}
19      {% endif %}
20      </div>
21
22      {% endblock %}
```

Listing D.6: newpassword.html

```
1  <div id="information">
2  Das Board: {{ board.name }} wurde freigegeben.
3  </div>
```

Listing D.7: release.html

```
1  <div id="information">
2  Das Board: {{ board.name }} wurde nicht von Ihnen reserviert!
3  </div>
```

Listing D.8: releasefailure.html

```
1  <div id="information">
2  {% if info == 'free' %}
3  Mittels "Reservieren" öknnen sie ein Board reservieren.
4  {% else%}
5      {% if board.isReservedBy == request.user %}
6      {{ board.isReservedBy }} hat {{ board.name }} am {{ board.reservedDate }}
           reserviert.
7      Mit folgender Kombination Username: {{ board.name }} und Passwort: {{ board.
           currentPasswort }}ö
8      knnen sie sich am Server anmelden.
9      {% else %}
10     <em>{{ board }}</em> ist von <em>{{ board.isReservedBy }}</em> bereits am
11     <em>{{ board.reservedDate }}</em> reserviert worden!
12     Wenn Sie das Board dringend brauchen öknnen sie  <em>{{ board.isReservedBy }}</em>
13     unter <em>{{ request.user.email }}</em> erreichen.
14     <br/><br/>
15     {% endif %}
16 {% endif%}
17
18 </div>
```

Listing D.9: reservation.html

```
1  <div id="information">
2  {% if info == 'free' %}
```

```
3  Sie ümssen das Board reservieren, um diese Aktion üauszufhren
4  {% else %}
5      {% if board.isReservedBy == request.user %}
6      {{ status }}: Reset wurde üausgefhrt
7      {% else %}
8      Board anderweitig reserviert
9      {% endif %}
10
11 {% endif %}
12
13 </div>
```

Listing D.10: reset.html

```
1  {% extends "mikrocontrollertemplate/index.html" %}
2
3  {% block head %}
4   <div id="headcontent">
5          <a href="/mikroadmin/start/">
6          <div id="logo">
7              <!-- included in css -->
8          </div>
9          </a>
10
11         <div id="startnavigation">
12         <span id="home">
13          <img src="/site_media/mikrocontrollertemplate/images/home.png" /> <a href="/
               mikroadmin/start/"> Home</a>
14         </span>
15
16         {% if request.user.is_superuser %}
17         <span id="adminlink">
18             <img src="/site_media/mikrocontrollertemplate/images/wrench.png" /><a href=
                   "/mikroadmin/myadmin/"> Adminbereich</a>
19         </span>
20         {% endif %}
21
22         <span id="logout">
23           <img src="/site_media/mikrocontrollertemplate/images/cross.png" />    <a href
                ="/mikroadmin/accounts/logout"> Logout</a>
24         </span>
25         <div id="usernamelogin">
26          <img src="/site_media/mikrocontrollertemplate/images/user.png" /> <span> {{
               request.user }}</span>
27         </div>
28         </div>
29
30         <div class="clear"></div>
31         </div>
32 {% endblock %}
33
```

```
{% block content %}
<div id="boardcontent">

<ul id="boardoverview">
{% for board in boards %}
<li class="boardunit">
    <a href="/mikroadmin/board/{{ board.name }}">
            {% if board.isReservedBy == request.user %}
            <img class="boardimage" src="/site_media/mikrocontrollertemplate/images/
                standardmikrocontroller-user.jpg" alt="{{ board.name }}" />
            {% else %}
             {% if board.isReservedBy %}
             <img class="boardimage" src="/site_media/mikrocontrollertemplate/images/
                 standardmikrocontroller-reserviert.jpg" alt="{{ board.name }}" />
             {% else %}
             <img class="boardimage" src="/site_media/mikrocontrollertemplate/images/{{
                  board.image }}" alt="{{ board.name }}" />
             {% endif %}
            {% endif %}
    </a>
    <div class="boardname">{{ board.name }}</div>
</li>
{% endfor %}
</ul>

<div class="clear"></div>
</div>
{% endblock %}
```

Listing D.11: start.html

```
<div id="information">
{% if info == 'free' %}
Sie ümssen das Board reservieren, um diese Aktion üauszufhren
{% else %}
    {% if board.isReservedBy == request.user %}
    {{ statusdict }}
    {% else %}
    Board anderweitig reserviert
    {% endif %}

{% endif %}

</div>
```

Listing D.12: statusinformation.html

```
<div id="information">
{% if info == 'free' %}
Sie ümssen das Board reservieren, um diese Aktion üauszufhren
{% else %}
```

```
5       {% if board.isReservedBy == request.user %}
6       {{ status }}: Das Board wurde ausgeschaltet
7       {% else %}
8       Board anderweitig reserviert
9       {% endif %}
10
11  {% endif %}
12
13  </div>
```

Listing D.13: turnoff.html

```
1   <div id="information">
2   {% if info == 'free' %}
3   Sie ümssen das Board reservieren, um diese Aktion üauszufhren
4   {% else %}
5       {% if board.isReservedBy == request.user %}
6       {{ status }}: Das Board wurde eingeschaltet
7       {% else %}
8       Board anderweitig reserviert
9       {% endif %}
10
11  {% endif %}
12
13  </div>
```

Listing D.14: turnon.html

D.2.1 css

```
1   *{
2    margin: 0;
3   }
4
5   html, body {
6   background-color:#666666;
7   margin: 0;
8   padding: 0;
9   height: 100%; /* WICHTIG!!! STRECKT ANZEIGENBEREICH AUF 100% */
10  font-family:Arial, Helvetica, sans-serif;
11
12  }
13
14  a:link{
15      color:white;
16      text-decoration:underline;
17  }
18
```

```
19 a:visited{
20     color:white;
21     text-decoration:underline;
22 }
23 
24 a:active{
25     color:white;
26     text-decoration:underline;
27 }
28 
29 a:hover{
30     color:#3B5998;
31     background-color:white;
32     text-decoration:none;
33 }
34 
35 em{
36     font-weight: bold;
37 }
38 
39 .transparent{
40     color:transparent;
41 }
42 
43 #divbody{
44     margin:auto;
45     /*width: 60em;*/
46     background-color:#666666;
47     /*margin: auto;*/
48     height:100%; /* öMindesthhe üfr moderne Browser */
49     overflow: hidden !important; /* FF Scroll-leiste */
50     width:60em;
51 
52 }
53 
54 #head{
55     background-color: #3B5998;
56     height:6em;
57     border-bottom-style:solid;
58     border-bottom-color:white;
59 }
60 
61 #headcontent{
62     position:relative;
63     top:1em;
64     width: 60em;
65     margin: auto;
66 
67 }
68 
69 #logo{
```

```
70      float:left;
71      /*background-color: blue;*/
72      height: 3.75em;
73      background-image:url(../images/logo.png);
74      width:14.625em;
75  }
76
77  #login{
78      /*width: 40em;*/
79      float: right;
80      /*margin-right: 10em;*/
81      /*background-color:green;*/
82      height: 3.75em;
83      margin:auto;
84  }
85
86  #loginformular{
87      margin-top:1.1em;
88      color: white;
89      float:right;
90  }
91
92  #loginformular > input{
93      border-style:solid;
94      border-color:black;
95      border-width: 0.188em;
96  }
97
98  #loginerror {
99      background-color:white;
100     border:medium solid;
101     border-color: red;
102     min-height:3em;
103     left:15em;
104     padding:1em;
105     position:absolute;
106     top:10em;
107     z-index:99;
108 }
109
110 .errorlist{
111     list-style-type:none;
112     color:yellow;
113     font-size:0.8em;
114 }
115
116 #username{
117     width: 10em;
118 }
119
120 #password{
```

```
121     width: 10em;
122 }
123 
124 #loginbutton{
125 /*  float:left;*/
126 }
127 
128 #content{
129     /*background-color:green;*/
130     margin:auto;
131     width: 60em;
132     min-height:40em;
133     border-bottom-color:#999999;
134     border-bottom-style:solid;
135 
136 
137 }
138 
139 #beschreibung{
140     float: left;
141     min-height:inherit;
142     width:40em;
143     background-image:url(../images/atmel.jpg);
144 }
145 
146 
147 #beschreibungtext{
148     background-color:#3B5998;
149     width:20em;
150     min-height:inherit;
151     opacity:0.9;
152 
153     /*border-style:solid;*/
154     /*border-color:black;*/
155     color:white;
156     border-left-style:solid;
157     border-left-color:#999999;
158 
159 }
160 
161 #beschreibungtextcontent{
162     padding-top:3em;
163 
164 }
165 
166 #about{
167     /*position:relative;
168     top:5em;
169     */
170     margin:auto;
171     width:15em;
```

```
172     border-style:solid;
173     border-color:white;
174     padding: 1em;
175 
176 }
177 
178 /*
179 #abouthead{
180     font-weight:bold;
181     font-size:1.2em;
182 }
183 */
184 
185 #registrierung{
186     float:left;
187     width:19.8em;
188     min-height:inherit;
189     background-color:#4B70BF;
190     border-right-color:#999999;
191     border-right-style:solid;
192 }
193 
194 #registerformular{
195     /*position:relative;
196     top:5em;
197     */
198 
199     width:13em;
200     margin:auto;
201     margin-top:3em;
202     color:white;
203 
204 }
205 
206 #registerformular > *{
207      width: 11em;
208      /*float:left;*/
209 }
210 
211 #registerformularfieldset{
212     border-color:white;
213     border-style:solid;
214 
215 }
216 
217 #registerformularfieldset input{
218     border-style:solid;
219     border-color:black;;
220     margin-bottom:1em;
221 
222 }
```

```
223
224
225 #registrierunginfo{
226     /*position:relative;
227     top:7em;
228     */
229     margin:auto;
230     width:13em;
231     color:white;
232     margin-top:1em;
233 }
234
235
236 #registinfo{
237     position:absolute;
238     left:34em;
239     top:10em;
240
241     width: 20em;
242
243     background-color:white;
244
245     border:medium solid;
246     border-color: red;
247
248     min-height:2em;
249
250     padding-bottom:1em;
251
252     z-index:99;
253 }
254
255 #registmessage{
256     padding:1em;
257 }
258
259 #registerok{
260     color:white;
261     margin:4em;
262     padding:1em;
263     border-style:solid;
264     border-color:white;
265 }
266
267 .push{
268     /*height:4em;*/
269
270 }
271
272 #footer{
273     background-color:#666666;
```

```
274     color:white;
275     min-height:5em;
276     margin-top:0;
277     /*background-color:black;*/
278     /*background-image:url(../images/blackpixel.png);*/
279     /*background-repeat:repeat-y*/
280     /*height:auto;*/
281 
282 }
283 
284 #footercontent{
285     margin:auto;
286     width:60em;
287     height:inherit;
288     background-color:#666666;
289     padding:1em;
290 }
291 
292 #footercontent > span{
293     margin:0.5em;
294     font-size:0.8em;
295 }
296 
297 .legend{
298     font-size:1.2em;
299     font-weight:bold;
300 }
301 
302 .clear{
303     clear:both;
304 }
305 
306 .altf4{
307     border-bottom-style:solid;
308     border-color:red;
309     overflow:hidden;
310     height:1em;
311     cursor:pointer;
312 
313 }
314 
315 .x{
316     float:right;
317     background-image:url(../images/cross.png);
318     overflow:hidden;
319     background-repeat:no-repeat;
320     width:1em;
321     color:transparent;
322     cursor:pointer;
323 }
324 
```

```
325
326
327 /* start-Template*/
328 #boardcontent{
329     border-left-style: solid;
330     border-left-color: #999999;
331
332     border-right-style: solid;
333     border-right-color: #999999;
334
335     border-bottom-style: solid;
336     border-bottom-color: #999999;
337
338     min-height:40em;
339
340     background-color: #3B5998;
341
342 }
343
344 #boardoverview{
345     list-style-type: none;
346     /*background-color: green;*/
347     min-height:inherit;
348
349     /*
350     border-left-style: solid;
351     border-left-color: white;
352
353     border-right-style: solid;
354     border-right-color: white;
355     */
356
357     width: 40em;
358     margin: auto;
359     color: white;
360
361
362     overflow: hidden;
363
364 }
365
366 .boardunit{
367     float:left;
368     margin:0.5em;
369
370
371     border-style:solid;
372     border-color:black;
373     border-width:0.15em;
374 }
375
```

```
376 .boardimage{
377     border:none;
378 }
379 
380 .boardname{
381     border-top-style:solid;
382     border-top-color:black;
383     border-top-width:0.15em;
384 
385     padding: 0.15em;
386 }
387 
388 #startnavigation{
389     float:right;
390 
391 }
392 
393 #home{
394     float:left;
395     margin-right: 0.5em;
396 }
397 
398 #adminlink{
399     float:left;
400     margin-right: 0.5em;
401 }
402 
403 
404 #logout{
405     float:left;
406     margin-right: 0.5em;
407     font-weight:bold;
408 }
409 
410 #usernamelogin{
411     color:white;
412     margin-top: 3em;
413 
414 }
415 
416 
417 
418 
419 /* board.html*/
420 #boardhead{
421     color:white;
422     margin-left:1.2em;
423 
424 }
425 
426 #oneboardcontent{
```

```
427     background-color: transparent;
428     overflow:hidden;
429     min-height:inherit;
430 
431 }
432 
433 #boardnavigation{
434     float:left;
435     list-style-type:square;
436     width:10em;
437     color:white;
438     min-height:inherit;
439 }
440 
441 #boardnavigation li a{
442     width: 10em;
443     display:block;
444 }
445 
446 
447 
448 #navigationfeedback{
449     float:left;
450     width: 40em;
451     margin-left: 2.5em;
452     margin-right: 4.5em;
453     background-color:white;
454     min-height:inherit;
455 }
456 
457 #imagereservate{
458     list-style-image: url("/site_media/mikrocontrollertemplate/images/padlock_closed.
        png");
459 }
460 
461 #imagerelease{
462     list-style-image: url("/site_media/mikrocontrollertemplate/images/padlock_open.png"
        );
463 }
464 
465 #imagestatusinformation{
466     list-style-image: url("/site_media/mikrocontrollertemplate/images/zoom.png");
467 }
468 
469 #imageturnon{
470     list-style-image: url("/site_media/mikrocontrollertemplate/images/playback_play.png
        ");
471 }
472 
473 #imageturnoff{
```

```
474     list-style-image: url("/site_media/mikrocontrollertemplate/images/playback_stop.png
            ");
475 }
476
477 #imagereset{
478     list-style-image: url("/site_media/mikrocontrollertemplate/images/playback_reload.
            png");
479 }
480
481 #imagehome{
482     list-style-image: url("/site_media/mikrocontrollertemplate/images/home.png");
483 }
484
485 /* isreserved.htm Template*/
486 #boardreserved{
487     width: 23em;
488     border-style:solid;
489     border-color:white;
490     border-width:0.1em;
491     margin-top:5em;
492     padding:1em;
493     margin:auto;
494     color: white;
495     position:relative;
496     top:8em;
497
498 }
499
500 /* myadmin.html */
501 #myadmin{
502     margin-top: 1em;
503     margin-left: 3em;
504     width:20em;
505 }
506
507 #myadmin ul li{
508     border-bottom-style:solid;
509     border-width:0.1em;
510 }
511
512 #myadminreleaseform{
513     margin-top: 1em;
514
515 }
516
517 .myadminbelegt{
518     list-style-image:url("/site_media/mikrocontrollertemplate/images/
            padlock_closed_black.png");
519     background-color: #FF7575;
520     padding:0.5em;
521 }
```

```
522
523 .myadminfrei{
524     list-style-image:url("/site_media/mikrocontrollertemplate/images/padlock_open_black
        .png");
525     background-color: #B3FFB3;
526     padding:0.5em;
527 }
528
529 /* newpassword.html */
530 #forgotpassword{
531     min-height: 40em;
532     background-color: #3B5998;
533     border-color: #999999;
534     border-left-style:solid;
535     border-right-style:solid;
536     border-bottom-style:solid;
537
538 }
539
540 #formforgotpassword{
541     width:25em;
542     margin:auto;
543     margin-top:10em;
544     background-color:white;
545     border-style:solid;
546     padding:3em;
547     border-color:black;
548 }
549
550 #formforgotpassword input{
551     border-style:solid;
552     border-color:black;
553 }
554
555 #newpasswordinfo{
556     width:20em;
557     margin:auto;
558     margin-top:10em;
559     background-color:white;
560     border-style:solid;
561     padding:3em;
562     border-color:red;
563 }
564
565 #newpasswordlink{
566     color:yellow;
567     font-size:0.7em;
568     margin-left:28em;
569 }
570
571 #newpasswordlink:hover{
```

```
572     background-color:transparent;
573 }
574
575
576
577 /* release.html */
578 #information{
579     position:relative;
580     top:5em;
581     left:5em;
582     width:25em;
583     padding:3em;
584
585     border-style: solid;
586 }
```

Listing D.15: style.css

D.2.2 js

```
1  $(document).ready(function(){
2      /* Tests: Difference between .text and .value function
3      $("#registinfo").text('Text value');
4
5
6      var textinhalt = $("#registinfo").text();
7
8      alert(textinhalt);
9
10
11     var textvalue = $("#registinfo").value();
12     alert(textvalue);
13     */
14
15     /* Function to remove infos after clicking X*/
16     $('.altf4').click(function(){
17         $('#registinfo').remove();
18     });
19
20     /* Delete error messages if all ist ok */
21     if($.trim($("#registmessage").text()) == 'ok'){
22         $("#registerformular").after('<div id="registerok">Eine Email mit dem
               Aktivierungscode wurde an Ihre Adresse versandt</div>').remove();
23         $("#registrierunginfo").remove();
24         $("#registinfo").remove();
25     }
26
27     /* Navigation Feedback */
28
```

```
29      $('.boardlink').click(function(){
30          var link = $(this).attr('alt')
31          /*alert(link);*/
32          $("#navigationfeedback").empty().html('<div id="information"><img src="/
                site_media/mikrocontrollertemplate/images/loading.gif" /><span>Loading ...
                </span></div>');
33
34          $('#navigationfeedback').load(link)
35      });
36
37      $('#usermail').click(function(){
38          $('#usermail').val('');
39      });
40
41
42
43  });
```

Listing D.16: script.js