



BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences



Beuth Hochschule für Technik Berlin
Fachbereich VI - Informatik und Medien
Luxemburger Str. 10, 13353 Berlin

Bachelorarbeit

Entwicklung einer Datenbank-Applikation für das "acaLoan-Raspi"-Projekt des PSE-Labors

**Development of a database application for the "acaLoan-Raspi"
project of the PSE laboratory**

Oleksandra Baga

Beuth Hochschule für Technik Berlin
Matrikelnummer 849852
Bachelor of Engineering (B.Eng.)
Computer Engineering - Embedded Systems
E-Mail: oleksandra.baga@gmail.com

Betreuer Prof. Dr. Christian Forler
Fachbereich VI - Informatik und Medien
Beuth Hochschule für Technik Berlin

Gutachter Prof. Dr. René Görlich
Fachbereich VI - Informatik und Medien
Beuth Hochschule für Technik Berlin

25.11.2020

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

25.11.2020..... Oleksandra Baga

Danksagung

Diese Abschlussarbeit entstand in Zusammenarbeit mit PSE-Labor der Beuth Hochschule für Technik Berlin.

An erster Stelle möchte ich Herrn Prof. Dr. Christian Forler bedanken, der meine Abschlussarbeit betreut hat. Für die hilfreichen Anregungen und die konstruktive Kritik möchte ich mich herzlich bedanken.

Ein besonderer Dank gilt den Mitarbeitern des PSE-Labor Andreas Döpkens und Brian Schüler, die eine spannende acaLab-Aufgabe für eine Abschlussarbeit vorgeschlagen haben, die ich für meine Bachelorarbeit gewählt durfte. Für das Engagement und die Beratungsgespräche bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Außerdem möchte ich mich bei meinem Mann Tymofii Baga bedanken, der mich während der Anfertigung dieser Bachelorarbeit unterstützt und motiviert hat. Ein besonderen Dank meinem Sohn Daniel, der während die Corona-Zeit wegen der geschlossenen Kita mich bei der Entwicklung des Systemdesigns und Implementierung zu Hause begleitet hat.

Contents

Abbildungsverzeichnis	I
Listings	III
Abkürzungsverzeichnis	V
1 Einleitung	1
1.1 Ziel der Arbeit	1
1.2 Aufbau der Arbeit	2
1.3 Motivation und Aufgabestellung	2
2 Theoretische Grundlagen	6
2.1 Raspberry Pi Board und Betriebssystem	6
2.2 Kontaktlose Smartcards MIFARE	8
2.3 Sender-Empfänger-System mit RFID	10
2.4 Django Framework	12
2.5 Grundlagen der Webanwendungen	14
2.5.1 Client-Server Kommunikation	15
2.5.2 Grundlagen der REST API	17
2.5.3 Clientseitiges JavaScript	18
3 Systemdesign	20
3.1 User Stories	20
3.2 Anforderungen	21
3.2.1 Funktionale Anforderungen	21
3.2.2 Nichtfunktionale Anforderungen	22
3.3 Anwendungsfälle	23
3.3.1 Ausleihe des Lab-Loan Boards	23
3.3.2 Ausleihe des Home-Loan Boards	24
3.3.3 Rückgabe des Lab-Loan Boards	25
3.3.4 Rückgabe des Home-Loan Boards	26
3.4 Systemarchitektur	26
3.4.1 Komponentendiagramm	26
3.5 aca-Loan Server	28
3.5.1 Klassendiagramm	28

3.5.2	Datenbank	29
	Entity Relationship Diagram	29
3.5.3	Endliche Zustandsmaschine	31
4	Implementierung	32
4.1	Register-Client	32
	4.1.1 Installation des Betriebssystem	33
	4.1.2 Installation der Treibers für RFID-Leser	34
	4.1.3 Spannungsproblem und Lösung	36
	4.1.4 Python Programmierung des RFID-Lesers	37
4.2	Server	41
	4.2.1 Erstellung acaLoan Django-Projekts	41
	4.2.2 Datenbankeinrichtung	43
	4.2.3 Einführung in das Django Admin	45
	4.2.4 Design der acaLoan-Website	47
	Erstellen der öffentlichen Schnittstelle	47
	Django HTML-Vorlage	49
	4.2.5 Implementierung des Anwendungsfällen	49
	Sequenzdiagramm	49
	Endliche Zustandsmaschine mit Django FSM	51
4.3	Display-Client	54
	4.3.1 Die Funktionsweise	55
	4.3.2 Start Webseite des acaLoan-Projekts	56
	4.3.3 Interaktive Benutzersitzung mit JavaScript	57
	4.3.4 Client-Server-Kommunikationsprotokoll	59
4.4	Automatisierten Tests	61
5	Analyse	64
6	Zusammenfassung	66
Glossar		I
Literaturverzeichnis		VII

Abbildungsverzeichnis

Abb. 1	Der Raspi Loan-Board	3
Abb. 2	Peripherieanschlüsse des Mikrocomputers	6
Abb. 3	Mifareproduktfamilie	10
Abb. 4	Verschiedenen RFID-Tags	11
Abb. 5	Die virtuelle Anforderungs- / Antwortschaltung	16
Abb. 6	UML Anwendungsfall: Ausleihe des Home-Loan Boards	24
Abb. 7	UML Anwendungsfall: Rückgabe des Lab-Loan Boards	25
Abb. 8	UML Komponentendiagramm	27
Abb. 9	UML Klassendiagramm des Servers	29
Abb. 10	Entity Relationship Diagram der DB	30
Abb. 11	Endliche Zustandsmaschine für RESTful Kommunikation	31
Abb. 12	ACR122U-A9 von Advanced Card Systems und Raspberry Pi	35
Abb. 13	RFID-Leser mit der Studentenkarte der Autorin und Tags	38
Abb. 14	Server nach Erstellung des acaLoanRaspiBoard-Projekts	42
Abb. 15	Screenshot von Django Admin Site mit Board Tabelle.	46
Abb. 16	Admin spezifische Site für CSV-Hochladen	46
Abb. 17	Screenshot von mockflow.com mit dem Design	47
Abb. 18	Teil der Index-Seite: dynamische generierte Tabelle mit verfügbaren im Labor Home-Loan Boards	48
Abb. 19	UML Sequenzdiagramm der erfolgreichen Ausleihe der Lab-Loan Raspi Board	50
Abb. 20	Django FSM mit Zustände und Übergänge	52
Abb. 21	Start Webseite des acaLoan-Projekts	56
Abb. 22	Webseite für eine neue Benutzersitzung mit dem Console-Log	57
Abb. 23	Webseite für einen Endzustand nach der Ausleihe des Boards	60

Abb. 24 Terminal nach der Ausführung aller Tests 63

Listings

4.1	Terminalbefehl für die Installation der GUI	34
4.2	Boot-Schleife beim Start des der Desktops	34
4.3	XML-Datei für VendorId und ProductID	36
4.4	Importieren der PCSC-Header-Dateien	37
4.5	Funktion SCardEstablishContext	38
4.6	Abgelesene Studentenkarte	39
4.7	Abgelesener Raspi-Tag	39
4.8	Monitor der angelegte und entfernte Smartkarten	39
4.9	acaLoanClient	39
4.10	Umgebungsvariable für den Serverstart	40
4.11	JSON-Schema	40
4.12	Python Version überprüfen	41
4.13	Erstellung acaLoan Django-Projekts	42
4.14	Deutsches Zeitformat in Django	42
4.15	Die zusätzliche Dateien in das Projekt hinzufügen	43
4.16	runserver-Befehl	43
4.17	Student Modell	44
4.18	Board Modell	44
4.19	Action Modell	44
4.20	Session Modell	45
4.21	Der Befehl migrate	45
4.22	Index-Ansicht in Django	48
4.23	Django Template Sprache in index.html	49
4.24	Session clean() für die Validation des Modells	51
4.25	Events-Ansicht in Django, JSON-Parsing	53
4.26	Events-Ansicht in Django, Aufruf von FSM Transitions	53
4.27	FSM Übergang beim Ablesen der Studentenkarte	53
4.28	FSM Übergang beim Ablesen der Board RFID-Tag	54
4.29	Fehler-HTTP-Statuscode	56
4.30	HTTP-Statuscodes für die neue Sitzung	56
4.31	function session_started()	58
4.32	Function refresh_session_state()	58

4.33 Änderung des DOM im Fall des Verbot der Home-Ausleihe	59
4.34 Sitzungsaktualisierungen mit JSON	59
4.35 Übergang aus dem Zustand rfid_state_active	60
4.36 Änderung des Statuses von active zu loaned	60
4.37 Factory für UIDs	61
4.38 StudentCardFactory	62
4.39 StudentFactory	62
4.40 Test für einen Student	63

Abkürzungsverzeichnis

AJAX Asynchrones Javascript und XML

API Application Programming Interface

ASGI Asynchronous Server Gateway Interface

ATR Answer-To-Reset

BGA Ball Grid Array

CPU Central processing unit

DOM Document Object Model

GPU Graphic processing unit

HTTP HyperText Transfer Protocol

JSON JavaScript Object Notation

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers

ISO International Organization for Standardization

MAC Media Accesss Control Layer

NFC Near Field Communication

SDRAM Synchronous dynamic random-access memory

SoC System-on-Chip

RAM Random-access memory

Raspi Raspberry Pi Board und Minicomputer im PSE-Labor der BHS

RFID Radio-frequency identification

URI Uniform Resource Identifier, Unified Resource Identifier

URN Uniform Ressource Name

Einleitung

Die Arbeit wurde im Rahmen des Projektes "acaLoan-Raspi" [TBc] des PSE-Labor geschrieben. Die acaLab-Projekte neben dem anfänglichen acaBot-Roboterprojekt vornehmlich Themen aus dem Bereich Praktischen und Technischen Informatik abdecken [TBa]. Während des Entwicklung des Projekts "acaLoan-Raspi" wird ein verteilte System entwickelt. Der Server wird mit Python Web Framework Django realisiert. Die Programmierung wird auf Python Programmierjungssprache gemacht. Mit der Verwendung von Open-Source-Bibliotheken wird den schrittweisen Prozess zur Integration der Hardware mit einer Low-Level-Bindung für die C-Bibliothek erledigt und somit ein RFID-Leser an Raspberry Pi Board angeschlossen. Benutzerorientierter Client, die einem Benutzer in einem Webbrower im Vollbildmodus angezeigt wird, wurde als interaktive Webseite in HTML implementiert und Styling durch CSS hinzugefügt,

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist die Entwicklung einer Datenbank-Applikation zur Verwaltung des Prozess von Ausleihe und Rückgabe der Hardware den Studentinnen und Studenten der Beuth Hochschule für Technik Berlin. Seit Jahren werden alle verknüpften mit der Ausleihverwaltung handlich verwaltet, was führt zum großen Arbeitszeitverbrauch. Es ist drei Bestandteile zu entwickeln: *Register-Client* mit angeschlossenen RFID-Leser, der die abgelesene UIDs der Studentenkarten dem *Server* schickt, der mit Python Web Framework Django realisiert ist und alle Datensätze zur Verwaltung behaltet und sich mit der Verifizierung von Studentenkarten und auszuleihenden Boards beschäftigt. Der letzte Bestandteil ist *Display-Client*, der sich mit der Anzeige der empfangene vom Server Information beschäftigt und zum Interaktion zwischen einem Endbenutzer und dem Server eine Schnittstelle zur Verfügung steht. Der Letzten nimmt in der REST Architektur die Rolle der Client.

Die entwickelte Software sollte es ermöglichen, nach der ihrer Lieferung eine Automatisierte Ausleihverwaltung für Übungsveranstaltungen im Studiengang Technische Informatik in den PSE-Labor zu betreiben.

1.2 Aufbau der Arbeit

Dieser Arbeit ist in fünf Teile unterteilt.

Kapitel 1 - Einführung.

Im ersten Teil wird die Aufgabe der Bachelorarbeit eingegrenzt und beschrieben, was die Motivation für Realisierung der Aufgabestellung ist.

Kapitel 2 - Theoretischen Grundlagen.

Im zweiten Kapitel wird der theoretische Rahmen der Arbeit dargelegt und betrachtet. Hierfür werden benötigte Grundlagen der Webanwendungen, Kontaktlose Smartcards MIFARE, die Aufbau des Sender-Empfänger-Systems mit RFID erläutert, sowohl die Grundlagen der Entwicklung der Webanwendungen mit Python Web Framework Django für Server und benutzerorientierter Client als auch die REST API erklärt.

Kapitel 3 - Systemdesign.

Im dritten Kapitel wird die Untersuchung und Analyse der zu entwickelten Software dargestellt. Der Kapitel präsentiert sowohl User Stories als auch funktionale und nichtfunktionale Anforderungen, gibt eine allgemeine Beschreibung über die architektonische Struktur und die einzelnen Komponenten, aus denen der fertige Lösung besteht.

Kapitel 4 - Implementierung.

Im vierten Kapitel wird die Aufgabestellung implementiert und die Programmierlungenaspekte des werden erläutern. Es ist über die drei Bestandteile zu lesen: Register-Client mit Raspberry Pi 3 Model B Rev 1.2 und angeschlossenen RFIR-Leser, Server mit Python Web Framework Django und benutzerorientierter Client. Auch wird die Methodik des Testens der Software dargestellt.

Kapitel 4 - Ergebnis und die Zusammenfassung.

In Kapitel 5 werden die Ergebnisse der Entwicklung analysiert.

1.3 Motivation und Aufgabestellung

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung einer Datenbank-Applikation für das PSE-Labor [TBc], das sich an der Beuth Hochschule für Technik Berlin befindet und seit fast 5 Jahren ein wichtiger Teil des Studiums im Studiengänge Technische Informatik und Medieninformatik ist. Die zwei Mitarbeiter, Andreas Döpkens und Brian Schüler, beitragen selbstmotiviert zur den Projekten, die zwar räumlich

im PSE-Labor der Beuth-Hochschule sind und betrieben werden können [TBa], wurden aber als "das acaLab" genannt, bei dem es sich sozusagen um ein virtuelles Labor in einem reellen Labor handelt. Die acaLab-Projekte sollen als interessante Mitarbeiter-Beiträge neben den individuellen Tätigkeiten im Fachbereich 6 der Beuth-Hochschule bereichern und werden deshalb auch exemplarisch jedes Jahr auf der Langen Nacht der Wissenschaften gezeigt [TBa]. An dieser Stelle ist auch noch anzumerken, die acaLab-Projekte aus den Mitteln des PSE-Labors finanziert werden und damit möglichst viele Studierende für lehr- und erkenntnisreichen Abschlussarbeiten eingeladen sind, da der Vergabe die Aufgaben aus den acaLab-Projekten als Abschlussarbeiten spart dem Fachbereich ein Budget. Die Erkenntnisse und Ergebnisse aus den acaProjekt-Tätigkeiten sind später in die Lehre einfließen zu lassen [TBa]. Meine Abschlussarbeit widmet sich dem neuen Projekt des Labors namens "acaLoan-Raspi".

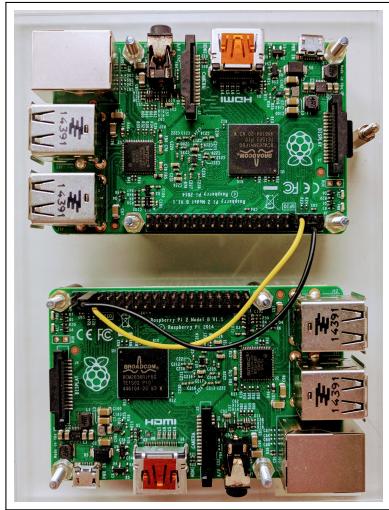


Abb. 1: Der Raspi Loan-Board

Während im PSE-Labor stattfindenden Übungsveranstaltungen im Studiengang Technische Informatik werden vorhandene im PSE-Labor die Raspberry Pi Minicomputers (kurz: Raspi die auf der Abbildung 1) an die Studierenden verliehen. Zu Beginn einer Laborübung werden die Raspi Boards den Studierenden vom Lehrkraft, der die Übung betreut, übergeben und am Ende der Laborübung zurückgezogen. Aus 16 vorhandenen im Labor Raspis, die markiert mit den Nummern 12-16 von Studierenden nach Hause (home-loan) genommen werden können [TBa].

Nachweislich ist das Vorgehen oft mit Reihe von Problemen verknüpft, die sich jedes Semester und fast jedes Mal wiederholen. Die folgenden Problemen wurden von Mitarbeitern des Labors bereits festgestellt und regelmäßig verlangsamen den Prozess der Verleihung und Übungsführung:

- Studierende kennen ihre am Semesteranfang zugewiesene Gruppennummer auch nach mehreren Wochen nicht und geben den Lehrkraft einen Board mit einer falschen Registriernummer, der einer anderen Gruppe früher zugewiesen wurde und nur von der zugewiesenen Gruppe benutzt werden darf.
- Studierende versuchen einen Board nach Hause auszuleihen, der zu den Lab-Boards gehört und nur im Labor während der Übungszeit verliefen werden darf.

Außerplanmäßig von Studierenden darf Lab-Board nicht ausgeliehen und auch mit nach Hause (home-loan) nicht genommen werden.

- Es gibt ein Verwaltungsaufwand für die ausleihbaren Home-Boards, die von den Studierenden für jeweils eine Woche mit nach Hause genommen werden können. Die Mitarbeiter müssen handlich die Studentenname, Matrikelnummer, Board und Zeit am Zettel registrieren und in einer Woche überprüfen, ob alle ausgeliehenen Boards pünktlich ins Labor zurückgekommen sind.
- Erfahrungsgemäß können Studierende nach Ablauf der Frist ein Ausleihgerät in einem sehr übelen Zustand der Verschmutzung oder Zerstörung zurückgeben, dass es besteht eine Notwendigkeit den Zustand des Gerätes stets zu kontrollieren, damit es immer bekannt wird, zum welchen Zeitraum Raspi Board zum letzten Mal funktionsfähig war und von wem ausgeliehen wurde.
- Falls gilt ein Raspi Board als verloren, es sollte eine Möglichkeit geben, alle vorherigen Ausleihen anzuschauen und festzustellen, von welchem Studierende es ausgeliehen und nicht zurückgegeben wurde. Mit den Zettelchen, auf denen einen Name von Studierende und eine Board Nummer gemerkt werden, ist es zu aufwändig nachvollziehen.

Somit ist schlusszufolgern, dass eine Notwendigkeit das Verleihprozedere für die Loan-Boards (Lab und Home) mit modernen Mitteln der Technischen Informatik zu lösen schon dringend besteht und eine lohnenswerte Aufgabe für zukünftige Abschlussarbeit ist, schließlich aus drei Teilen bestehen wird:

- Erstens wird an einem uComputer ein sogenannten Register-Client realisiert. Dafür ein RFID-Leser an uComputer angeschlossen wird. Register-Client ist neben dem Eingangstür eines kleinen Lagerraums des PSE-Labors zu platzieren ist, wo Raspi-Boards aufbewahrt werden. Es ist geplant, dass Studierende einen Board selbst aus dem Fach nehmen könnte und dann mit Hilfe des Register-Clients den genommenen Board auf sich oder seine Gruppe registrieren lassen. Der Register-Client hat selbst keinen Zugriff zur Datenbank und sollte nur die abgelesene Daten von der Smartcard der Studierende zur Server schicken.
- Zweitens ist ein Display-Client zu entwickeln, der den Studierenden es zulässt, die Begrüßung des System und eine Beschreibung die für Ausleihe notwendigen Schritten zu sehen. Es sollte in einem Browser-Fenster die aktuelle Server-Kommunikation und Auskunft angezeigt wird, ob die Ausleihe gelang oder ein Fehler aufgetreten war. Ein Android/iOS-Tablett ist eine gute Wahl für die technische Realisierung, da es die Kommunikation zwischen den Mensch und das System leicht und ohne erweiterte Hinweise zulässt.

- Drittens ist ein Web-Server für die Datenbank-Applikation schließlich zu implementieren. Er umfasst alle Datensätze über die vorhandenen im Labor Raspi-Boards, registrierten zum Kurs Studierenden und die abgewickelten Leihvorgänge. Web-Server wird mit einem Web-Framework Django erstellt. Django verfügt nun über die Funktionalität und Datenbasis, um die dafür erforderlichen Aktionen durchzuführen. Als Web-Framework bietet Django eine Reihe von Komponenten und Funktionen (Benutzeroauthentifizierung, Hochladen von Dateien, Umgang mit Daten usw.), die bei jeder Webanwendungen benötigt werden. Mit einem Web-Framework muss ein Entwickler keine Zeit damit verschwenden, denselben Code von Grund auf neu jedes Mal zu schreiben.

Nachdem nun grundlegenden Funktionsweisen der Bestandteilen des Verteilten Systems geklärt sind, geht es zur Auswahl der Hardware. Der wichtigste Aspekt beim Hardwarekauf ist einerseits das vorhandene Budget des PSE-Labors und zum Anderen die Aufgaben, die zusammenspielenden Hardware erfüllen sollen. Zunächst müssen die 16 für die Ausleihe zur Verfügung stehenden Raspi Loan-Boards mit einem geeigneten RFID-Tag ausgestattet werden. Es ist selbstklebende NFC Tags zu verwenden, die klein und dünn sind und lassen sich auf der Rückseite des Schutzschirms zu befestigen und ein Aussicht und Funktionen des Geräts nicht zu beeinflussen. Es ist nicht unerwähnt zu lassen, dass Tags nicht ausgelesen werden können, wenn diese auf einen metallischen Gegenstand/Oberfläche geklebt wurden, da die Kommunikation aus physischen Gründen zwischen Tag und Lesegerät gestört werden kann. Die Raspi Loan-Boards sind mit einem Schutzschirm aus dem ein transparenter thermoplastischer Kunststoff geschützt und es wurde ins Labor getestet, dass die geklebte auf der Rückseite RFID-Tag lassen sich ablesen und den Board auf RFID Chip-Kartenleser zu platzieren.

Theoretische Grundlagen

Als schon im Abschnitt "Einleitung" erwähnt wurde, bei der Aufgabestellung es um eine Entwicklung einer Webanwendung geht. Die zu realisierende Software basiert sich, wie die meisten Webanwendungen, auf einer Client-Server-Architektur, wobei der Client Informationen eingibt, während der Server die eingegebene Daten empfängt, bearbeitet und speichert.

2.1 Raspberry Pi Board und Betriebssystem

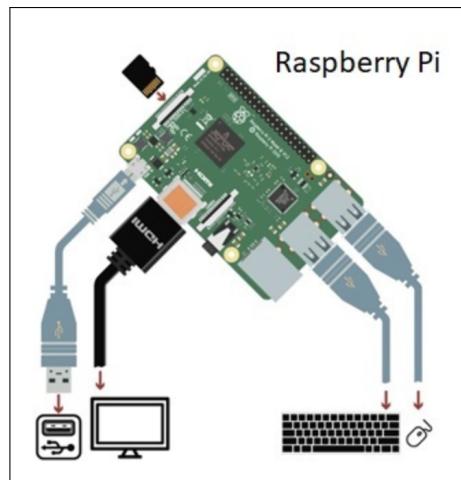


Abb. 2: Peripherieanschlüsse des Mikrocomputers

Raspberry Pi ist ein Einplatinencomputer, wobei die verschiedene Teile des Computers, die sich normalerweise auf separaten Platten befinden, werden hier nur auf einer dargestellt. Wie die meisten Einplatinencomputer ist der Raspberry Pi so klein wie eine Kreditkarte. Der Raspberry Pi ist eine kostengünstige Plattform - sein empfohlener Verkaufspreis beträgt weniger als 50€. Ein Mikrocomputer verfügt über alle Funktionen eines Personal Computer (PC): Prozessor, Speicher, Betriebssystem, Anschluss an einen Monitor (TV), die Vernetzung. Der Raspberry Pi verfügt im Gegensatz zu einem PC über zusätzliche Peripheriegeräte wie GPIO-Ports (General Purpose Input / Output). Über diese Pins kann der Mikrocomputer mit der elektronischen Welt der Sensoren, Bildschirmen und Aktoren interagieren. In Abbildung 2[Pri] sind die Anschlüsse schematisch dargestellt.

Für die Entwicklung der Abschlussarbeit und die Verwendung im Labor wird Raspberry Pi 3 Model B+ benutzt, an dem der RFID Leser angeschlossen wird. Das Modell "Raspberry Pi 3 B" ist eine kontinuierliche Weiterentwicklung zum Vorgängermodell "Raspberry Pi 2 B". Der Raspberry Pi 3 enthält einen Quad-Core-Prozessor mit 1,2 GHz von Broadcom und einen SDRAM-Arbeitsspeicher mit 1 GByte. Auf der Unterseite des Raspberry Pi befindet sich einen weiteren Chip, der wie ein kleines

schwarzes Plastikquadrat aussieht. Dies ist der Direktzugriffsspeicher (RAM) des Mikrocomputers. Wenn am Pi gearbeitet wird, wird die laufende Arbeit auf dem RAM gespeichert. Nur wenn die Daten explizit auf die microSD-Karte geschrieben werden, dann werden sie zum nächsten Verwendung gespeichert und nach dem Ausschalten nicht gelöscht. Der Raspberry Pi erlaubt den Austausch des aktuellen RAM-Chips nicht. Der aktuelle RAM-Chip ist als BGA-Gehäuse direkt auf die Oberseite der CPU gelötet. Das Hartlöten macht das Entfernen und / oder Ersetzen des RAMs sehr schwierig und deshalb es ist beim Kauf des bestimmtes Board auf die Größe des RAMs zu achten.

Die Besonderheiten des Raspberry Pi 3 ist, dass WLAN nach IEEE 802.11 b/g/n und Bluetooth Low Energy onboard sind und nicht durch externe USB-Adapter nachgerüstet werden müssen. Es ist für die vorgesehene Aufgabe wichtig ist, da die erforderlichen Komponenten nicht zusätzlich gekauft werden muss, was das Budget der PSE-Labors erspart und eine gewisse Zeit nicht geplant muss, dass die zusätzlich gekauften Komponenten irgendwie zum funktionieren bringen. Hier gab es in der Vergangenheit immer wieder Schwierigkeiten mit der Hardware-Erkennung oder Treiber-Probleme.

Die ARMv8-Architektur enthält Cortex-A53-Rechenkerne, die bei gleichem Takt schneller sind als die alten Cortex-A7-Kerne des Raspberry Pi 2 B. Von der 64-Bit-Fähigkeiten der ARMv8-Architektur profitiert allerdings nur neue Software, da 64 Bit auf der Hardware-Seite allerdings vom Betriebssystem und der Software auch unterstützt werden muss. Die folgende Liste gibt es eine kurzen Ansicht auf die Architektur des Mikrocomputers.

- System-on-Chip: BCM2837 64 Bit ARMv8 von Broadcom
- Prozessor: Quad-Core-Prozessor mit 1,2 GHz
- GPU: Dual-Core-GPU VideoCore IV mit OpenGL ES 2.0 und OpenVG mit Hardwarebeschleunigung und 1080p30 H.264 High-Profile-Decoding
- Arbeitsspeicher: 1 GByte LPDDR2-SDRAM
- WLAN: BCM43143 onboard für IEEE 802.11b, g und n im 2,4 GHz-Bereich
- Bluetooth: Bluetooth Classic und Low Energy (BLE) onboard (Bluetooth 4.1)

Eines der beliebtesten Betriebssysteme für den Raspberry Pi ist das Raspbian-Betriebssystem. Das Raspbian-Betriebssystem Raspbian basiert auf der ARM-Version von Debian 8 Jessie, ist für die Raspberry Pi-Hardware optimiert und enthält die

Standardprogramme wie die LibreOffice Office Suite, einen Webbrowser, Claws Mail, eine leichtgewichtige Desktop-Umgebung und einige Programmier-Lernwerkzeuge. Neuere Versionen des Betriebssystems Raspbian verfügen über einen völlig modernen Chromium-Browser, mit dem auch komplexe Webseiten korrekt angezeigt werden können. Um eine Information auf einem großen Bildschirm anzuzeigen, wird Chromium nur im Vollbildmodus gestartet, den Mauszeiger ausgeblendet und den Bildschirmschoner ausgeschaltet. Es gibt verschiedene Möglichkeiten, Raspbian auf einem Raspberry Pi 3 zu installieren. Die erste besteht darin, das Dienstprogramm NOOBS zu verwenden, die zweite darin, den Inhalt des Bildes des Betriebssystem direkt auf die Karte zu schreiben. Während der Entwicklung des Register-Clients wurde das Betriebssystem auf MicroSD-Karte geschrieben. Das Raspbian-Betriebssystem bootet von einer Micro-SD-Karte und das gesamte Betriebssystem läuft von der Karte.

Nach der kurzen Beschreibung der Raspberry Pi Architektur und sein Betriebssystem Raspbian ist schlusszufolgern, dass die Verwendung eines Raspberry Pi Mikrocomputers für die Implementierung des Register-Klient mit dem angeschlossenen RFID Leser ist eine lohnenswerte Entscheidung für in dieser Abschlussarbeit geschriebenen Aufgabe.

2.2 Kontaktlose Smartcards MIFARE

Anfänglich war die Smartcard eine Plastikkarte im ID-1-Format mit einer Größe von $85,60 \times 53,98$ mm und abgerundeten Ecken (eine Standardbank / Kreditkarte hat die gleiche Form und Abmessungen). Darin ist ein Mikrochip eingebettet, dessen Kommunikationskontakte zu einer Seite herausgeführt werden. Später erschienen Smartcards im ID-000-Format, sie wurden in Mobiltelefonen verwendet und sind uns als SIM-Karten bekannt. Dann erschienen Smartcards ohne externes Kontaktfeld, und sie begannen, einen Funkkanal für die Kommunikation und Energieübertragung zu verwenden. Smartcards haben keine eigene Stromquelle und sind auf eine externe angewiesen. Der Hauptvorteil einer Smartcard ist die physische Sicherheit der darauf gespeicherten Daten. Der Mikrochip ist sehr klein und alles notwendiges befindet sich darauf, ohne dass interne Kontakte hergestellt werden müssen, an die es sich zum Auffangen angeschlossen werden kann.

Der ungefähre Lebenszyklus einer Smartcard (z. B. einer Bankkarte mit Chip) besteht aus mehreren Phasen. Daran sind der Chiphersteller, der Smartcard-Hersteller, der Kunde und der Kunde beteiligt:

- **Herstellung von Chips/Prozessoren.** Zu diesem Zeitpunkt schreibt der direkte Hersteller des Chips nach der physischen Produktion die vom Hersteller der Smartcards bereitgestellte universelle und für alle Karten des gleichen Typs dieselbe Firmware auf.
- **Initialisierung der Smartcard.** Die Chips werden an den Kartenhersteller gesendet. Er ändert die Firmware nach Bedarf. Beispielsweise schreibt er eine eindeutige Seriennummer in jede Karte. Anschließend deaktiviert er die Möglichkeit, diese mit einer speziellen Anforderung zu ändern.
- **Smartcard-Herstellung.** Der Hersteller legt den Chip in Karten des gewünschten Formats ein und sendet diese an den Kunden, beispielsweise eine Bank.
- **Personalisierung.** Der Kunde schreibt unter Verwendung der Methoden der Firmware auf der Karte seine Anwendungen darauf, z. B. Bankgeschäfte sowie zusätzliche Daten, z. B. Kundenname, Kontonummer usw. Danach wird die Karte durch eine spezielle Anfrage finalisiert, wonach beispielsweise die Aufzeichnung neuer Anträge eingeschränkt wird.
- **Ausgabe.** Die Karte wird dem Kunden ausgegeben und vom Kunden verwendet.
- **Zerstörung.** Die Karte wird weggeworfen.

Einer der größten Hersteller von Smartcards ist die Firma NXP Semiconductors mit Hauptsitz in den Niederlanden. Mit einer großen Produktpalette schafft diese Lösungen für kontaktlose Zugangs- und Zeitkontrollen und sichere kontaktlose Automobilzugangskontrollen. Die weltweit meistgenutzte kontaktlose Chipkartentechnik „MIFARE“ wurde von NXP Semiconductors entwickelt und die Produktfamilie umfasst mittlerweile vier Produktreihen (siehe Abbildung 3) [NXP]. Alle MIFARE-ICs sind konform zur Norm ISO/IEC 14443 und erfüllen somit die Standards für die kontaktlose Kommunikation zwischen Chipkarten. Sie arbeiten mit 13,56 MHz und ihre Leseentfernung ist bis 10 cm möglich. Kontaktlose MIFARE-Smartcards verfügen über einen 1-KByte-Speicher, der in 16 Sektoren mit jeweils 16 Byte unterteilt ist. Die Datenspeicherdauer beträgt bis zu 10 Jahre. Die Anzahl der Umschreibzyklen beträgt 100.000 Zyklen. Neben NXP Semiconductors werden Chips für Mifare-Smartcards von der deutschen Firma Infineon im Rahmen einer Lizenzvereinbarung hergestellt [Too]. Nur Smartcards mit Chips von NXP Semiconductors und Infineon dürfen die Marke Mifare in ihrem Namen tragen. Nur Karten mit diesen Chips können als "Original" bezeichnet werden.

Die Studentenkarte der Beuth Hochschule wurden mit Mifare DESfire EV1 Kartenchip hergestellt. MIFARE DESFire EV1 ist die nächste Generation von Mifare Desfire mit einigen verbesserten Funktionen der Sicherheit und Verschlüsselung [Chi14, p.83]. Unberechtigte können sie aufgrund der AES-Verschlüsselung nicht auslesen. Zudem enthalten die Karten elektronisch keine persönlichen Daten [Tbb]. Alle auf der Karte gespeicherte Daten werden kodiert, z.B. mit der UID der einzelnen Karte verschlüsselt. MIFARE DESFire wird in vielen NFC- und RFID-Anwendungen verwendet, da er als sicherer Transponder mit einem von drei verschiedenen Typen von Verschlüsselung gesichert ist: Single DES, Triple DES oder AES. Im Allgemeinen wird AES als die sicherste Verschlüsselungsstufe der oben aufgeführten Methoden angesehen. Der AES-Authentifizierungsprozess besteht aus mehreren Schritten, in denen der NFC / RFID-Reader und das MDFEV1-Tag verschlüsselte Daten austauschen, um zu überprüfen, ob sie denselben Schlüssel verwenden. Während dieses Vorgangs wird ein Sitzungsschlüssel erstellt, der für bestimmte Befehle wie den ChangeKey-Befehl verwendet wird [JI].

MIFARE® contactless tag IC family overview															
Product features	MIFARE Ultralight®			MIFARE Classic®			MIFARE Plus®			MIFARE® DESFire®					
	Nano	EV1	C	EV1	SE	EV2	Light	EV3	EV2						
RF Interface					ISO/IEC 14443-2, Type A 13.56 MHz										
Protocol					ISO/IEC 14443-3				ISO/IEC 14443-3&4				ISO/IEC 14443-4		
UID - unique identifier	7-byte UID				7-byte UID, 4-byte NUID, Random ID				7-byte UID, Random ID						
Communication speed	106 Kbps				106-848 Kbps										
Memory size [Bytes]	40	48	128	144	1K	4K	1K	2K	4K	640	2K	4K	8K	16K	32K
Memory model	Compact, 4-byte pages				Compact, sectors & 16-byte blocks				Pre-configured file system				Flexible file system		
Crypto	3XDES				Crypto-1, AES				AES/LRP				DES/3XDES/3XDES/AES		
Key length	112-bit				48-bit				48-bit Crypto-1, 128-bit AES				128-bit AES, up to 168-bit DES		
Authentication	Password				3-pass mutual										
Communication security					Encrypted				Plain, CMACed, encrypted w. CMAC				✓		
MifareApp															
Transaction MAC	-				-				✓				✓		
Transaction Timer	-				-				-				-		
Security Level upgrade	-				card				sector per sector				-		
SL1SL3MaxMode	-				-				✓				-		
Multi key sets	-				-				-				✓		
Proximity check	-				-				✓				✓		
Virtual card concept	-				-				✓				✓		
Restrict update operations in SL1	-				-				✓				-		
Originality check features	ECC signature programmable	ECC signature	-	ECC signature	-	AES originality keys	AES originality keys, ECC signature	-	AES originality keys, ECC signature						
CC Certification	-				-				EAL5+				EAL5+		
ISO 7816-4 APDU	-				-				-				-		
NFC compliance	NFC Forum type 2 tag compliant				Not supported by majority of NFC devices				NFC capable in SL3				NFC Forum type 4 tag V2.0 compliant		
Target application	Public transport & event ticketing, loyalty programs, limited use tickets				Single application - not recommended for new design				Public transport / campcards / access management				Smart city platforms/ advanced mobility multi-applications/ micropayment/ loyalty programs/ access management		
Input capacitance [pF]	1750				17				17/70				17/70		
Multi applications	Supported via MAD				Supported via MAD				Fixed, single application				Dynamic		

Abb. 3: Mifareproduktfamilie

2.3 Sender-Empfänger-System mit RFID

Im vorherigen Kapitel 2.2 wurden die Grundlagen und Vorteilen der MIFARE-Technologie für Smartcards erklärt. Im aktuellen Kapitel wird über die Sender-Empfänger-System mit RFID beschrieben. Für die Implementierung der Aufgabe wird sowohl MIFARE-Smartcards als auch RFID-Tag-Mikrochips verwendet. Der letzte wird zum Identifizierung der auszuleihenden Raspi-Boards benutzt und auf der Rückseite des jeden Boards unter dem Schutzschirm geklebt werden. Beide Typs können mit dem RFID-Leser ACR122U-A9 von Advanced Card Systems abgelesen werden. Ab hier wird weiter nur den Begriff "Tag" sowohl für MIFARE-Smartcards

als auch für RFID-Tag-Mikrochips verwendet und den Unterschied in Namen nicht weiter angemerkt. Die gespeicherte Daten warten darauf, gelesen zu werden. Die Antenne des Tags erhält Energie von einer RFID-Leseantenne. Mit der Stromversorgung der internen Batterie oder des Lesegeräts sendet das Tag Funkwellen an das Lesegerät zurück. Der Leser nimmt die Funkwellen des Tags auf und interpretiert die Frequenzen als Daten. RFID-Tags, die über einen Teil des elektromagnetischen Spektrums gesendet werden, und die genaue Frequenz können ausgewählt werden, um Interferenzen mit anderen elektronischen Geräten zu vermeiden. Der Leser sendet ein Signal an das Tag und liest seine Antwort [Tec]. Der Leser verfügt über einen Funkempfänger, der als Transceiver bezeichnet wird und ein codiertes Funksignal an das Tag sendet. Das Signal aktiviert das Tag und der Transponder wandelt das Signal dann in eine nutzbaren Leistung um und sendet auf den Leser. Das Tag empfängt die Nachricht und antwortet dann mit seiner Identifikation und anderen Daten. Dies kann eine eindeutige Seriennummer des Produkts oder produktbezogene Daten sein. In der Abschlussarbeit wird nach der UID des Tags gefragt.



Abb. 4: Verschiedenen RFID-Tags

Während sich jedes Sender-Empfänger-System mit RFID hinsichtlich Gerätetyp und Komplexität unterscheidet, enthält jedoch jedes RFID-System mindestens die folgenden vier Komponenten: Leser, Antennen, Tags und Kabel. Das einfachste System kann aus einem mobilen Hand-RFID-Lesegerät (mit integrierter Antenne) und RFID-Tags bestehen, während komplexere Systeme mit Multi-Port-Lesegeräten, GPIO-Boxen, zusätzlichen Funktionsgeräten, mehreren Antennen, Kabeln und RFID-Tags ausgelegt sind und ein komplettes Software-Setup benötigen können. RFID-Tag-Typ Bestimmt das RFID-System. Derzeit gibt es drei verschiedene Arten von Tags: **aktiv, semi-passiv und passiv**. Ein aktives Tag sendet

mit seiner Batterie Radiowellen an einen Leser, während eine semi-passive Tag-Batterie in Gegenwart eines Lesers aktiviert wird. Aktive und semi-passive Tags werden über größere Entfernung gelesen. Sie senden hohe Frequenzen von 850 bis 950 MHz, die von 30 Meter oder mehr gelesen werden können. Zusätzliche Batterien können die Reichweite eines Tags auf über 90 Meter erhöhen. Passive RFID-Tags haben keine Batterie und verwenden die vom Lesegerät übertragene Funkenergie als Stromquelle. Diese Tags werden bis zu ein paar Meter entfernt gelesen und sind kostengünstiger [Tec]. Für die Markierung der Raspi-Boards im PSE-Labor werden die passive RFID-Tags verwendet, die für vorgesehenen Zwecken

ideal dienen. Die runde benutzte im Abschlussprojekt RFID-Tags sind auf der Abbildung 4 im Vergleich in der Größe zu 1-Euro Münze zu sehen. RFID-Systeme können nach Tag- und Lesertyp klassifiziert werden: passiver Leser - aktiver Tag (PRAT), aktiver Leser - passiver Tag (ARPT) und aktiver Leser - aktiver Tag (ARAT). Das ARPT-System wird im PSE-Labor eingesetzt und verfügt über einen aktiven Leser und empfängt Authentifizierungssignalantworten von passiven Tags. Ich möchte an dieser Stelle auch noch anmerken, dass der Initialisierungsprozess für eine kontaktlose Karte viel komplizierter als für eine Kontaktkarte ist. Das meiste davon wird vom sogenannten Antikollisionszyklus besetzt. Eine Kollision tritt auf, wenn mehr als eine Karte gleichzeitig auf das elektromagnetische Feld der RFID-Leser trifft und der diese Karten voneinander unterscheiden muss. Der Algorithmus dieses Prozesses ist sehr komplex und umfasst mehrere zehn Seiten Beschreibung in den Normen ISO/IEC 14443-2 und ISO / IEC 14443-3. Daher werde ich ihn hier nicht angeben, da es nicht von Entwickler wirklich etwas zu machen benötigt wird - das Terminal und der RFID-Leser sind voll damit beschäftigt.

2.4 Django Framework

Django ist ein Open Source-Webentwicklungsframework, das in der Python-Sprache geschrieben ist. Es wurde entwickelt, um so viele Prozesse wie möglich zu automatisieren, sodass es sich auf die Softwareentwicklung konzentriert werden können, ohne Zeit mit dem Einrichten des Servers verschwenden zu müssen. Django wurde ursprünglich so konzipiert, dass es lose Kopplung zwischen verschiedenen Teilen der Infrastruktur hat, sodass es unabhängig voneinander gearbeitet werden kann. Diese Unabhängigkeit bedeutet, dass es in der Entwicklungsprozess nur die Teile von Django verwendet werden kann, die benötigt werden, ohne sich um Probleme mit der Interdependenz von Komponenten kümmern zu müssen. Durch die Verwendung von Django wird die erforderliche Codemenge reduziert, wodurch das Schreiben von Webanwendungen schneller und die Wartung Ihrer Anwendung in Zukunft erheblich vereinfacht wird. Django folgt strikt dem DRY-Prinzip (Don't Repeat Yourself), dass jeder einzelne Code oder jede einzelne Daten an nur einem Ort gespeichert werden sollte. Dies vereinfacht und beschleunigt den Softwareänderungsprozess erheblich, da eine Anwendung, die geändert werden muss, an einem einzigen Ort ausgeführt werden muss.

Nach der Django Instalation und dem Erstellen eines Projekts ist es nützlich, die erstellte Projektstruktur zu betrachten [Foub]:

- **manage.py** ist ein Befehlszeilenprogramm, mit dem auf verschiedene Weise mit Django-Projekt interagiert wird . Diese Datei muss nicht geändert werden.

- **acaLoanRaspiBoard** Projektarbeitsverzeichnis enthält:
 - __init__.py*: Eine leere Datei, die Python mitteilt, dass dieses Verzeichnis für das Python-Paket bestimmt ist.
 - settings.py*: Die Einstellungen und Konfigurationsdatei für das Django-Projekt.
 - urls.py*: Die URL-Beschreibungsdatei für dieses Django-Projekt.
 - asgi.py*: Damit kann die Anwendung mit einem Webserver unter Verwendung des ASGI-Protokolls arbeiten.

Die generierte Datei *settings.py* enthält eine Grundkonfiguration für die Verwendung einer SQLite-Datenbank und eine Liste von Django-Anwendungen, die standardmäßig zum Projekt hinzugefügt werden sollen. In der Datei *settings.py* ist es möglich, den Debug-Modus des Projekts zu aktivieren / deaktivieren. Wenn "Debug=true" festgelegt ist, zeigt der Server bei Auftreten einer nicht erfassten Ausnahme detaillierte Fehlermeldung an. Es muss beim Wechsel zur Produktionsversion auf "false" gesetzt, da mit aktiviertem Debugging alle Benutzer vertrauliche Projektdaten sehen können.

Die Architektur von Django basiert auf den Ideen des The Model-View-Controller (MVC) Design Pattern, womit Anwendungsschicht, Benutzeroberfläche (UI) und Datenzugriffsebenen getrennt werden, sodass jede dieser Ebenen unabhängig von anderen geändert werden kann. Als Konzept ist das MVC-Entwurfsmuster wirklich einfach zu verstehen [Geo17, pp. 15-16]:

- **Das Modell (M)** ist ein Modell oder eine Darstellung Ihrer Daten. Es handelt sich nicht um die tatsächlichen Daten, sondern um eine Schnittstelle zu den Daten. Mit dem Modell können Sie Daten aus Ihrer Datenbank abrufen, ohne die Feinheiten der zugrunde liegenden Datenbank zu kennen. Das Modell stellt normalerweise auch eine Abstraktionsschicht mit Ihrer Datenbank bereit, sodass Sie dasselbe Modell mit mehreren Datenbanken verwenden können.
- **Die Ansicht (V)** ist das, was mit den Augen gesehen werden kann. Dies ist die Präsentationsebene für das Modell. Auf dem Computer wird die Ansicht im Browser für eine Web-App oder in der Benutzeroberfläche für eine Desktop-App angezeigt. Die Ansicht bietet auch eine Schnittstelle zum Sammeln von Benutzereingaben.

- **Der Controller (C)** steuert den Informationsfluss zwischen dem Modell und der Ansicht. Mithilfe der programmierten Logik wird entschieden, welche Informationen über das Modell aus der Datenbank abgerufen und welche Informationen an die Ansicht übergeben werden. Es erhält auch Informationen vom Benutzer über die Ansicht und implementiert die Aufgaben der Anwendung: entweder durch Ändern der Ansicht oder durch Ändern von Daten über das Modell oder durch Ändern die beiden Elementen.

2.5 Grundlagen der Webanwendungen

In diesem Kapitel werden die grundlegenden Begriffe zum Entwerfen und Erstellen eines Webanwendung erläutert. Hierbei konzentrieren wir uns auf die Probleme im Zusammenhang mit der Verarbeitung von HTTP-Anfragen und -Antworten (siehe Kapitel 2.5.2). Da in den letzten Jahren sich Webanwendungen rasant weiterentwickelt und die Desktop-Lösungen schrittweise ersetzt haben, es sollte auch nicht unerwähnt bleiben, welche Vorteile die Webanwendungen haben:

- **Zugriff von jedem Gerät** Die Webanwendung kann überall auf der Welt von einem Computer, Tablet oder Smartphone zugegriffen und verwendet werden. Notwendig ist, dass dem Gerät eine Internetverbindung zur Verfügung steht.
- **die Kostensparnis** Webanwendungen können auf allen Plattformen ausgeführt werden und müssen nicht mehr separat für Android und iOS entwickelt werden.
- **Anpassungsfähigkeit** Wenn native Anwendungen bestimmte Betriebssysteme erfordern, jedoch können jedes Betriebssystem (Windows, MAC, Linux usw.) und jeder Browser (Internet Explorer, Opera, FireFox, Google Chrome usw.) für die Arbeit mit einer Webanwendung.) verwendet werden.
- **Keine Software zum Herunterladen** Es ist günstig und einfach dem Endnutzer zu liefern, zu warten und zu aktualisieren. Das Aktualisieren auf die neueste Version erfolgt beim nächsten Laden der Webseite.
- **Netzwerksicherheit** Das Websystem verfügt über einen einzigen Einstiegspunkt, der zentral geschützt und konfiguriert werden kann.
- **Skalierbarkeit** Mit zunehmender Belastung des Systems ist es nicht erforderlich, die Leistung des Computer von Endbenutzer zu erhöhen. Mit einer Webanwendung kann in der Regel nur mithilfe von Hardwareressourcen eine

größere Datenmenge verarbeitet werden, ohne den Quellcode neu zu schreiben und die Architektur ändern zu müssen.

- **Verhinderung von Datenverlust** Benutzerdaten werden in der "Cloud" gespeichert, für deren Integrität die Hosting-Anbieter verantwortlich sind, deswegen vom Verlust geschützt, falls die Festplatte des Computers beschädigt wird.

Ich möchte an dieser Stelle auch noch ein Konzept des Caches erklären, bei dem häufig verwendete Daten, anstatt jedes Mal aus der Datenbank abgerufen, berechnet oder auf andere Weise vorbereitet, an einem schnell zugänglichen Ort gespeichert werden. Zum Beispiel erhielt Django eine Anfrage, Daten für ein Diagramm in einem Bericht abzurufen. Daten aus der Datenbank werden abgeholt, vorbereitet und abgelegt in einer Datenbank mit schnellem Zugriff z.B. "memcached", die beispielsweise für eine Stunde zwischengespeichert wird. Bei der nächsten Anfrage werden die notwendigen Daten sofort von "memcached" erhalten und an das Frontend gesendet. Wenn es festgestellt wird, dass die Daten nicht mehr relevant sind, werden sie ungültig beziehungsweise gelöscht aus dem Cache.

2.5.1 Client-Server Kommunikation

Als nächstes geht die Frage nach, wie Client und Server miteinander kommunizieren können. Der Client spricht mit dem Server über das HTTP-Protokoll. Das HTTP-Protokoll setzt die Verwendung einer Client-Server-Datenübertragungsstruktur voraus. Es verwendet normalerweise Port 80 als Transportschichtprotokoll - TCP. Das HTTP-Protokoll ist in RFC 1945 (HTTP 1.0 [TBL]), 2068 [RF] und 2616 (HTTP 1.1) definiert. Die Clientanwendung generiert eine Anforderung und sendet sie an den Server. Anschließend verarbeitet die Serversoftware diese Anforderung, generiert eine Antwort und sendet sie an den Client zurück [RR03, pp.33-34]. Die Clientanwendung kann dann weiterhin andere Anforderungen senden, die auf ähnliche Weise behandelt werden. Wenn ein Webserver eine Anforderung zum Bereitstellen einer statischen Webseite erhält, sendet er die Seite direkt an den Browser.[Ado] Wenn jedoch eine dynamische Seite angefordert wird, sind die Handlungsweise des Webservers nicht so einfach. Der Server übergibt die Seite an ein spezielles Programm, das die letzte Seite bildet. Ein solches Programm wird als Anwendungsserver bezeichnet. Der Anwendungsserver liest den Code auf der Seite, rendert die letzte Seite gemäß dem gelesenen Code und entfernt sie dann von der Seite. Als Ergebnis all dieser Operationen wird eine statische Seite erhalten, die an den Webserver übertragen wird, der sie wiederum an den Client-Browser sendet. Alle Seiten, die der Browser empfängt, enthalten nur HTML-Code [Ado].

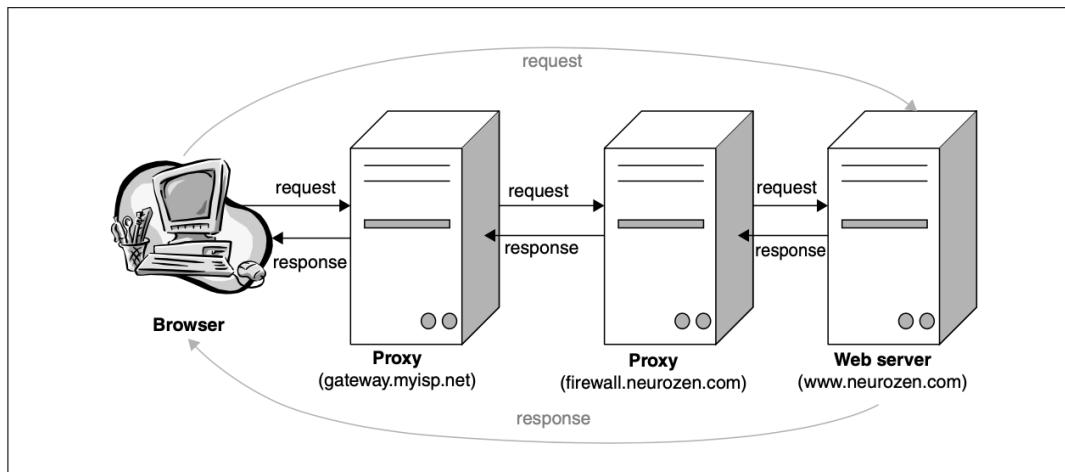


Abb. 5: Die virtuelle Anforderungs- / Antwortschaltung

Dieses grundlegende Paar Anforderung-Antwort wird im nächsten Kapitel 2.5.2 untersucht. In der Praxis kommunizieren Server und Browser selten direkt - dazwischen befinden sich ein oder mehrere Proxys. Eine Verbindung ist als virtuelle Verbindung definiert, die aus HTTP-Agenten besteht, einschließlich des Browsers, des Servers und der am Austausch teilnehmenden Zwischenproxys (Abbildung 5 [RR03, pp.33-34]).

HTTP definiert eine Reihe von Anforderungsmethoden, um die gewünschte Aktion anzugeben, die für eine bestimmte Ressource ausgeführt werden soll. Sie werden als HTTP-Verben bezeichnet. Diese Anforderungsmethoden sind unten aufgeführt:

- **GET** ist die einfachste Anforderungsmethode. Wenn eine URL in Browser eingeben oder auf einen Hyperlink geklickt wird, um eine andere Seite zu besuchen, verwendet der Browser die GET-Methode, um die Anforderung an den Webserver zu senden.
- **POST** wird benutzt, um eine neue Ressource zu erstellen. POST-Anforderungen enthalten normalerweise Daten zum Erstellen einer neuen Ressource.
- **PUT** ist zum Aktualisieren einer vorhandenen Ressource. Der Inhalt kann aktualisierte Daten für die Ressource enthalten.
- **DELETE** ist zum Löschen einer vorhandenen Ressource.

2.5.2 Grundlagen der REST API

In der Client-Server-Kommunikation haben wir zunächst zwei Partner: den Client und den Server. Um die Kommunikation zwischen diesen beiden Partnern zu verstehen, müssen wir einige einfache Themen kennen:

- **Anforderungen** werden vom Client gesendet, um den Server nach Daten wie Dateien zu fragen oder den Server über Ereignisse zu informieren, z. B. dass sich ein Benutzer mit seinen Anmeldeinformationen anmelden möchte
- **Antwort** wird vom Server an den Client gesendet und ist die Reaktion des Servers auf eine Anforderung des Clients.

Die ankommenden Anforderungen muss vom Server interpretiert und weitergeleitet werden. Während dieses Schritts übernimmt der Webserver die Verantwortung für Bestimmung, welche Maßnahmen zur Verarbeitung der Anforderung erforderlich sind. Diese Aktion kann das einfache Abrufen einer statischen HTML-Datei oder der Zugriff zur Datenbank sein. Das kann komplexe Funktionen ausführen, die mit einer vollständigen Webanwendung verbunden sind [RR03, p.204]. Wenn ein Benutzer versucht, auf eine Webanwendung zuzugreifen, sendet der Browser zuerst eine HTTP-Anforderung an einen Webserver. Die mit der Anforderung verknüpften Informationen (häufig als Anforderungskontext bezeichnet) umfasst die weitere Informationen, die in der URL, dem Anforderungsheader und dem Nachrichtentext enthalten sind. Es enthält auch Sitzungsinformationen, die den Anwendungsstatus über aufeinanderfolgende Anforderungen hinweg beibehalten [RR03, p.205].

Die Client-Server-Kommunikation ist zustandslos. Dies bedeutet, dass sich der Server keine Informationen über den Status des Clients merken muss, während der Client alle Informationen in seine Anforderung aufnehmen muss.

Das REST- oder RESTful-API-Design (Representational State Transfer) ist eine Client-Server-Architektur und nutzt normalerweise HTTP-Protokoll [ML]. Der Server speichert und bearbeitet Informationen und stellt sie dem Benutzer auf effiziente Weise zur Verfügung. Der Client nimmt diese Informationen und zeigt sie dem Benutzer an und verwendet sie, um nachfolgende Informationsanforderungen auszuführen. Diese Trennung von Funktionsweise ermöglicht es sowohl dem Client als auch dem Server, sich unabhängig voneinander zu existieren und entwickelt werden, da nur die Kommunikationsschnittstelle gleich bleiben muss [Dix]. In einem REST-API-Aufruf werden eine Teilmenge der HTTP-Methoden für die Aktionen verwendet, die wir ausführen müssen. Diese Methoden wurden oben im Kapitel 2.5.1 schon erklärt. REST-APIs geben normalerweise die von Benutzer angeforderten Daten im

JSON-Format zurück, dessen Payload eine String (Zeichenfolge) ist. Bevor die Daten tatsächlich verwendet werden können, müssen diese Zeichenfolge analysiert und in Objekte umgewandelt werden. Die Funktionsweise das REST-APIs von acaLoan-System sind in Kapitel 4.3.4 nachzulesen.

2.5.3 Clientseitiges JavaScript

Das Document Object Model (DOM) ist eine Programmierschnittstelle für HTML- und XML-Dokumente. Es stellt die Seite dar, sodass Programme die Dokumentstruktur, den Stil und den Inhalt ändern können [conb]. Clientseitiges JavaScript (CSJS) ermöglicht die Verbesserung und Bearbeitung von Webseiten über den soforten Zugang zu den Knoten des DOM: in einer Browserumgebung hat Ihr Code Zugriff auf Dinge, die nur vom Browser bereitgestellt werden, wie das Dokumentobjekt für die aktuelle Seite, das Fenster, Funktionen wie Warnungen, die eine Nachricht anzeigen usw. Die Hauptaufgaben von clientseitigem JavaScript sind die Validierung Eingabe, Animation, Bearbeiten von UI-Elementen, Anwenden von Stilen und einige kleine Berechnungen können durchgeführt werden. Bei der Webentwicklung ist es der Browser auf dem Computer des Benutzers, der diesen Javascript Code ausgeführt [conc]. Bei der Implementierung der Aufgabestellung für acaLoan-Systen wird davon ausgegangen, dass der Chrome-Browser als Rendering-Frontend verwendet wird, dies ist jedoch nicht streng Anforderung und der Client sollte ohne Probleme mit anderen modernen Browsern arbeiten, da der Code des clientseitigen JavaScripts in einer Mehrheit von Browsern ausgeführt werden kann muss.

Ein wichtiges Merkmal von JavaScript ist die Möglichkeit, Ereignishandler (event handlers) zu definieren - beliebige Codeteile, die ausgeführt werden, wenn ein bestimmtes Ereignis auftritt. Normalerweise werden diese Ereignisse vom Benutzer ausgelöst, wenn er beispielsweise die Maus über einen Hypertext-Link bewegt, einen Wert in ein Formular eingibt oder in einem Formular auf die Taste Senden klickt. Diese Funktion zur Ereignisbehandlung hat eine wichtigste Bedeutung, da für die Programmierung mit grafischen Oberflächen wie HTML-Formularen ein ereignisgesteuertes Modell erforderlich ist. JavaScript kann jede Art von Aktion als Reaktion auf Benutzerereignisse auslösen. Typische Beispiele könnten sein, eine spezielle Nachricht in der Statuszeile anzuzeigen, wenn der Benutzer die Taste drückt. Sodass wird es im acaLoan-System nach dem Drucken die Taste "Loan / Return Board" und erfolgreichen Ablesen der Studentenkarte den Name der Studierende und die Nummer der Board angezeigt.

In diesem Kapitel wurden die Grundlagen der Webanwendungen und des HTTP-Protokolls erörtert. Diese Diskussion war nicht als umfassende Beschreibung aller

Funktionen gedacht, eher als Überblick zum Verstehen der zukünftigen Verwendung der entwickelte Software im PSE-Labor.

Systemdesign

Die Softwareentwicklung beginnt mit einer Beschreibung der Bedürfnisse und ihrer Analyse. Je genauer und korrekter die Beschreibung der Softwareanforderungen und deren Analyse ist, desto einfacher ist es, alle nachfolgenden Schritte abzuschließen. Das Hauptproblem in dieser Phase ist der Unterschied in den Ansichten des Kunden (in dem Fall der vorliegenden Abschlussarbeit sind die Kunden die PSE-Labor Mitarbeiter) und des Entwicklers (die Autorin der Abschlussarbeit). Es wurde auch die Entscheidung getroffen, mit welchen Hardware ist die Aufgabestellung zu implementieren. Jedoch während der Entwicklung der Register-Client (der einer von drei Bestandteilen der Software, an dem ein RFID-Leser angeschlossen werden muss), wurde schließlich ein RFID-Leser gewechselt. Der Fall ist im Kapitel 4.1.2 nachzulesen. Im Rahmen der Analyse und des Systemdesigns wurden die Struktur und Zusammenhänge der Elemente des zu entwickelnden Systems untersucht. Das Ergebnis dieser Untersuchung enthält genügend Informationen, um das System zu implementieren und ist unten in folgenden Kapitels detailliert beschrieben.

3.1 User Stories

Zuerst wurden die User Stories erstellt, die eine diskutierte Darstellung der Absicht (Endbenutzer muss/will so etwas tun) zeigen können. Es ist mithilfe der User Stories zu klären, was die zu entwickelnde Datenbank leisten soll. Es wird auf die Frage konzentriert, welche Daten in der Datenbank gespeichert werden sollen. Der Text der User Story selbst sollte die Rolle / Aktionen des Benutzers im System, seine Bedürfnisse und den Gewinn erläutern, den der Benutzer nach dem Auftreten der Story erhält. Zum Beispiel: Wie *<Benutzerrolle / Charakter>, ich <möchte etwas bekommen>, <für diesen und jenen Zweck>*. Während des Schreibens der User Story wurden zwei Gruppe von Stakeholders definiert: die Studierende (Beuth Studentinnen und Studenten) und der Admin (PSE-Labor Mitarbeitern). Es wurden die folgenden User Stories erstellt, die später während der Entwicklungsphase implementiert wurden:

- As a student, I want **to loan a board** so I can work at lab
- As a student, I want **to loan a board** so I can work at home
- As a student, I want **to list boards assigned to me** so that I am sure I to return

- As a student, I want **to return a board from lab work** so that I can loan again
- As a student, I want **to return a board from home work** so that I can loan again
- As a student, I want **to initiate session** so I can loan a board
- As a lab admin, I want **to mark a board** so it can be loaned for home or for lab
- As a lab admin, I want **to terminal session** with timeout so that students can use the loan station
- As a lab admin, I want **to block a student** so that they won't be able to loan a board
- As a lab admin, I want **to see all students** registered on the course so that I can manage their profiles
- As a lab admin, I want **to see all loaned boards** so that I can know their expected return date
- As a lab admin, I want **to register students** so that they are able to loan boards
- As a lab admin, I want **to register new boards** so that they can be loaned
- As a lab admin, I want **to delete student's record** when semester ends so that they are not stored anymore in database

3.2 Anforderungen

Während die meisten neuen Funktionen mithilfe der User Stories aus Anwendersicht definiert werden sollten, ist dies nicht immer machbar oder sogar hilfreich, wenn es zu Sicherheitsfunktionen oder Infrastrukturanforderungen kommt, die nicht kundenorientiert sind. Es gibt zwei Arten von Anforderungen: funktionale Anforderung und nichtfunktionale Anforderungen. Während der Analysephase wurden die beiden Arten von Anforderungen definiert.

3.2.1 Funktionale Anforderungen

Eine funktionale Anforderung beschreibt, was ein Softwaresystem tun sollte. Es werden die folgenden funktionalen Anforderungen definiert:

- The background color for all windows in the application will be white and have a hexadecimal RGB color value of 0x0000FF.
- The colors of design guidelines of Beuth Hochschule will be used.

- The software automatically validates whether a student is able to loan a board for a homework.
- The software automatically shows the information about boards that student already loaned.
- Student will see their name after they scanned their valid student card.
- Student will see board's number after they scanned a Raspberry Board.
- If student can not loan a board they will see an information message on a display.
- If a student can not loan a board the session should be terminated
- If an error during the loan process is occurred the student will see detailed information so they can later talk with an admin about the issue.
- Error states will be marked with red color on the page
- Succeeded states will be marked with a green color on the page

3.2.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen bestimmen nicht die Funktionen, sondern die Eigenschaften des Systems: Leistung, Zuverlässigkeit, Verfügbarkeit, Skalierbarkeit und eine Reihe anderer Parameter, die das System einschränken und verbessern sollen.

- The Server has to be implemented using a modern Python web framework Django.
- The user Interfaces (frontend) shall be implemented as HTML pages with dynamic content inside based on Django Templates.
- The admin views must require authorization. A view decorated with this function will be executed normally only if the logged user has admin rights.
- The SQLite relational database will be used in order to store students, boards and loan records
- For terminating the session automatically after timeout the command line application (CLI) will be implemented using python and will be run on the server.
- Users must use for the initial login their student card. Moreover, every next login will be done with the same card.
- Students never allowed to loan home board longer than 1 week (7 calendar days). Such attempt should be reported to the security administrator.
- Students never allowed to loan lab board longer than 120 minutes. At the end of the exercise administrator should be notified if the board was not returned.

- Loan process can not be started if a student was not properly registered on the course
- Every unsuccessful attempt by a user to loan/return an item shall be recorded on an audit trail.
- Only one active session is allowed for loan/return process. No multi-user mode is intended
- If the current session is inactive longer than 180s the session will be terminated and should be restarted
- The actions made on RFID-Reader should be displayed on the screen with an acceptable delay for a human (less than 5 seconds)

Während die User Story selbst die Verbindung zwischen der menschlichen Wahrnehmung und der technischen Umsetzung ist, können mithilfe der Anforderungen die Betriebsfunktionen und Einschränkungen des Systems beschrieben werden, die seine Funktionalität verbessern.

3.3 Anwendungsfälle

Anwendungsfalldiagramme beschreiben selbst kein Verhalten und keine Abläufe, sondern nur die Zusammenhänge zwischen einer Menge von Anwendungsfällen und den daran beteiligten Akteuren. Sie eignen sich daher besonders gut, um für Benutzer des Systems (Akteure) relevante funktionale Anforderungen an ein System zu analysieren [SSG]. Während der Analyse der Systemanforderungen wurde es festgestellt, dass die Software aus vier zentralen Anwendungsfällen bestehen sollte. Diese sind: Die Ausleihe des Lab-Loan Boards, die Rückgabe des Lab-Loan Boards, die des Ausleihe des Home-Loan Boards, die Rückgabe des Home-Loan Boards. Jeder von diesen Anwendungsfällen kann erfolgreich oder mit dem Fehler beendet werden.

3.3.1 Ausleihe des Lab-Loan Boards

Ein Studierende soll während der Übung im PSE-Labor ein Lab-Loan Board für 120 Minuten ausleihen, um die Aufgaben der Lehrkraft zu erledigen. Die 120 Minuten entsprechen die 90 Minuten der Übung + zusätzlichen 15 Minuten vor und nach der Übung, um Board aus-/einzuwickeln, an-/auszuschalten und laufenden Arbeit am Ende der Übung zu speichern. Ein erfolgreichen Szenario besteht aus den folgenden Schritten: nachdem hat ein Studierende ein Lab-Loan Board aus dem Schrank genommen drückt er die Start-Taste am Bildschirm des Display-Client. Dann lässt er seine Studentenkarte am Register-Client über RFID-Leser abzulesen und danach

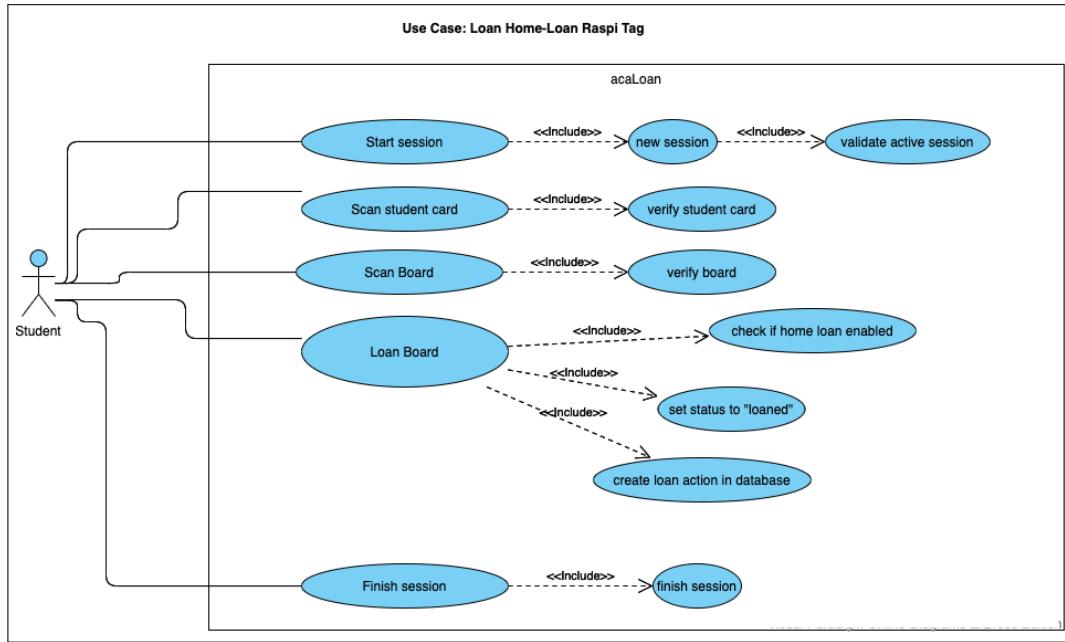


Abb. 6: UML Anwendungsfall: Ausleihe des Home-Loan Boards

sieht eine Begrüßung des Systems und seinen eigenen Name. Ein Studierende mithilfe des RFID-Lesers liest als nächsten der RFID-Tag des Lab-Loan Board ab, dessen Nummer wird nach dem Ablesen am Bildschirm des Display-Client unter dem Name der Studierende gezeigt. Dann muss die Taste "Loan Board" gedrückt werden. Nach der Bestätigung der Ausleihe drückt der Studierende eine Taste "Finish" und somit wird Anwendungsfall erfolgreich beendet.

Jedoch sind auch Fehlerzustände für den Anwendungsfall vorgesehen. Falls der Studierende zum Kurs von Admin nicht registriert wurde, wird ein Ausleihvorgang nach dem Ablesen der Studentenkarte sofort mit der Fehlermeldung "*Invalid student card*" im Fehlerzustand terminiert. Falls der Studierende hat schon einen Lab-Loan Board ausgeliehen und versucht jetzt einen weiteren an sich zuweisen, wird ein Ausleihvorgang sofort mit der Fehlermeldung "*Same bord type*" im Fehlerzustand terminiert. Falls es auf einen Versuch käme, ein dritten Board auszuleihen, wird Fehlermeldung "*Maximum boards reached*" angezeigt und sofort im Fehlerzustand terminiert. Falls ein nicht bekannte RFID-Tag am RFID-Leser anstatt der Lab-Boards wird abgelesen, kommt es zu einem Fehlerzustand "*Status error*", da ein RFID-Tag keinem Board mit irgendwelchen Zustand in System zugewiesen werden kann.

3.3.2 Ausleihe des Home-Loan Boards

Der folgenden Anwendungsfall ist ähnlich zum vorherigen Anwendungsfall "Ausleihe des Lab-Loan Boards" aus dem Kapitel 3.3.1. Der Home-Loan Board darf in diesem Fall für 7 Kalendertagen ausgeliehen werden. Es gibt jedoch einen wichtigen Unterschied

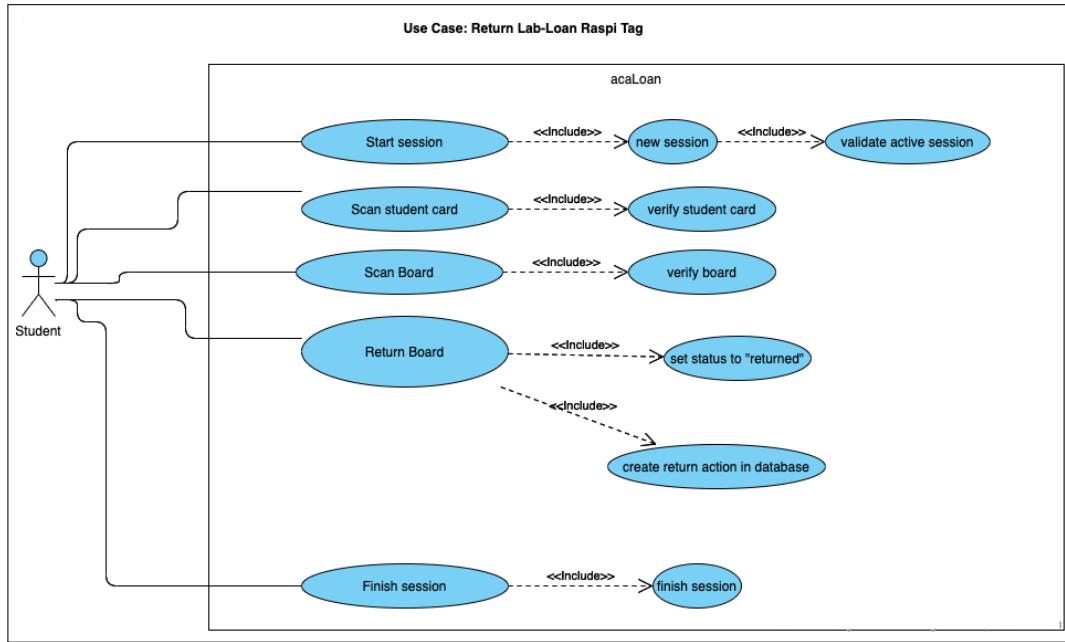


Abb. 7: UML Anwendungsfall: Rückgabe des Lab-Loan Boards

zur Lab-Ausleihe: solange ein Student ein Teilnehmer des Moduls ist, kann es dem Studierende nicht verweigert werden, während der Übung des Lab-Loan Board für die Lösung der Aufgaben zu benutzen und auszuleihen. Jedoch für Home-Ausleihe gilt eine andere Regelung, wann es ihm die Home-Ausleihe-Absicht gesperrt werden kann. Falls das Gerät in einem sehr übeln Zustand der Verschmutzung oder Zerstörung ohne eine Besprechung mit dem Administrator zurückgegeben wurde, darf der Studierende ein Gerät nächsten Mal nicht ausleihen. Falls es auf einen Versuch käme, ein Home-Loan Board auszuleihen mit dem ungesetzten im Profil Flag `"home_loan_is_enabled"`, wird Fehlermeldung `"Home loan disabled"` angezeigt und der die Sitzung sofort im Fehlerzustand terminiert. Der oben geschrieben Anwendungsfall ist auf der Abbildung 6 zu sehen.

3.3.3 Rückgabe des Lab-Loan Boards

Das erfolgreichen Szenario als folgenden vorgesehen: nachdem hat ein Studierende mit einem Lab-Loan Board im Raum PSE-Labor angekommen ist, drückt er die Start-Taste am Bildschirm des Display-Client. Dann lässt er seine Studentenkarte am Register-Client über RFID-Leser abzulesen und danach sieht eine Begrüßung des Systems und seinen eigenen Name. Ein Studierende mithilfe des RFID-Lesers liest als nächsten der RFID-Tag des Lab-Loan Board ab, dessen Nummer wird nach dem Ablesen am Bildschirm des Display-Client unter dem Name der Studierende gezeigt. Dann muss die Taste "Return Board" gedrückt werden. Nach der Bestätigung der Rückgabe drückt der Studierende eine Taste "Finish" und somit wird Anwendungsfall

erfolgreich beendet. Der oben geschrieben Anwendungsfall ist auf der Abbildung 7 zu sehen.

Es sind auch Fehlerzustände für den Anwendungsfall vorgesehen. Es wird die gleiche Fehlerzustände "*Invalid student card*" und "*Status error*" benutzt. Darüber hinaus wird ein Fehlerzustand "*Return error*" vorgesehen, falls es zum einem Versuch käme, ein Lab-Board zurückzugeben, der nicht von angemeldeten in der Sitzung Studierende ausgeliehen wurde.

3.3.4 Rückgabe des Home-Loan Boards

Der folgenden Anwendungsfall ist ähnlich zum vorherigen Anwendungsfall "*Rückgabe des Lab-Loan Boards*" aus dem Kapitel 3.3.1. Es wird keine erweiterten Beschreibung des Anwendungsfall benötigt, da erfolgreichen Szenario wiederholt sich. Es ist erwartet, dass bei den Home-Loan Boards es häufiger sein kann, dass Board nicht rechtzeitig von einem Studierende zurückgegeben wird. Falls der Rückgabetermin versäumt ist, wird trotzdem ein Board ohne Fehlermeldung zurückgenommen, da die Rückkehr alle Boards ins Labor hat höher Priorität als die Fehlermeldungen, wegen deren den Studierende überhaupt ablehnen können, Board zurückzugeben. Das gleiche gilt für den Anwendungsfall "*Rückgabe des Home-Loan Boards*".

3.4 Systemarchitektur

Die Architektur besteht aus den grundlegenden Beschreibungen (Ansichten), die zeigen, aus welchen grundlegenden Teilen (Komponenten, Modulen, Layouts usw.) das System besteht und wie diese Beschreibungen zusammenhängen.

3.4.1 Komponentendiagramm

UML-Komponentendiagramme stellen die Beziehungen zwischen einzelnen Systemkomponenten in einer statischen Entwurfssicht dar. Dabei können sowohl logische als auch physische Modellierungsaspekte berücksichtigt werden. Im UML-Kontext sind Komponenten modulare Teile eines Systems, die unabhängig sind und durch äquivalente Komponenten ausgetauscht werden können. Sie sind in sichgeschlossen und kapseln beliebig komplexe Strukturen. Kontakt zu anderen Komponenten nehmen die gekapselten Elemente ausschließlich über Schnittstellen auf [ION]. Die Komponentendiagramm eines zu entwickelten System ist auf der Abbildung 8 gezeichnet. Die Komponenten werden miteinander über HTTP-Protokoll kommunizieren. Es wird mithilfe API-Endpunkt geschehen, an dem eine Verbindung

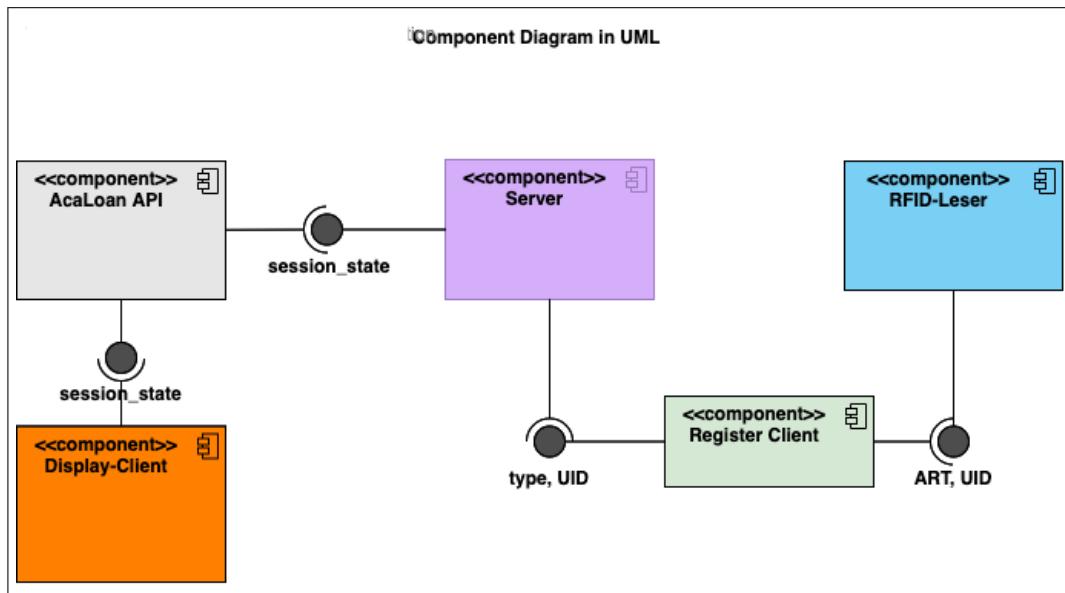


Abb. 8: UML Komponentendiagramm

mit den drei Bestandteilen der Softwareprogramm herstellt wird. An den API-Endpunkt werden Informationsanforderungen von einer Webanwendung einem Webserver geschickt und die Antwort empfangen. Die Aufgabestellung lässt sich die folgenden Struktur definieren, indem drei verteilte Teile zu entwickeln ist:

Register-Client wird als erste Komponente auf einem Raspberry Pi 3 Model B Rev 1.2 und RFID-Leser realisiert. Mit Hilfe des Register-Clients können der Studierende den genommenen Board auf sich registrieren lassen. Der Register-Client hat selbst keinen Zugriff zur Datenbank und sollte nur die abgelesene Daten von der Smartcard der Studierende zur Server schicken.

Display-Client ist zweite Komponente, der Benutzerinteraktion mit HTML-Seitenelementen, d. H. Klicken auf ein beliebiges Element oder eine Schaltfläche, erlaubt. Interaktive Webseite wird einem Benutzer im Vollbildmodus geöffneten Webbrower angezeigt.

Server sollte die Verwaltung der Datenbank erledigen und wird mit Django Webframework implementiert. Der Server wird mit einer Verbindung zur Datenbank, CRUD-Aufgaben (create, read, update, delete), Benutzerverwaltung beschäftigt und mittels Django stellt gute Sicherheit (CSRF, Kennwort-Hashing usw.) und RESTful-API bereit.

REST API wird zwischen Server und Klient für RESTful Kommunikation verantwortlich, die eine Implementierung eines Webdienstes unter Verwendung von HTTP- und REST-Prinzipien ist. Client-Server Architektur bedeutet im

Wesentlichen, dass sich Clientanwendung und Serveranwendung getrennt voneinander entwickeln müssen, ohne voneinander abhängig zu sein. Ein Client sollte nur Ressourcen-URIs kennen, und das ist alles.

Wie erfolgt nun die Abwicklung des eigentlichen Leihvorgangs von der Ausleihe bis zur Rückgabe eines Boards? Zuerst wird eine Studentenkarte am Register-Client ablesen und nachdem sollte einen Name und die Anzahl schon ausgeliehenen Boards am Display-Client angezeigt werden. Falls der Studierende zum Kurs zugelassen ist, darf dann ein gewünschten Board am Register-Client abgelesen werden. Es ist möglich, dass zu den schon ausgeliehenen Lab-Board noch zusätzlich einen Home-Board nach Hause mitgenommen wird. Das abgelesene Board ist entweder auszuleihen oder zurückzugeben. Sämtliche im Verleih befindlichen Geräte werden von den Mitarbeitern des Labors regelmäßig nach jeder Übung vor dem nächsten Ausleihevorgang auf Funktionsfähigkeit geprüft. Es kann sein, dass einem Studierenden die Home-Loan-Absicht eines Boards (12-16) verweigert wird, da in der Vergangenheit schon einmal vom Studierende ein Board in einem inakzeptabel Zustand zurückgeben war und die Ursachen mit den Mitarbeiter des Labor nicht geklären hat. Falls der Studierende, dem Home-Loan verboten wurde, ein Board auszuleihen versucht, wird eine Fehlermeldung auf dem Bildschirm gezeigt und der Leihvorgang mit dem Fehlerzustand terminiert.

3.5 aca-Loan Server

Der Server empfängt die Clientanforderungen und enthält die vom Client gewünschten Ressourcen. Der Server verfügt über eine API zur Interaktion mit Client, ohne ihnen direkten Zugriff auf in seiner Datenbank gespeicherte Inhalte zu gewähren.

3.5.1 Klassendiagramm

Für das Systemarchitektur wird das UML-Klassendiagramm entwickelt, das einen Überblick über ein Softwaresystem bietet, indem Klassen, Attribute, Operationen und deren Beziehungen angezeigt werden [Edi]. Auf der Abbildung 9 sind die geplante Klassen des Bestandteile "Server" gezeichnet, da diese Klassen dienen wie ein Entwurf für die zukünftigen Datenbank. In der Entwurfsphase wird es festgelegt, welche Klassen das System benötigt wird. Die festgestellte Klassen werden weiter nicht wie üblichen Python-Klassen implementieren, jedoch wie eine Django Modelle direkt zum Erzeugen der Datenbank verwendet. Das Klassendiagramm erklärt, was sind die Komponenten des Systems und wo sollen wir die Einkapselungsbarrieren platzieren. Welche Entscheidungen sind innerhalb von Komponenten zu verbergen,

damit sie geändert werden können, ohne den Rest des Systems zu beeinträchtigen. Die geplante Klassen sind auf der Abbildung 9 gezeichnet.

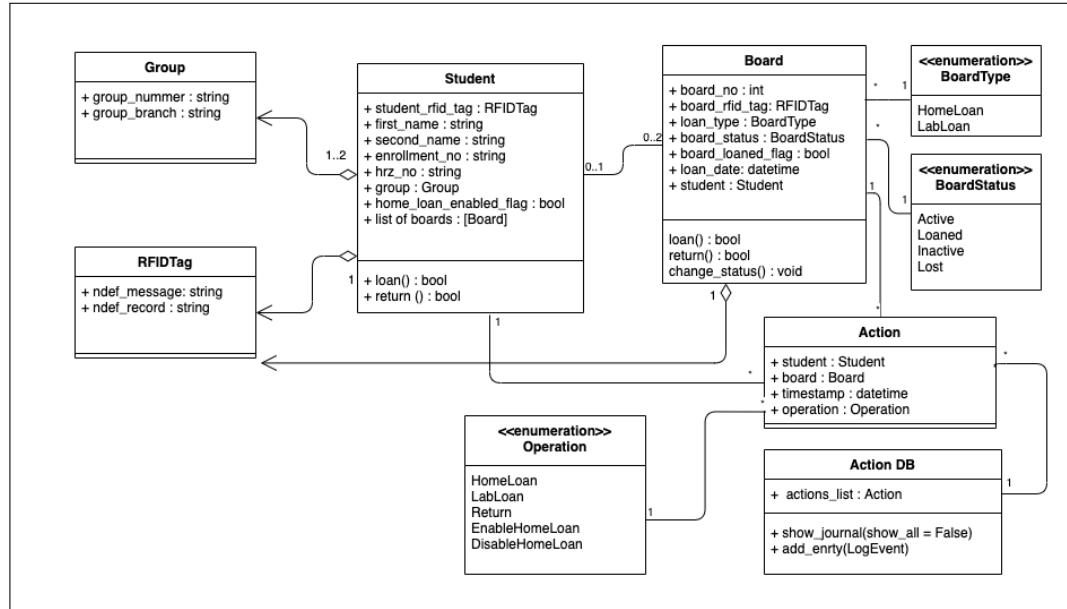


Abb. 9: UML Klassendiagramm des Servers

3.5.2 Datenbank

Die Datenbank wird mit einer SQLite Datenbank implementiert, die wird von Django unterstützt. SQLite ist eine Softwarebibliothek, die ein relationales Datenbankverwaltungssystem bereitstellt. Das "Lite" in SQLite bedeutet ein geringes Gewicht in Bezug auf Einrichtung, Datenbankverwaltung und erforderliche Ressourcen.

Entity Relationship Diagram

Das Generierte ERD (Entity Relationship Diagram, Abbildung 10) dient zum verstehen, wie die verwendete Datenbank aussieht. Durch Definieren der Entitäten, ihrer Attribute und Anzeigen der Beziehungen zwischen ihnen veranschaulicht zeigt ein ER-Diagramm die logische Struktur von Datenbanken. ER-Diagramme werden verwendet, um den Entwurf Django Datenbank zu skizzieren und später zu implementieren, wie es im Kapitel 4.2.2 beschrieben ist.

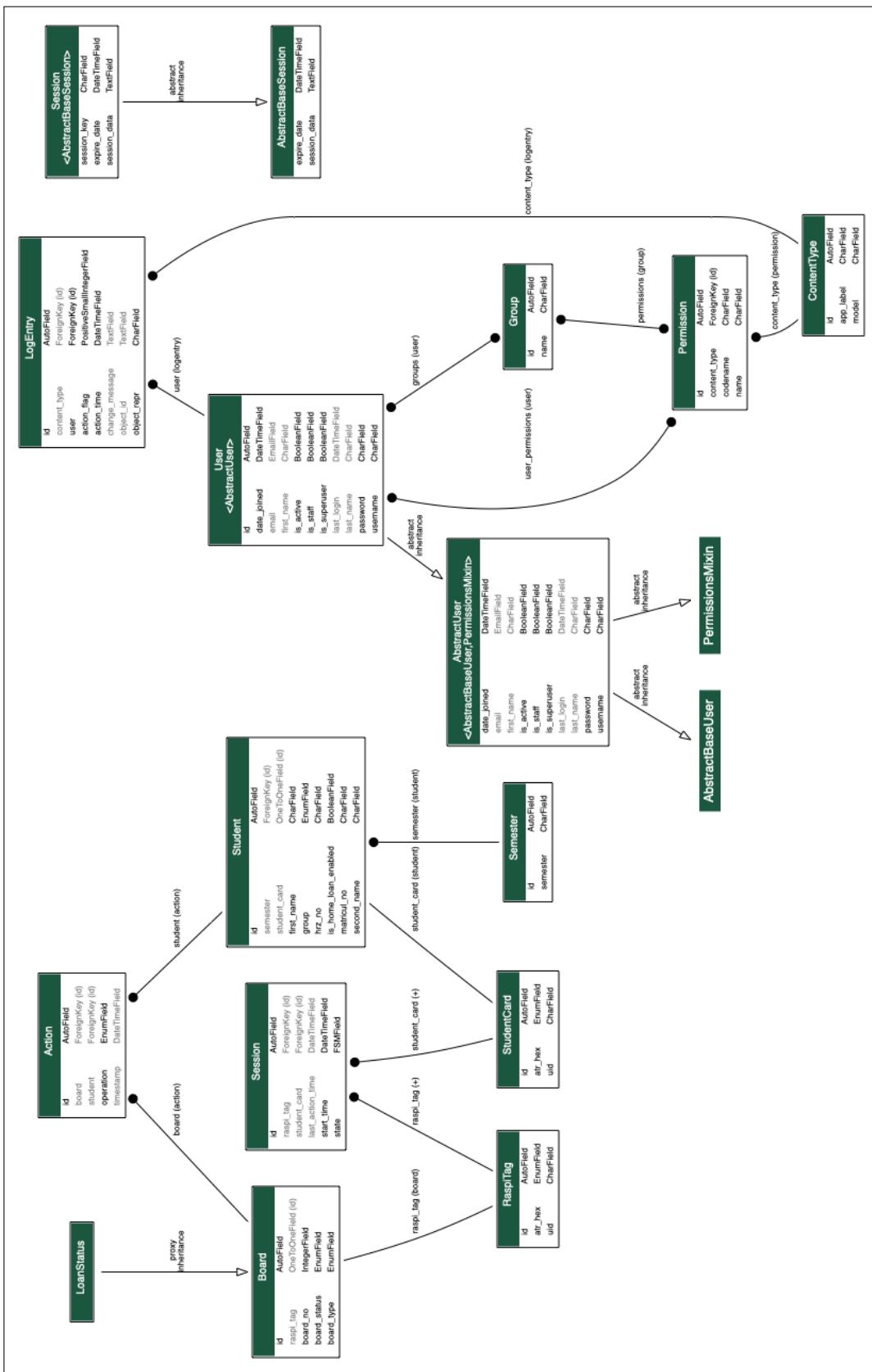


Abb. 10: Entity Relationship Diagram der DB

3.5.3 Endliche Zustandsmaschine

Eine Endliche Zustandsmaschine (oder einfach FSM - Finite-State-Maschine) wird für den Sitzungszustand verwendet, da immer nur ein Zustand aktiv sein kann. Sie ist ein Herz einer RESTful Kommunikation zwischen dem Server und dem Display-Client. Um eine Aktion auszuführen, muss die Maschine daher ihren Status ändern. Zustandsmaschine für die Realisierung der Abschlussarbeit wird verwendet, um den Ausführungsfluss zu organisieren und darzustellen. Eine Sitzung wird mit dem Startzustand "session started" angefangen, mit der erfolgreichen Transition "student card inserted" in den Zustand "valid student card" gewechselt. In diesem Zustand wird den Studierende seinen Name und Vorname auf Bildschirm der Display-Client angezeigt. Es macht keinen Sinn alle Zustände schriftlich zu beschreiben, da diese auf der Abbildung 11 abgebildet sind. Die Implementierung der endlichen Zustandsmaschine erfolgt mit django-fsm, die eine einfache deklarative Statusverwaltung für Django-Modelle hinzufügt.

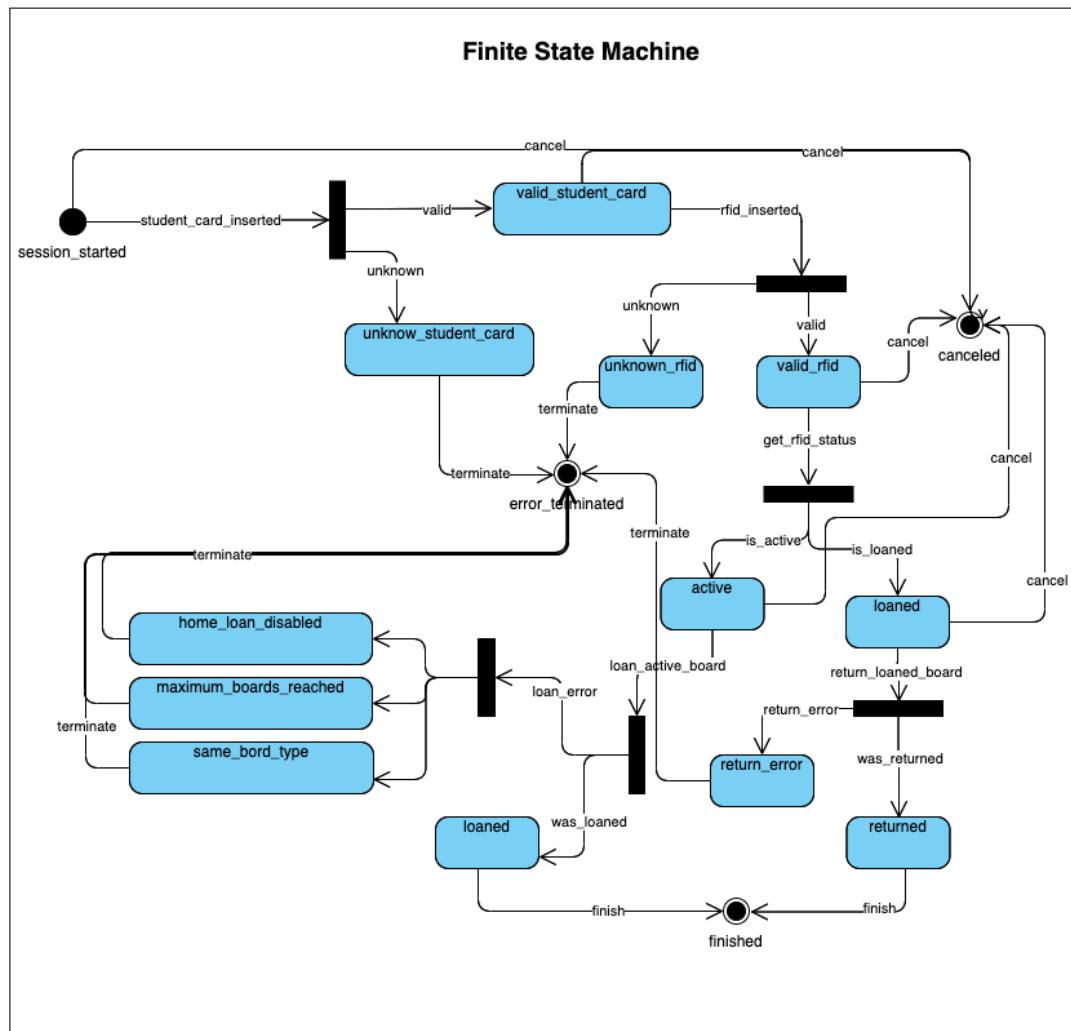


Abb. 11: Endliche Zustandsmaschine für RESTful Kommunikation

Implementierung

In meiner Abschlussarbeit präsentiere ich die praktische Lösung für das PSE-Labor der Beuth Hochschule für Technik Berlin. Die gesamte Aufgabe lässt sich in drei Bestandteile unterteilen: Register-Client, Server und Display-Client. Erstens wird Register-Client implementiert, damit wird RFID Leser am Raspberry Pi Mikrocomputer angeschlossen, alle Treiber installiert und auf Python Programmierung Sprache die Software geschrieben, die die ständige Überwachung des empfangenden von RFID Leser Daten zulässt und die Verbindung mit dem Server zulässt. Falls die empfangene Daten korrekt sind, d.h. eine richtige MIFARE Studentenkarte oder einen richtigen RFID-Transponder abgelesen wurde, schickt die Software die abgelesene Daten zum Server ab. Der Server ist der zweite Bestandteil der Abschlussarbeit und wird mit Hilfe Django Framework, Django Finite State Machine auf Python Programming Sprache implementiert. Server enthält die Datenbank mit die Datensätzen über die alle im PSE-Labor vorhandenen ausleihenden Boards, die zum Modul im laufenden Semester registrierten Studenten und geschehenen Ausleihe/Rückgabe-Vorgänge. Es wird von Server überprüft, ob eine von Register-Client abgelesene Studentenkarte einem zugelassenen für die Ausleihe Student gehört und die entsprechenden Information auf Display-Client geschickt. Es wird auch von Server bestätigt, ob für die Ausleihe/Rückgabe neben dem RFID-Leser gehaltenen Raspi Board dem Student ausgeliehen/vom Student zurückgegeben werden darf. Darauf aufbauend, wird der dritte Teil namens ein Display-Client als dynamische HTML-Seite realisiert, die eine Verbindung zum Server Mithilfe des HTTP-Protokolls und eingebauten im Browser Kommunikationsmittel die asynchrone Nachrichten zu schicken, bereitstellt. Für die dynamische Aktualisierung des Inhalts der Webseite und einen Zugang zum asynchronen HTTP-Client wird jQuery benutzt.

4.1 Register-Client

Das folgende Kapitel beschäftigt sich mit der Implementierung des Register-Client auf Raspberry Pi Board mit angeschlossenen RFID-Leser. Dieser Teil der verteilte System lässt sich wie folgendes unterteilen. Zuerst wurde das Betriebssystem Raspbian auf Board zum Leben gebracht und dann die alle notwendigen für RFID-Leser Treiber installiert. Nach dem der RFID-Leser funktionieren angefangen und die Daten von

RFID-Transponder abgelesen hat, wurde die nächste Herausforderungen gelöst: die Struktur die zu empfangenen Daten wurde verstanden, richtig bearbeitet, eine JSON-Datei erstellt und durch die HTTP-Protokoll dem Server geliefert.

4.1.1 Installation des Betriebssystem

Der vorhandene für die Abschlussarbeit Raspberry Pi 3 Model B+ wurde nicht als Starter Kit mir übergeben, dann wurde es zusätzlich benötigt [Hal19, pp. 21-22]:

- **USB-Netzteil** mit einer Nennleistung von 2,5 A (2,5 A) oder 12,5 Watt (12,5 W) und einem Micro-USB-Anschluss.
- **microSD-Karte**, die als permanenter Speicher des Raspberry Pi dient; Alle von Benutzer erstellten Dateien und die installierte Software sowie das Betriebssystem selbst werden auf der microSD-Karte gespeichert.
- **USB-Tastatur und -Maus**, mit denen den Raspberry Pi gesteuert werden kann. Fast jede kabelgebundene oder kabellose Tastatur und Maus mit USB-Anschluss funktioniert mit dem Raspberry Pi.
- **Das HDMI-Kabel**, das Ton und Bilder vom Raspberry Pi auf Fernseher oder Monitor überträgt. Sie müssen nicht viel Geld für ein HDMI-Kabel ausgeben.

Die Arbeit mit einem RaspberryPi setzt ein paar Anfangsinvestitionen voraus, die auch von den angestrebten Aufgaben und Projekten abhängen. Zuerst gäbe es die Möglichkeiten, dass der gekaufte Raspberry Pi Board bereits ein Betriebssystem darauf installiert hätte. Aber es war nicht der Fall von vorhandenen im PSE-Labor Board. Um ein Betriebssystem auf diesen Raspberry Pi zu bringen, muss eine SD-Karte mit einem Betriebssystem-Image "geflasht" werden. Dafür zunächst wurde die Distribution von der Website Raspbian.org herunterladen und die MicroSD-Karte in den Kartenleser eines vorhandenen im PSE-Labor PC eingelegt. Anschließend wurde mit dem Macintosh Disk Utility-Dienstprogramm das heruntergeladene und entpackte Betriebssystem für den RaspberryPi auf eine Speicherkarte geschrieben. Dann ist die Karte in Raspberry Pi einzulegen. Der Raspi ist damit betriebsbereit und muss für die zukünftigen Anwendungen noch konfiguriert werden. Wenn der Pi zum ersten Mal eingeschaltet wird, wird viel Text auf dem Bildschirm angezeigt. Diese werden als Startmeldungen bezeichnet. Wenn Raspbian zum ersten Mal gestartet wurde, kann es ein oder zwei Minuten dauern, um die Nutzung des freien Speicherplatzes auf der microSD-Karte optimal anzupassen. Beim nächsten Start geht es schneller. Schließlich ist kurz ein Fenster mit dem Raspberry Pi-Logo zu sehen, dann wird Terminal Fenster angezeigt, in dem es einloggt werden muss. Zum

ersten Einloggen wird den Standardbenutzernamen "pi" und das Standardkennwort "raspberry" verwendet. Um Register-Client vor sowohl Online-Bedrohungen als auch von Missbrauch im Labor zu schützen, wurde das Standardkennwort sofort geändert. Der nächste Schritt ist Raspbian bis zur Version "Raspbian mit dem Raspberry Pi Desktop" zu aktualisieren, damit die grafische Benutzeroberfläche und Chromium Browser zu Verfügung stehen können. Dies kann mit dem Terminalbefehl gemacht werden:

```
sudo apt-get install lxde-core xserver-xorg xinit
```

Dann ist der Raspberry Pi erneut zu laden. Nachdem Raspberry Pi-Logo wieder angezeigt wurde, wäre der Raspbian-Desktop zu sehen. Somit gilt Betriebssystem als vollständig installiert und kann benutzt werden. Das war aber nicht der Fall mit dem vorhandenen Hardware, da es plötzlich eine Boot-Schleife vorkam, nachdem der Mikrocomputer eingeschaltet wurde und der Startvorgang nicht abgeschlossen werden konnte. Anstatt das zum Benutzung bereiteten Betriebssystem mit der grafische Benutzeroberfläche zu sehen, wird eine Schleife erzeugt, in der die Startvorgang kontinuierlich und wiederholt ausgeführt wurde und somit eine Nutzung der Mikrocomputers unmöglich ist. Nach den mehreren Recherchen wurde es vermutet, dass es durch eine unzureichende Stromversorgung verursacht werden könnte. Es wurde aber zuerst nicht versucht, einen USB-Netzteil zu wechseln, da die anderen USB-Netzteil man durch PSE-Labor bestellen und eine Zeit abwarten muss. Jedoch wurde eine erzeugte Boot-Schleife mit einem anderen Terminalbefehl erfolgreich gelöst:

```
sudo apt-get install --reinstall pcmanfm
```

Bei der Arbeit mit dem Mikrocomputer tritt jedoch später ein Problem mit der Stromversorgung auf. Der Fall kann im entsprechenden Kapitel 4.1.3 nachgelesen werden.

4.1.2 Installation der Treibers für RFID-Leser

Im weiteren Verlauf der Arbeit wird den RFID Leser/Schreiber ACR122U erfolgreich angefahren, der auf der Basis der 13,56 MHz kontaktlosen (RFID) Technologie entwickelt wurde. Der ACR122U USB Kartenleser unterstützt nicht nur Mifare ® Technologien, sondern auch alle vier Typen von NFC -Tags. Diese Anforderung ist für vorliegende Abschlussarbeit wichtig, da Studierenden-Ausweise der Beuth Hochschule mit dieser Technologie gelesen werden können. Obwohl Raspbian mit einer Reihe von Software vorinstalliert ist, wird es aber zusätzlich benötigt, die Treiber für RFID-Leser zu installieren. Es sollte an dieser Stelle auch noch angemerkt werden, dass am Anfang der Entwicklungsprozess ein anderen RFID-Leser angeschlossen wurde als der, den in der Abschlussarbeit zu beschreiben und zu

beobachten ist. Zuerst wurde die Treiber für Reiner SCT CyberJack RFID Basis [SCT] installiert. Die Schritte sollten in der Abschlussarbeit nicht unerwähnt bleiben, da die damals für den ersten RFID-Leser installierte Treiber und Daemons (Appendix 6) wurden endlich für den zweiten RFID-Leser benutzt, mit dem die Entwicklung der Aufgabe abgeschlossen wurde. Der Grund für die Hardwareaustausch ist die festgestellte Tatsache, dass Reiner SCT CyberJack RFID Leser die Studentenkarten nicht ablesen konnte. Obwohl in der Spezifikation es steht, dass die Reiner RFID-Leser kontaktlose RFID Chipkarten wie eID mit dem neuen Personalausweis (nPA), Geld-Karte oder eTicketing unterstützt, wurde es unmöglich mit den MIFARE-Transponder 13,561 MHz Funkbereich ins Spiel zu bringen. Die Studierenden-Ausweise der Beuth sollten mit dieser Technologie gelesen werden und somit ist diese Anforderung für die vorliegende Abschlussarbeit wichtig.

Um RFID-Leser anzuschließen, muss man den Leser über USB mit einem Raspberry Pi verbunden. Der Typ des RFID-Lesegeräts, der für die vorliegende Abschlussarbeit verwendet wird, ist ein ACR122U-A9 von Advanced Card Systems[Ltd], den auf der Abbildung 12 zu sehen. Zuerst müssen wir die Paketlisten aktualisieren und einige Pakete herunterladen und installieren [OG]: Die folgenden Abhängigkeiten werden benötigt im System mit dem Befehl "sudo apt-get install" zu installieren: *libusb-dev, libpcsc-lite-dev, libpcsc-lite1, libccid, pcscd, pcsc-tools, libpcsc-perl, libusb-1.0-0-dev, libtool, libssl-dev*.



Abb. 12: ACR122U-A9 von Advanced Card Systems und Raspberry Pi

PC/SC ist ein Standard für die Schnittstelle von Computern mit Smartcards, der auf den meisten Betriebssystemen, einschließlich Windows und Linux, verfügbar ist. PC/SC-Kopplungsgeräte benötigen einen Treiber, mit dem Anwendungen die Karte einfach erreichen können. Da PC/SC für Smartcards entwickelt wurde - und in einer Zeit, in der Smartcards nur Kontaktkarten waren funktioniert es auch mit den kontaktlosen Karten, falls RFID-Leser es unterstützt. Das Daemon-Programm für pcsc-lite namens pcscd koordiniert die Kommunikation mit Smartcard-Lesegeräten und Smartcards sowie kryptografischen Tokens, die mit dem System verbunden sind. Normalerweise wird pcscd beim Booten von /etc/init.d/pcscd gestartet.

Damit können Anwendungen auf Smartcards und Lesegeräte zugreifen, ohne die Details der Karte oder des Lesegeräts zu kennen. Das Laden von Treibern für Kartenleser wird von pcscd koordiniert. Der Zweck von pcsc-lite besteht darin, eine kompatible

API (Winscard) für die Migration von Windows-basierten PC / SC-Anwendungen auf Unix bereitzustellen [CR]. Die allgemeinen Zugriff auf USB-Geräte bietet eine C-Bibliothek namens libusb. Sie soll von Entwicklern verwendet werden, um die Produktion von Anwendungen zu erleichtern, die mit USB-Hardware kommunizieren. Es ist portabel, da mit einer einzigen plattformübergreifenden API auf USB-Geräte unter Linux, MacOS, Windows usw. zugegriffen werden kann. Sie wird im Benutzermodus ausgeführt und somit für die Kommunikation der Anwendung mit einem Gerät keine besonderen Berechtigungen oder Erhöhungen erforderlich sind [lib].

Dann laden wir die Open-Source-Bibliothek libnfc für Near Field Communication (NFC) herunter, extrahieren, konfigurieren und installieren es. Nach der erfolgreichen Installation kann den verbindenden über USB RFID-Leser mithilfe des Tools lsusb angesehen werden: es wird VendorId und ProductID angezeigt. Es ist auch notwendig die abgelesene VendorId und ProductID in der entsprechenden XML-Datei. Die Datei auf der MicroSD-Karte ist zu finden :

```
/usr/lib/pcsc/drivers/ifd-ccid.bundle/Contents/info.plist
```

4.1.3 Spannungsproblem und Lösung

Nachdem der RFID-Leser angeschlossen wurde, tritt es ein weiteres Problem, das die freie Benutzung der Raspberry Pi unmöglich macht. Es wird unabhängig von der Zeit und vorherigen Geschehen eine Fehlermeldung "under voltage detected (0x000050000000)". Es wird zuerst versucht die kabellose USB-Tastatur und -Maus abzuschalten. USB eine aktive Abfrage seine Ports (Polling) benötigt, was bedeutet, dass die Anzahl der für andere Aufgaben verfügbaren CPU Zyklen geringer ist. Dies kann dazu führen, dass die CPU-Frequenz ansteigt, wodurch der Stromverbrauch höher wäre. Es wurde festgestellt, dass mit unverbundenen USB-Tastatur und -Maus die Fehlermeldung trotz vorkommt. Nur wenn der RFID-Leser angeschaltet wurde, kam es keine neue Fehlermeldung. Dann es wurde geprüft, ob ausgewählten RFID-Leser mit dem vorhandenen Mikrocomputer überhaupt kompatibel ist und nach der Lösung gesucht, mit der die zukünftige Entwicklung weiterlaufen kann.

Anfänglich wurde als Spannungsversorgungsteil ein USB-Netzteil von einem modernem bei Autorin der Abschlussarbeit vorhandenen zu Hause Smartphone benutzt. Das offizielle Raspberry Pi-Netzteil ist die empfohlene in der Dokumentation Wahl, jedoch wurde zuerst mit dem Mikrocomputer nicht gekauft. Ein leistungsfähiger USB-Netzteil könnte den schnell wechselnden Strombedarf des Raspberry Pi bewältigen. Nach der Besprechung des Problems mit den PSE-Labor Mitarbeiter wurde ein Samsung USB-Netzteil gegen einen Anker USB-Netzteil ausgetauscht und festgestellt, dass das Spannungsproblem mit den verbundenen sowohl USB-Tastatur und -Maus als auch RFID-Leser während des Ablesevorgangs nicht wieder erscheint.

4.1.4 Python Programmierung des RFID-Lesers

Nachdem es gelingt mir, die Hardware angefahren und die ersten Studentenkarte so abzulesen, dass es ein Schallton von RFID-Leser erzeugt wurde, sollte eine weitere Aufgabe gelöst werden: die Daten von RFID-Tag auf einem auszuleihenden Raspi Board von Studentenkarte unterscheiden zu können. Es sollte ausgeschlossen werden, dass ein Ausleihe-/Rückgabevorgang angefangen wird, falls eine falsche Studentenkarte (z.B. mit einer BVG Jahresfahrkarte) am RFID-Leser präsentiert wird. Als es im Kapitel 2.2 erklärt wurde, sind die neuen Campus Karte der Beuth Hochschule für Technik Berlin mit den die MIFARE-Transponder hergestellt. Für die Programmierung des RFID-Lesers wird Smartcard-Schnittstelle benutzt, deren Installierung geschah zusammen mit pyscard und im Kapitel 4.1.2 geschrieben. Diese Schnittstelle steht für den Entwickler für die Arbeit mit Smartcards und NFC-Geräten zur Verfügung, sie wird in Form mehrerer Systemdienste implementiert und ihr Schnittstellenteil ist das PC/SC-Framework. PC/SC steht für Personal Computer/Smart Card.

Es steht mehreren Möglichkeiten für die Entwicklung eine Sprache zu wählen: die Funktionen der Smartcard-Schnittstelle können mit Python, C/C++ und Java Sprache verwendet werden. Die Programmierung des Register-Clients wird auf Python Sprache gemacht, dieselbe Sprache wird für Server während der Arbeit mit Django Framework benutzt. Die Entwicklung mit dem Importieren der PCSC-Header-Dateien beginnt.

```
from smartcard.System import readers
from smartcard.ATR import ATR
from smartcard.CardMonitoring import CardMonitor, CardObserver
from smartcard.CardRequest import CardRequest
```

Wann eine RFID-Leser über USB angeschlossen wird, kann es eine Verbindung zur PC/ SC-Bibliothek hergestellt und eine Liste der verfügbaren Terminals abgerufen werden. Alle API-Funktionen geben einen Statuscode zurück. Wenn die Funktion erfolgreich ist, wird die Konstante *SCARD_S_SUCCESS* zurückgegeben. Alle anderen Daten werden über Funktionsargumente zurückgegeben, in denen die Adresse der gewünschten Variablen übergeben wird. Die Verbindung (Initialisierung) erfolgt über die Funktion *SCardEstablishContext()* [Chi14, p. 101]. Die Adresse der Variablen *sc_context* vom Typ *SCARDCONTEXT* wird an sie übergeben. Als Nächstes müssen Sie eine Liste der Terminals abrufen. Dies erfolgt über die Funktion *SCardListReaders()*. Dann muss die Liste der angeschlossenen Lesers erhalten und gelesen. Es ist eine Reihe von Zeichenfolgen, die durch ein Nullbyte getrennt sind. Dies ist ein Windows-Format zur Darstellung von Zeichenfolgenlisten und wird üblicherweise als Zeichenfolge mit doppelter Nullterminierung bezeichnet. Durch Aufrufen der Funktion *SCardListReaders()* erhalten wir eine Liste aller Namen

[Chi14, p. 102]. Da in Abschlussarbeit es ist vorgesehen, dass nur ein RFID-Leser über USB am Register-Client angeschlossen werden darf, wird nur ein Name nach der Funktionsausruf zurückgegeben:

```
Found readers: ['ACS ACR122U']
```

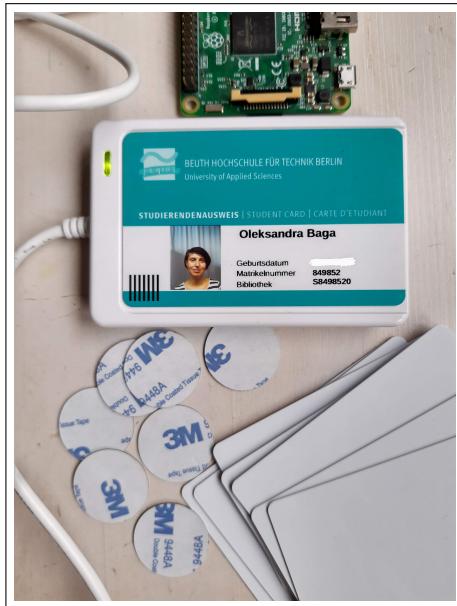


Abb. 13: RFID-Leser mit der Studentenkarte der Autorin und Tags

nikationsprotokoll gespeichert wird. Auf Anwendungsebene interessiert es uns nicht besonders, welches Protokoll zum Herstellen der Verbindung ausgewählt wurde, aber diese Informationen müssen während der Programmierung gespeichert werden - sie werden später zum Senden von Befehlen benötigt [Chi14, p. 104].

```
HRESULT hresult, hcontext = SCardEstablishContext(SCARD_SCOPE_USER)
if (hresult == SCARD_S_SUCCESS):
    hresult, readers = SCardListReaders(hcontext, [])
    if len(readers) > 0:
        reader = readers[0]
        hresult, hcard, dwActiveProtocol =
            SCardConnect(hcontext, reader,
                        SCARD_SHARE_SHARED, SCARD_PROTOCOL_T0 |
                        SCARD_PROTOCOL_T1)
```

Listing 4.5: Funktion SCardEstablishContext

ATR (Answer-To-Reset) ist ein kurzes (nicht mehr als 33) Byte-Array, das die Karte beim Anschließen an das Terminal senden muss. Wenn die Karte dies nicht innerhalb einer bestimmten Zeit tut, wird davon ausgegangen, dass sie nicht richtig funktioniert. ATR enthält grundlegende Informationen über die Karte und die technischen

Wie die weitere Entwicklung darf einfach den ersten gefundenen RFID-Leser genommen werden. Wenn keine Terminals angeschlossen sind, wird das Programm beendet und eine entsprechende Meldung der PSE-Labor Mitarbeiter ausgegeben. Dann muss mit dem ausgewählten Terminal verbunden werden. Dies erfolgt durch Aufrufen der Funktion *SCardConnect()*. Beim Aufruf von *SCardConnect()* geben wir den gemeinsam genutzten Modus (*SHARE_SHARED*), das bevorzugte Kommunikationsprotokoll (in unserem Fall T0 und T1) an, übergeben die Adresse der Variablen, in die das Handle für weitere Operationen mit der Karte gespeichert wird, und die Adresse der Variablen *active_protocol*, in der gewählten Kommunikationsprotokoll gespeichert wird.

Parameter der Verbindung. Das Format ist jedoch recht kompliziert und hängt vom Hersteller des Kartenchips, den darauf installierten „Anwendungen“ usw. ab. ATR wird im Terminal gespeichert, solange die Karte angeschlossen ist. Es lässt sich die zwei verschiedenen RFID-Transponder (MIFARE Smart-Studentenkarte und RFID-Tag) mit Hilfe ATR voneinander unterscheiden. Das folgendes wird z.B. von der Studentenkarte zurückbekommen, wann Autorin ihre eigene Studentenkarte am RFID-Leser ablesen lässt (siehe Abbildung 13):

```
+Inserted: 3B 81 80 01 80 80
Student card added
uid 04 2F 75 7A 30 40 80
```

Listing 4.6: Abgelesene Studentenkarte

Das anderes wird aber angezeigt, wann den RFID-Transponder am RFID-Leser präsentiert wird, den in der Zukunft am im PSE-Labor vorhandenen Raspi-Boards angeklebt wird:

```
+Inserted: 3B8F8001804FOCA00000030603000100000006A
Raspi board added
uid 5C E7 87 30
```

Listing 4.7: Abgelesener Raspi-Tag

Klasse *DetectionObserver* mit der Vererbung vom Klasse *CardObserver* wird benutzt, um zu erkennen, wann eine Karte dem Kartenleser vorgelegt wurde, und dann die eindeutige Kennung (UID) von einer Karte zu lesen. Es wird mithilfe der Klasse *CardMonitor* getan, die das Einsetzen / Entfernen von Smartcards überwacht und den *CardObserver* benachrichtigt.

```
def update(self, observable, actions):
    (addedcards, removedcards) = actions
    for card in addedcards:
        atr = toHexString(card.atr)
        added_card = self.get_cardtype(toHexString(card.atr),
                                       "added")
        self.read_uid(added_card)
```

Listing 4.8: Monitor der angelegte und entfernte Smartkarten

Sobald eine korrekte Studentenkarte oder RFID-Transponder am RFID-Leser erscheint und ohne Kollision mit anderen in der Nähe bleibenden elektromagnetische Felde abgelesen wird, wird abhängig vom dem Typ der Karte eine JSON-Datei erstellt und von einem acaLoan-client zum Server geschickt. Der acaLoan-client selbst ist ein kleinen Python-Script, der die korrekte Erstellung der JSON-Datei erlaubt und eine Verbindung mittels HTTP-Protokoll zum Server bedient.

```
class AcaLoanClient:
```

```

    def __init__(self, base_url):
        self.base_url = base_url

    def send_event(self, type, uid):
        endpoint = "{}/loan/api/events".format(self.base_url)
        payload = {"type": type, "uid": uid}
        r = requests.post(endpoint, json=payload)

```

Listing 4.9: acaLoanClient

Es wird weiteres von Register-Client auf keine Antwort vom Server erwartet, ob der abgelesenen Studentenkarte die Ausleihe des Raspi-Boards erlaubt ist oder ob gehaltenen in der Hand Raspi-Board ausgeliehen oder zurückgegeben darf. Sobald die Verbindung zum Server erfolgreich hergestellt wurde und eine JSON-Datei geschickt, wird der RFID-Leser wieder zum Ablesen der nächsten RFID-Transponder freigegeben. Der Python-Script für RFID-Leser muss am Register-Client mit der Verwendung der Umgebungsvariable *LOANSERVER*. Während der Entwicklung der Abschlussarbeit geschieht es mit dem Name des Django Entwicklungsservers. Mehr darüber ist in folgenden Kapitel 4.2 nachzulesen.

```
LOAN_SERVER_URL=http://127.0.0.1:8000 python reader.py
```

Als letztes für die Implementierung des Register-Clients ist es wichtig, die kurze Beschreibung der entwickelten von Autorin JSON-Datei anzugeben. Definition der Struktur, des Inhalts und der Semantik von JSON-Objekten geschieht mithilfe von der Grammatiksprache namens JSON-Schema. Hier kann Metadaten (Daten über die Daten) angegeben werden, die die Eigenschaften eines Objekts beschreiben und gültige Werte.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "array",
  "items": [
    {
      "type": "object",
      "properties": {
        "type": {"type": "string",
                  "enum": ["card", "uid", "cancel_button", "terminate_button", "get_rfid_status", "return_scanned_board_button", "loan_scanned_board_button", "finish_button"]},
        "uid": {"type": "string"}
      },
      "required": ["type", "uid"]
    }
  ]
}
```

Listing 4.11: JSON-Schema

Zu Zweck Definition der Struktur wird das Schlüsselwort "items" mit einem Array gesetzt, wobei jedes Element ein Schema ist, das jedem Index des Arrays des Dokuments entspricht. Das heißt, ein Array, bei dem das erste Element das erste Element des Eingabearrays validiert, das zweite Element das zweite Element des

Eingabearrays validiert usw [MD]. Es ist zu betonen, dass für das ersten Element der JSON-Datei es eine Zeichenfolge aus einem festen Wertesatz sein muss. Nur diesen Zeichenfolgen können von Endliche Zustandsmaschine, die im Kapitels 3.5.3 und ?? beschrieben sind, abgearbeitet werden.

4.2 Server

Als er wurde im Kapitel 4.1.4 erwähnt, während der Implementierung der Aufgabe der Abschlussarbeit wurde mit dem Entwicklungsserver gearbeitet. Ein Entwicklungsserver ist ein Servertyp, der die Entwicklung und das Testen von Programmen, Websites, Software oder Anwendungen für Softwareprogrammierer erleichtert. Der bietet eine Laufzeitumgebung sowie alle Hardware- / Software-Dienstprogramme, die für das Debuggen und die Entwicklung von Programmen unerlässlich sind. Django ist eines der effizientesten modernen Frameworks für die Entwicklung von Webprojekten. Der Grund für diese Effizienz ist ein klarer Mechanismus für die Arbeit mit einem Projekt, ein praktisches ORM (Object Relational Mapping Layer), mit der mit Anwendungsdaten aus verschiedenen relationalen Datenbanken wie SQLite, PostgreSQL und MySQL interagiert werden kann.

4.2.1 Erstellung acaLoan Django-Projekts

Glücklicherweise ist der Installationsprozess für Django unkompliziert, sodass das Einrichten Ihrer Entwicklungsumgebung schnell und entspannt ist. Django ist vollständig in Python geschrieben, daher muss zuerst Python installiert werden, um Django zu installieren. Weil die Autorin für die Implementierung des Servers ihren eigenen Mac OS Rechner verwendet, ist Python bereits Computer installiert. Wenn "Python" in die Befehlszeile eingegeben (mithilfe der Terminal.app auf meinem Mac), wurde folgendes angezeigt werden:

```
$ python
Python 3.8.1 (default, Feb 17 2020, 23:55:16)
[Clang 11.0.0 (clang-1100.0.33.17)] on darwin
```

Sobald Python auf Computer installiert ist, kann Django auch installiert werden. Es gibt drei Möglichkeiten: Installation der offizielle Django-Version, Verwendung eines verteilungsspezifisches Installationsprogramm oder Herunterladen der Version, die derzeit immer noch entwickelt wird. In der Abschlussarbeit wird nur die Installation der offiziellen Version verwendet. Der Installation wird mit "Pipenv"-Tool geschieht, das isolierte Python-Umgebungen bietet, die somit praktischer sind als die systemweite Installation von Paketen. Pipenv verwaltet Projektpakete automatisch über die Pipfile-Datei, während die Pakete installiert oder deinstalliert werden. Pipenv

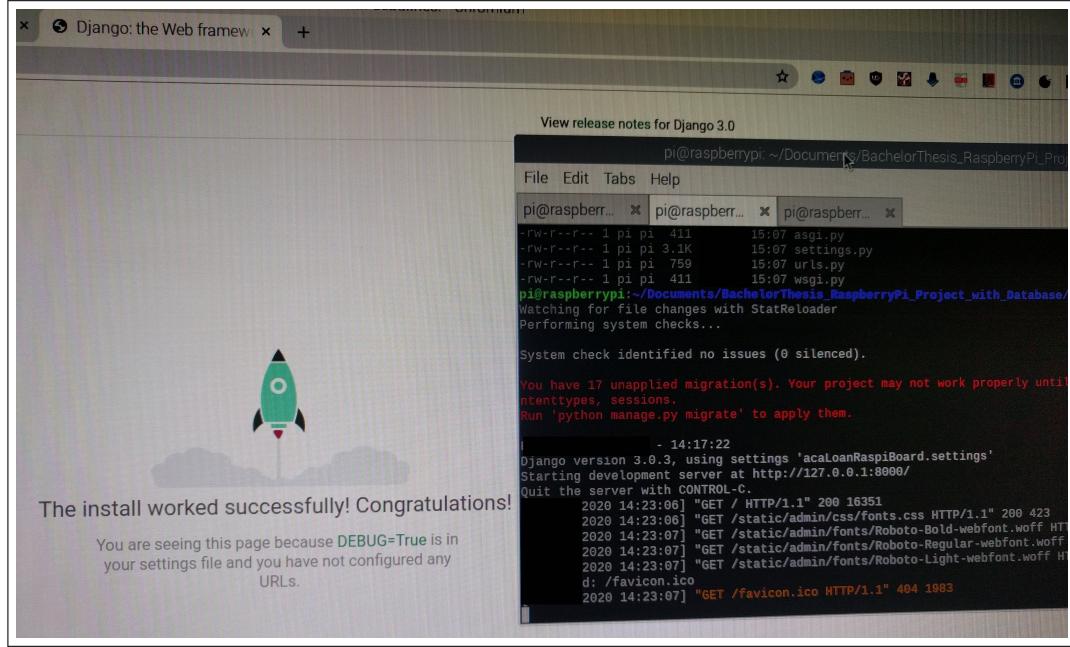


Abb. 14: Server nach Erstellung des acaLoanRaspiBoard-Projekts

generiert auch die Datei Pipfile.lock, mit der deterministische Builds erstellt und eine Momentaufnahme Ihrer Arbeitsumgebung erstellt werden. Zusätzlich für die Aufgabeimplementierung wird django-fsm installiert, das mit der Django Installation nicht geliefert wird und erlaubt eine einfache deklarative Zustandsverwaltung für Django-Modelle.

Der ganzen Quellcode für das acaLoan-Server wird als Django Projekt dargestellt. Django hat einen Befehl zum einfachen Erstellen einer anfänglichen Projektstruktur. Es wird mit der Ausführung des folgenden Befehls vom Terminal erledigt:

```
django-admin startproject acaLoanRaspiBoard
```

Die Datei settings.py enthält eine Grundkonfiguration für die Verwendung einer SQLite-Datenbank und eine Liste von Django-Anwendungen und wurde als während der Implementierung geändert, um das üblichste deutsche Zeitformat anstatt des amerikanischen zu verwenden.

```
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'Europe/Berlin'
TIME_INPUT_FORMATS = ['%H:%M:%S']
TIME_FORMAT = '%H:%M:%S'
DATETIME_FORMAT = "d.m.Y H:m"
DATE_FORMAT = "Y-m-d"
```

Listing 4.14: Deutsches Zeitformat in Django

acaLoan-Server muss im zusätzliche Dateien wie Bilder, JavaScript oder CSS bereitstellen. In Django werden diese Dateien als "statische Dateien" bezeichnet. Django stellt `django.contrib.staticfiles` zur Verfügung, um den Entwickler bei der Verwaltung zu unterstützen. Dafür müssen in die Datei `settings.py` die folgenden Zeilen hinzugefügt werden:

```
STATIC_URL = '/static/'  
STATICFILES_DIRS = [os.path.join(BASE_DIR, "staticfiles")]
```

In der `settings.py` ist auch den Debug-Modus des Projekts aktiviert, um während der Entwicklung detaillierte Fehlermeldungen zu sehen. Nach der Erstellung des Projekts und kleinen notwendigen Änderungen, kann endlich acaLoan-Server gestartet werden. Da Django einen praktischen Mechanismus bietet, um die Konfiguration Ihres Webservers während der Entwicklung zu vermeiden, kann enthaltenen bereits ein Webserver mit dem Befehl aufgerufen werden:

```
python manage.py runserver
```

Nach dem Befehl ausgeführt wurde und im Browser zu `http://127.0.0.1:8000/` navigiert wurde, wird die folgende Ausgabe angezeigt, die auf der Abbildung 14 zu sehen ist.

4.2.2 Datenbankeinrichtung

Nach dem Erstellung des Projekts und dem ersten Starten des acaLoan-Server ist es die Zeit, sich mit der Einrichtung von Datenbank zu beschäftigen. Theoretisch kann ein Django-basierte Projekt ohne Datenbank ausgeführt werden. Für die hier vorliegenden Abschlussarbeit was jedoch es ein Zweck, eine Datenbank einzurichten, um alle Ausleihe-/Rückgabevorgänge zu verwalten, so dass in einem echten Projekt kann man nicht auf Datenbankeinrichtung verzichten. Für die Implementierung wird für Verwendung von SQLite3 entschieden. Die Datenbank selbst wird in einer Datei auf der Festplatte gespeichert. Um die Datenbank mit dem Django-Projekt zu verbinden, muss den DATABASES-Block in der Datei `mysite/settings.py` bearbeitet. Um sqlite zu verwenden, muss nur `django.db.backends.sqlite3` als ENGINE angegeben und den Namen der Datei, in der die Datenbank gespeichert wird, in den Parameter NAME geschrieben werden [Fouc].

Als nächsten müssen die Modelle in Django erzeugt werden, die die Daten definieren, mit denen während der Entwicklung gearbeitet werden. Die Modelle in Django werden entsprechend des Klassendiagramm aus dem Kapitel 3.5.1 und des Entity Relationship Diagramm realisiert, das im Kapitel 3.5.2 gezeigt wurde. Die wichtigste Modelle sind unten zu sehen:

```

class Student(models.Model):
    student_card = models.OneToOneField(StudentCard, on_delete=
        models.SET_NULL, blank=True, null=True)
    semester = models.ForeignKey(Semester, on_delete=models.
        CASCADE, blank=True, null=True)
    first_name = models.CharField('first name', max_length=50)
    second_name = models.CharField('second name', max_length=50)
    matricul_no = models.CharField('matriculation', max_length
        =10, unique=True)
    hrz_no = models.CharField('hrz', max_length=10, unique=True)
    group = enum.EnumField(StudentGroup)
    is_home_loan_enabled = models.BooleanField(default=True)

```

Listing 4.17: Student Modell

ForeignKey wird verwendet, um eine Viele-zu-Eins-Beziehung zu definieren, die eine Beziehung zwischen mehr als einer Instanz einer Entität und einer Instanz einer anderen Entität ist. Sodass in der Datenbank für ein Semester können mehrere Studierende immatrikuliert werden. *on_delete = models.CASCADE* ist ein SQL-Standard, das nicht nur in Django verwendet wird. Wenn das referenzierte Objekt gelöscht wird, ist auch die Objekte zu löschen, auf die es verweist. Wenn es beispielsweise einen Semester zu entfernen ist, müssen auch die Studenten gelöscht, die während dieses Semester in System registriert wurden. Damit sollen alle veraltet Daten der Studierende nach dem Semesterende endgültig aus der Datenbank gelöscht werden.

```

class Board(models.Model):
    raspi_tag = models.OneToOneField(RaspiTag, on_delete=models.
        SET_NULL, blank=True, null=True)
    board_no = models.IntegerField('board number', unique=True)
    board_type = enum.EnumField(BoardType, default=BoardType.
        LAB_LOAN)
    board_status = enum.EnumField(BoardStatus, default=
        BoardStatus.ACTIVE)

```

Listing 4.18: Board Modell

Um eine Eins-zu-Eins-Beziehung zu definieren, wird OneToOneField verwendet. In einer relationalen Datenbank besteht eine Eins-zu-Eins-Beziehung, wenn eine Zeile in einer Tabelle nur mit einer Zeile in einer anderen Tabelle verknüpft sein kann und umgekehrt.

```

class Action(models.Model):
    # Holds the record about the loan operation, student, board
    # and time
    student = models.ForeignKey(Student, on_delete=models.
        SET_NULL, blank=True, null=True)
    board = models.ForeignKey(Board, on_delete=models.SET_NULL,
        blank=True, null=True)

```

```

        timestamp = models.DateTimeField(auto_now=True)
        operation = enum.EnumField(Operation, default=Operation.
            UNKNOWN_OPERATION)

```

Listing 4.19: Action Modell

Mit *DateTimeField(auto_now = True)* wird die Zeit der Ausleihe-/Rückgabeaktion genau entsprechend der Zeit des Servers gespeichert.

```

class Session(models.Model):
    # Holds the information about the interaction between user (
    # student) and system
    TERMINAL_STATES = ['timeout', 'finished', 'canceled', ,
        'error_terminated']
    start_time = models.DateTimeField(default=datetime.datetime.
        now)
    state = FSMField(default='session_started')
    last_action_time = models.DateTimeField(auto_now=True)
    student_card = models.ForeignKey(StudentCard, on_delete=
        models.SET_NULL, blank=True, null=True, related_name='+')
    raspi_tag = models.ForeignKey(RaspiTag, on_delete=models.
        SET_NULL, blank=True, null=True, related_name='+')

```

Listing 4.20: Session Modell

An dieser Stelle muss man besonders betonen, dass Feld *state = FSMField(default = 'session_started')* wird für Django Finite State Machine Framework benutzt. Anstatt einem Django-Modell ein Statusfeld hinzuzufügen und seine Werte manuell zu verwalten, wird *FSMField* verwendet und Modellmethoden mit dem Übergangsdekorator markiert [Pod]. Diese Methoden können Änderungen der Zustandsänderung enthalten. Die Implementierung wird in Kapitel 4.2.5 erklärt. Nachdem die Django Modelle entsprechend der Beschreibung aus dem Kapitel 3.5.1 erzeugen wurden, müssen die Tabellen in der Datenbank erstellt werden, bevor sie verwendet werden können. Dazu wird den Befehl ausgeführt:

```
$ python manage.py migrate
```

Der Befehl *migrate* überprüft die Einstellung *INSTALLED_APPS* und erstellt alle erforderlichen Datenbanktabellen gemäß den Datenbankeinstellungen in der Datei *acaLoanRaspiBoard/settings.py* [Fouc].

4.2.3 Einführung in das Django Admin

Einer der mächtigsten Teile von Django ist die automatische Administrationsoberfläche. Es liest Metadaten aus erstellten Modellen, um eine schnelle, modellzentrierte Oberfläche bereitzustellen, über die vertrauenswürdige Benutzer Inhalte auf der

<input type="checkbox"/>	BOARD NUMBER	BOARD TYPE	BOARD STATUS	RASPI TAG
<input type="checkbox"/>	1	LAB_LOAN	LOANED	01 01 01 01
<input type="checkbox"/>	2	LAB_LOAN	ACTIVE	02 02 02 02
<input type="checkbox"/>	3	LAB_LOAN	ACTIVE	6C E9 87 30
<input type="checkbox"/>	4	LAB_LOAN	LOANED	9C 89 88 30
<input type="checkbox"/>	5	LAB_LOAN	ACTIVE	9C 98 89 30
<input type="checkbox"/>	6	LAB_LOAN	ACTIVE	06 06 06 06
<input type="checkbox"/>	7	LAB_LOAN	ACTIVE	07 07 07 07
<input type="checkbox"/>	8	LAB_LOAN	ACTIVE	08 08 08 08
<input type="checkbox"/>	9	LAB_LOAN	ACTIVE	09 09 09 09
<input type="checkbox"/>	10	LAB_LOAN	ACTIVE	10 10 10 10
<input type="checkbox"/>	11	LAB_LOAN	ACTIVE	11 11 11 11
<input type="checkbox"/>	12	HOME_LOAN	ACTIVE	12 12 12 12
<input type="checkbox"/>	13	HOME_LOAN	ACTIVE	12 12 12 12
<input type="checkbox"/>	14	HOME_LOAN	ACTIVE	12 12 12 12
<input type="checkbox"/>	15	HOME_LOAN	ACTIVE	12 12 12 12
<input type="checkbox"/>	16	HOME_LOAN	ACTIVE	12 12 12 12

Abb. 15: Screenshot von Django Admin Site mit Board Tabelle.

Website verwalten können [Foua]. Die Admin-Site verwendet werden kann, indem die URL aufgerufen wird, mit der Die Admin-Site verbunden wurde (im AcaLoan-Projekt ist es `/admin/`). Wenn einen Benutzer zum Anmelden erstellt werden muss, wird es mit dem Befehl `createsuperuser` gemacht. Für die Anmeldung beim Administrator muss der Benutzer standardmäßig das Attribut `is_superuser` oder `is_staff` auf True gesetzt haben. Mithilfe der Django-Admin Site können die Mitarbeiter nicht benutzerorientierten Inhalt verwalten. Z.B. so werden am Anfang jedes Semesters die neuen Datensätzen für die registrierten für das Modul Studierende erstellt.

Um einen Arbeitszeitverbrauch des Mitarbeiter des PSE-Labor zu senken, wurde das kleine Python Script geschrieben, das permanenten Identifikationsnummer (UID) von einer kontaktlosen Speicherplatte der Studierende ablesen und als CSV-Datei auf der Festplatte speichern kann. Nachdem die Datensätze der Studierende in der Datenbank erzeugen und die UIDs mithilfe des Scripts am RFID-Leser abgelesen wurden, kann Admin des acaLoan-Systems die CSV-Datei hochladen. Somit werden die UIDs der Studentenkarten automatisch mit dazugehörigen Studentensätzen über eindeutigen Matrikelnummer verknüpft. Die Seite mit dem Hochladen der CSV-Datei ist nur für den vertrauenswürdigen Admin des Systems zu Verfügung stehen dürfen und den Zugriffsrechten sind von Django zu verwalten, dazu den entsprechenden Schnittstelle mit dem `@staff_member_required` geschützt wird geworden.

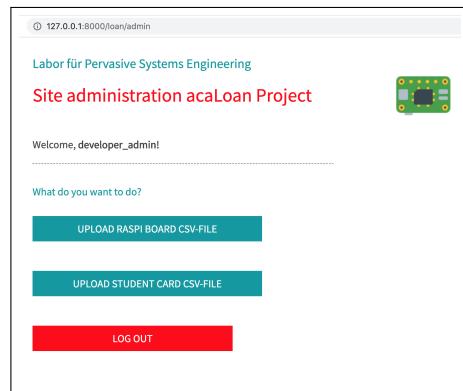


Abb. 16: Admin spezifische Site für CSV-Hochladen

höriegen Studentensätzen über eindeutigen Matrikelnummer verknüpft. Die Seite mit dem Hochladen der CSV-Datei ist nur für den vertrauenswürdigen Admin des Systems zu Verfügung stehen dürfen und den Zugriffsrechten sind von Django zu verwalten, dazu den entsprechenden Schnittstelle mit dem `@staff_member_required` geschützt wird geworden.

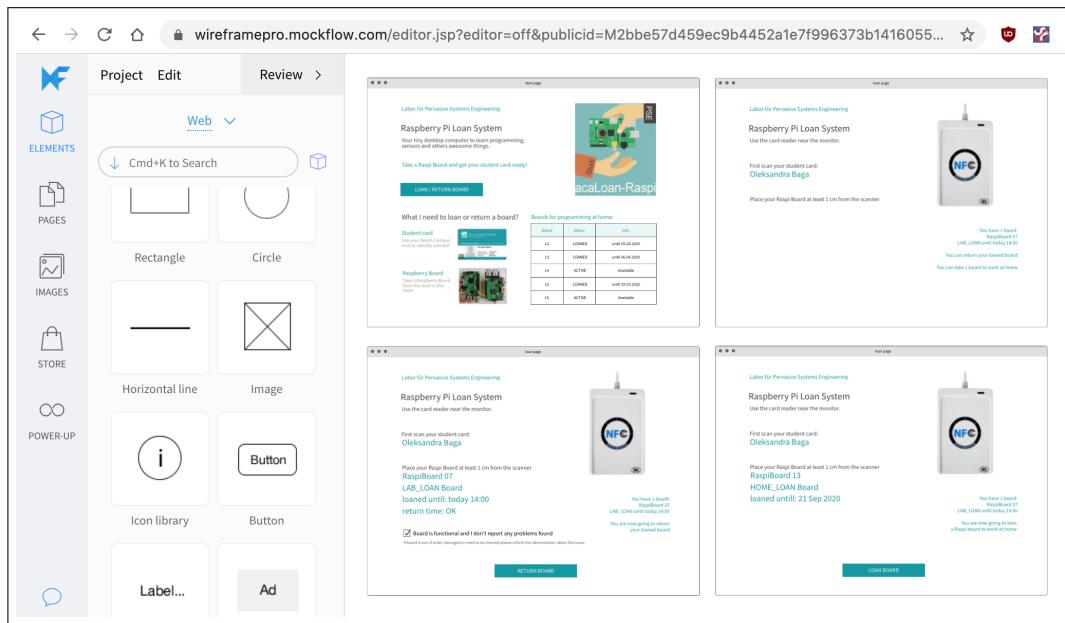


Abb. 17: Screenshot von mockflow.com mit dem Design

4.2.4 Design der acaLoan-Website

Django wird als Framework für die Erstellung vollständiger Websites benutzt. Das Rendern von HTML ist eine der grundlegenden Fähigkeiten von Django. Da das Design für Display-Client erstellt werden muss, wurde der Styl der Beuth Hochschule für Farben und Schriftarten verwendet ein einfaches Website-Design im kostenlosen Online-Tool "mockflow.com" erstellt. Die Seiten sind auf der Abbildung 17 abgebildet. Dann wurden aus dem Design-Bilder direkt ohne automatisierten Dienste die HTML- und CSS-Dateien erstellt, die später mit Django HTML-Vorlagen verwenden werden.

Erstellen der öffentlichen Schnittstelle

Nach dem Speichern der HTML- und CSS-Dateien in `/loan/templates` gespeichert wurden, müssen die entsprechenden Ansichten erstellt werden, um die Seiten zu rendern. Eine Ansicht ist eine spezifische Webseite in der Django-Anwendung, die bestimmte Funktion erfüllt und über eine bestimmte Vorlage verfügbar ist [Foud]. Sie werden durch eine Python-Funktion (oder eine Methode bei klassenbasierten Ansichten) dargestellt. Django wählt eine Ansicht aus, indem er die angeforderte URL überprüft (genauer gesagt den Teil der URL nach dem Domainnamen) [Foud]. Mit der Ansicht "index" wird acaLoan-Projekt beim Starten ausgeführt und ihre HTML-Vorlage wird dem Benutzer im Webbrowser im Vollbildmodus angezeigt. Mit

dem Drücken der Taste "Loan/Return Board" wird eine neue Sitzung in dem Zustand *session_started* angefangen.

```
def index(request):
    now = datetime.datetime.now()
    loans = []
    for b in Board.objects.filter(board_no__gte=
        HOME_LOAN_MINIMAL_NO):
        if b.board_status == BoardStatus.LOANED:
            action = b.action_set.order_by("-timestamp").first()
            deadline = action.timestamp + MAX_HOME_LOAN_LIMIT
            if deadline > now:
                loans.append({'board': b, 'deadline':
                    deadline})
        else:
            loans.append({'board': b, 'deadline': None})
    template_name = "loan/index.html"
    context = {"loans": loans}
    return render(request, template_name, context)
```

Listing 4.22: Index-Ansicht in Django

Hier ist er zu erwähnen, dass in intelligente HTML-Seite die die Variablen anstelle bestimmter Werte verwendet werden kann, um in *index.html* die verfügbare zur Home-Ausleihe Boards dem Studierende anzuzeigen zu können. Hier werden die verfügbaren im Labor Home-Boards und die Boards, deren Rückgabefrist ist immer noch nicht versäumt, aus der Datenbank zugreifen und als Python-Wörterbuch mit Variablennamen der HTML-Vorlage zum rendern übergeben (Abbildung 18). In der Vorlage namens "start" wird den Zustand der Sitzung angezeigt, den wird vom Server über die REST API Kommunikationsschnittstelle dem Display-Client geschickt, die regelmäß abgerufen werden muss. Die Sitzungsaktualisierungen werden in der events-Ansicht behandelt.

What I need to loan or return a board?

Student card

Use your Beuth Campus card to identify yourself
<https://www.beuth-hochschule.de/campus-card>



Raspberry Board

Take a Raspberry Board from the shelf in this room



Boards for programming at home:

Board	Status	Info
12	ACTIVE	Ready to loan
13	ACTIVE	Ready to loan
14	ACTIVE	Ready to loan
15	ACTIVE	Ready to loan
16	LOANED	Loaned until 23.11.2020 13:11

Abb. 18: Teil der Index-Seite: dynamische generierte Tabelle mit verfügbaren im Labor Home-Loan Boards

Django HTML-Vorlage

Als es im Kapitel oben schon erwähnt wurde, wird von einer Django-Ansicht ein Kontext wie ein Python-Wörterbuch mit Variablennamen als Schlüssel und deren Werten als Wert erzeugt und in die Django HTML-Vorlage übergeben. Beispielsweise ist auf der Abbildung 18 die Tabelle mit Home-Loan Boards gezeigt, die im Labor vorhanden sind oder auf dessen baldige Rückkehr erwartet ist. Den Inhalt der Tabelle wurde als ein Kontext aus der Index-Ansicht übergeben und dynamisch generiert. Django unterstützt Jinja, die eine Erweiterung von HTML ist, mit der Daten mit einer doppelten Klammer `{% data %}` und Ausdrücke mit ähnlicher Syntax `{% expression %}` eingefügt werden können.

```
{% for loan in loans %}
<tr>
    <td>{{ loan.board.board_no }}</td>
    <td>{{ loan.board.board_status }}</td>
    <td>
        {% if loan.board.board_status == 1 %}
            Ready to loan
        {% endif %}
        {% if loan.board.board_status == 2 %}
            {% if loan.deadline %}
                Loaned until {{ loan.deadline }}
            {% endif %}
        {% endif %}
    ...
    ...

```

Listing 4.23: Django Template Sprache in index.html

4.2.5 Implementierung des Anwendungsfällen

Vor der Implementierung der User Cases wurde auch Sequenzdiagramm erzeugt, die einfach die Interaktionen zwischen Objekten in einer sequentiellen Reihenfolge zeigt, d.h. die Reihenfolge, in der diese Wechselwirkungen stattfinden. Die Sequenzdiagramm wurde als nächstes als eine Basis für das Design des Zustandsmaschine verwendet.

Sequenzdiagramm

Ein Sequenzdiagramm beschreibt, wie und in welcher Reihenfolge die Objekte in einem System funktionieren. In der Abschlussarbeit werden ein Sequenzdiagramm gezeigt, die auf den Abbildungen 19 zu betrachten ist. Es ist ein erfolgreichen Szenario für die Ausleihe der Lab-Loan Raspi Board von einem gültigen Studierende.

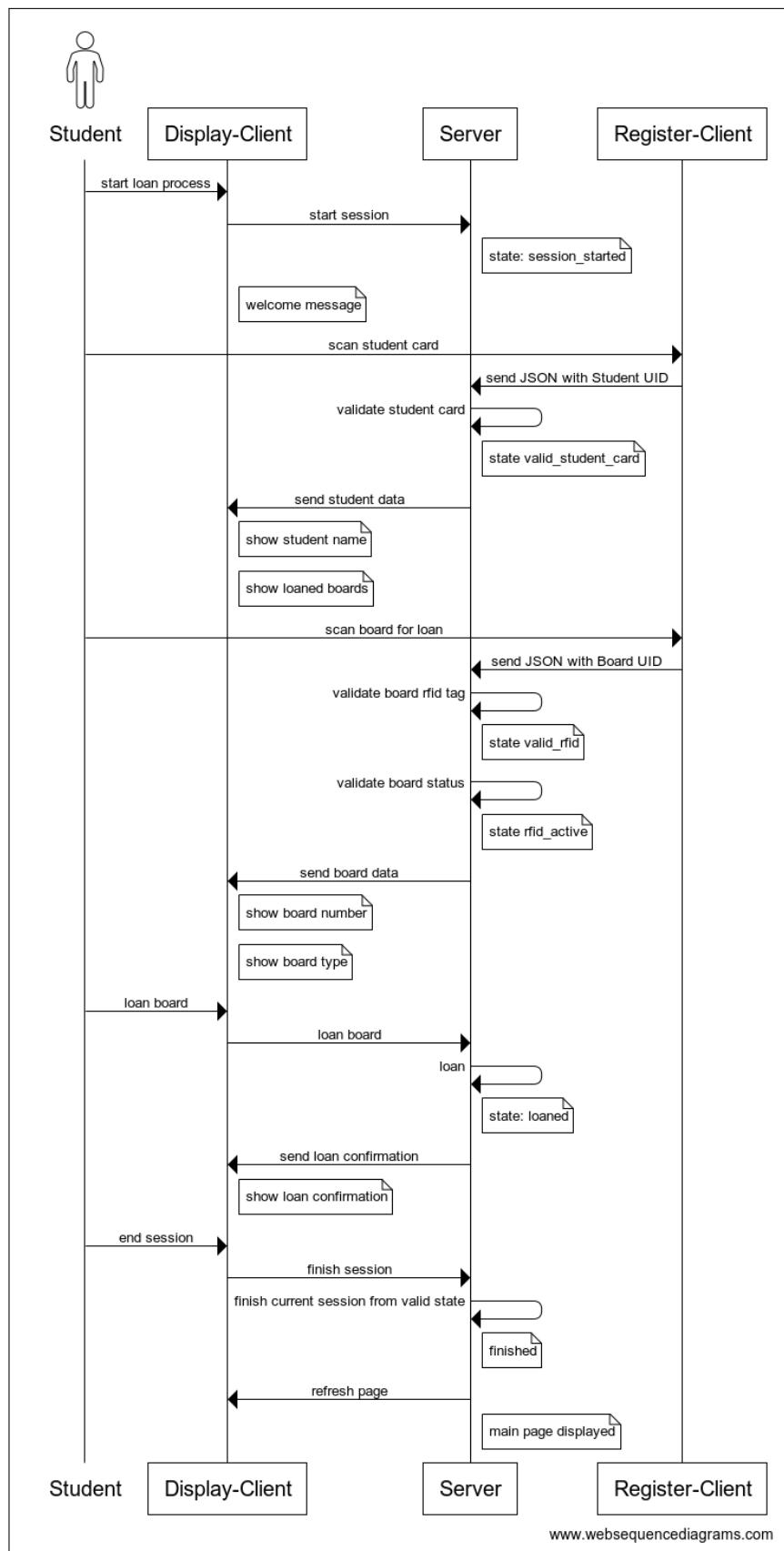


Abb. 19: UML Sequenzdiagramm der erfolgreichen Ausleihe der Lab-Loan Raspi Board

Das Sequenzdiagramme entspricht der oben genannte User Story "As a student, I want to loan a board so I can work at lab". Die meisten Kommunikation geschieht über die asynchrone Nachrichten. Zum Beispiel, es wird von Register-Klient mit dem angeschlossenen RFID-Leser nicht darauf gewartet, ob die abgelesene Studentenkarte gültig ist oder ob ein Student zum Kurs zugelassen ist. Sofort die vorherigen Ablesevorgang angeschlossen wurde und JSON-Datei dem Server geschickt, steht der RFID-Leser wieder zur Verfügung und ein RFID-Tag des Boards abgelesen werden kann. Es kann sogar sein, dass anstatt erwarteten in erfolgreichen Szenario RFID-Tags eine weitere Studentenkarte abgelesen wird (z.B. nebenstehende Kumpel der Studierende aus Spaß oder wegen der Eile lässt seine Karte ablesen vor dem Beenden des laufenden Ausleihevorgang). Obwohl es aus der Sicht der Zustandsmaschine nicht zugelassen ist, eine weitere Studentenkarte zu lesen, wird trotzdem die Karte von RFID-Reader abgelesen, eine weitere JSON-Datei zum Server geschickt und weiter RFID-Leser zum nächsten Lesen freigegeben. Es ist rein die Aufgabe des Servers die ankommenden Daten zu verifizieren und den Zustandsmaschine in einem weiteren Zustand zu schalten. Die entsprechende Information mit der Begrüßung der Studierende oder Fehlermeldung wird auch vom Server generiert und dem Display-Client zum Anzeigen übergeben. Mehr über diese Vorgehensweise in entsprechenden Kapitels 4.1.4, 4.2.5 und ?? der Implementierungsphase nachzulesen.

Endliche Zustandsmaschine mit Django FSM

Die endliche Zustandsmaschine wird mit Django FSM realisiert (auf der Abbildung 20). FSMField des Session Modells wird für eine automatisierte Verwaltung des Zustands der FSM verwendet. Modellmethoden werden mit dem Übergangsdekorator markiert, um die Änderung des Zustands zu ermöglichen. Der FSM wird im Zustand *session_started*" gestartet. Vor der Erstellung einer neuen Sitzung, wird zuerst die Felder des Modells mit der Ausführen der Funktion "clean" validiert. Falls die eine aktive Sitzung schon existiert, wird ein ValidationError ausgelöst:

```
def clean(self):
    open_session = Session.objects.exclude(state__in=Session.
        TERMINAL_STATES).count()
    if open_session != 0:
        raise ValidationError('Active session already exists!
            ')
    super().clean()
```

Listing 4.24: Session clean() für die Validation des Modells

Der nächsten erwarteten Schritt ist das Ablesen der Studentenkarte am Register-Client mit angeschlossenen RFID-Leser. Von Register-Clien am Endpunkte der REST

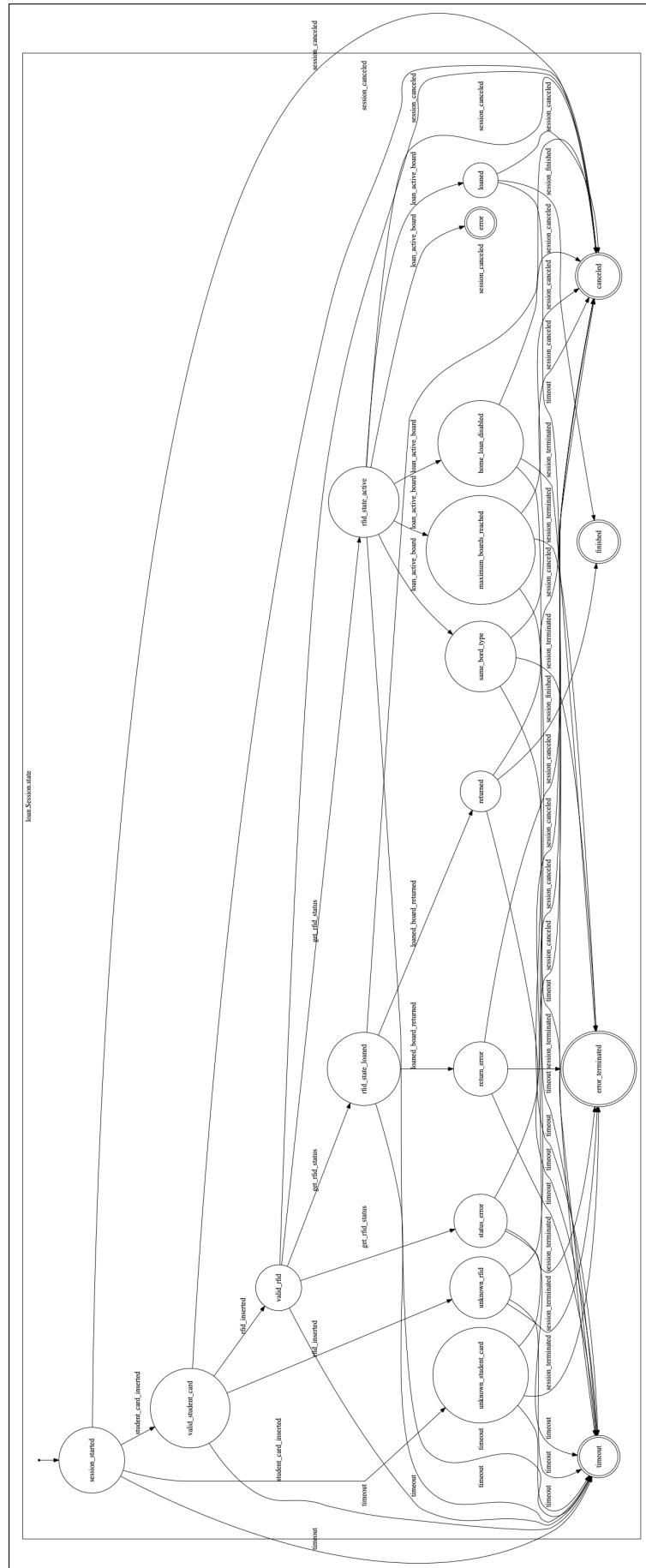


Abb. 20: Django FSM mit Zustnde und bergnge

API wird auf eine JSON-Datei erwartet, die zuerst von FSM überprüft wird, ob sie eine erwartete von dem Server Nachricht erhält.

```

if request.method != "POST":
    return HttpResponseNotFound()
...
try:
    body = json.loads(request.body)
    input_type = body['type']
except (KeyError, json.JSONDecodeError):
    return HttpResponseBadRequest()
if input_type not in ["card", "tag", "cancel_button", "terminate_button", "get_rfid_status",
"return_scanned_board_button", "loan_scanned_board_button", "finish_button"]:
    return HttpResponseBadRequest()

```

Listing 4.25: Events-Ansicht in Django, JSON-Parsing

Falls die JSON-Datei korrekt ist, wird ihre Inhalt für die Übergang der endlichen Zustandsmaschine verwendet.

```

if input_type == "card":
    try:
        uid = body['uid']
        session.student_card_inserted(uid)
    except StudentCard.DoesNotExist:
        pass
elif input_type == "tag":
    uid = body['uid']
    session.rfid_inserted(uid)
elif input_type == "cancel_button":
    session.session_canceled()
...

```

Listing 4.26: Events-Ansicht in Django, Aufur von FSM Transitions

Die nächsten erlaubten bei der FSM Zustände sind: *canceled*, *timeout*, *valid_student_card* und *unknown_student_card*. Ein Übergang in den Zustand *canceled* is immer erlaubt und wird mit dem Drücken der "Cancel"-Taste auf einem Bildschirm des Display-Client ausgelöst. *timeout* wird von Server behandelt und dient dazu, um zu lang gedauerte Sitzungen zwangsweise zu beenden und Display-Client für einen nächsten Versuch wieder freizumachen. Um in einem von beiden Zielzustände zu getreten, muss die Existenz der Studentenkarte in der Datenbank überprüft werden.

```

@transition(field=state, source='session_started', target=
    RETURN_VALUE('valid_student_card', 'unknown_student_card'))
def student_card_inserted(self, card_uid):
    try:
        card = StudentCard.objects.get(uid=card_uid)

```

```

        if card.student is not None:
            self.student_card = card
            return 'valid_student_card'
    except StudentCard.student.RelatedObjectDoesNotExist:
        return 'unknown_student_card'
    except StudentCard.DoesNotExist:
        return 'unknown_student_card'

```

Listing 4.27: FSM Übergang beim Ablesen der Studentenkarte

Die ähnlichen Vorgänge werden bei jedem Übergang der FSM verwenden. So beispielsweise wird den Status der angelesene Board überprüft und dann wird es von FSM festgestellt, ob ein Studierende einen Board ausleihen oder zurückgeben möchte. Es wird vom Studierende nicht erforderlich, die Operation zu bestimmen.

```

@transition(field=state, source='valid_rfid',
target=RETURN_VALUE('rfid_state_loaned', 'rfid_state_active', ,
status_error'))
def get_rfid_status(self):
    board = self.get_active_board()
    if board is not None:
        if board.board_status == BoardStatus.LOANED:
            return 'rfid_state_loaned'
        elif board.board_status == BoardStatus.ACTIVE:
            return 'rfid_state_active'
    ...

```

Listing 4.28: FSM Übergang beim Ablesen der Board RFID-Tag

Die alle implementierte Zustände und Übergänge sind auf der Abbildung 20 zu betrachten. Sie werden mit dem Tool *graphviz* aus den existierenden in dem acaLoan-Projekt Modell *Session* und seine mit dem Django FSM Zuständen und Übergänge automatisiert erzeugt. Sodass werden alle Anwendungsfälle, die im Kapitel 3.3 definiert wurden, erfolgreich mit der Verwendung der Django FSM implementiert.

Nach jeder Änderung des Zustands wird über HTTP-Protokoll vom Server die Nachricht an Display-Client geschickt, sodass es die Änderungen dem Studierende anzeigen könnte. Die Implementierung des Clientseitiges JavaScript wird in folgenden Kapitel beschrieben.

4.3 Display-Client

Display-Client ist der dritte Bestandteil und wird implementiert als interaktive Webseite, die einem Benutzer im in das Vollbildmodus geöffneten Webbrowser angezeigt wird. Die Client-Anwendung besteht aus zwei Teilen:

- Visuelle Darstellung eines Status einer Benutzersitzung mit in HTML beschriebenen Strukturelementen der Benutzeroberfläche mit der Verwendung eines visuellen Styling durch hinzugefügten CSS.
- Javascript-Code, der von einem Browser geladen und ausgeführt wird, um eine dynamische Interaktion mit dem System zu ermöglichen

Ein solcher Ansatz ermöglicht die Nutzung moderner visueller Rendering-Engines und trägt zur Erreichung der Plattformunabhängigkeit bei.

4.3.1 Die Funktionsweise

Bevor der Client verwendet werden kann, muss eine spezielle URL geöffnet werden. Diese URL wird vom acaLoan-Server bereitgestellt, der liefert dem Browser sowohl eine HTML-Seite und CSS-Stylesheets zur Anzeige visueller Elemente als auch Javascript-Code zur Ausführung. Wenn der Browser die HTML-Seite lädt, werden eingehende Daten abgelesen und zusammen mit dem Rendern für einen Benutzer wird eine interne Darstellung aller auf einer Seite vorhandenen Elemente in eine baumartige Struktur namens Document Object Model. Mithilfe einer Reihe von Programmierschnittstellen, die von einem Browser bereitgestellt werden, können Attribute von DOM-Objekten oder die Struktur von DOM geändert werden. Jede Änderung am DOM, sowohl das Hinzufügen, Entfernen oder Reorganisieren von Elementen als auch das Ändern des Inhalts oder des Attributwerts des Tags, kommt sofort zur Wirkung.

Unter anderem bietet der Browser die Möglichkeit, eine asynchrone HTTP-Anforderung an einen Remote-Webserver auszuführen. In diesem Fall bedeutet asynchron, dass die Benutzerinteraktion mit der Webseite für die Zeit, in der die Anforderung ausgeführt wird, nicht blockiert wird. Die Standard-API-Schnittstelle für solche Vorgänge heißt XMLHttpRequest. Diese Programmierschnittstellen werden vom Browser über eine Javascript-Codeausführung verfügbar gemacht. Der Entwickler kann ein Javascript-Programm erstellen, das mit externen Diensten kommunizieren und DOM-Elemente aktualisieren darf, um Änderungen anzuzeigen.

Die Kombination dieser Funktionen dient als Grundlage für die Erstellung eines interaktiven Client-Flows:

1. Die Clientanwendung stellt eine Kommunikation mit einem Server her und empfängt Aktualisierungen über den aktuellen Status einer Benutzerinteraktion

2. Wenn der Server mit dem aktuellen Status antwortet, kann er mit einem derzeit bekannten Anwendungsstatus eines Clients verglichen werden, und bei Bedarf können Änderungen am DOM vorgenommen werden.
3. Da HTTP-Anforderungen für einen Benutzer transparent sind und Änderungen am DOM sofort sichtbar sind, entsteht eine echte interaktive Erfahrung.

4.3.2 Start Webseite des acaLoan-Projekts

Labor für Pervasive Systems Engineering

Raspberry Pi Loan System

Your tiny desktop computer to learn programming, sensors and others awesome things.

Take a Raspi Board and have your student card ready!

LOAN / RETURN BOARD

What I need to loan or return a board?

Student card
Use your Beuth Campus card to identify yourself
<https://www.beuth-hochschule.de/campus-card>

Raspberry Board
Take a Raspberry Board from the shelf in this room

Boards for programming at home:

Board	Status	Info
12	ACTIVE	Ready to loan
13	ACTIVE	Ready to loan
14	ACTIVE	Ready to loan
15	ACTIVE	Ready to loan
16	LOANED	Loaned until 23.11.2020 13:11

Abb. 21: Start Webseite des acaLoan-Projekts

Zunächst wird dem Benutzer eine Zielseite angezeigt. Diese Seite verfügt über eine HTML-Taste (siehe Abbildung 21), die der Studierende drucken muss, um eine neue Benutzersitzung anzufangen. Durch Klicken auf diese Taste "Loan / Return Board" wird eine synchrone HTTP-POST-Anforderung an einen Server gesendet, der die Erstellung einer neuen interaktiven Benutzersitzung anfordert. Im Fehlerfall antwortet der Server mit einem Fehler-HTTP-Statuscode und der Browser zeigt eine Fehlermeldung an.

```
Forbidden: /loan/api/sessions
"POST /loan/api/sessions HTTP/1.1" 403 47
```

Wenn keine aktive Sitzung vorhanden ist, treten folgende Ereignisse auf:

1. Der Server erstellt eine neue Benutzersitzung

```
"GET /loan/start HTTP/1.1" 200 5423
"POST /loan/api/sessions HTTP/1.1" 201 120
"GET /loan/api/sessions/17 HTTP/1.1" 201 216
```

2. Der Server antwortet mit dem HTTP-Statuscode 201 und liefert dem Browser das grundlegende Layout der Sitzungsinteraktionsseite. In der Beispiel oben wurden die neue Benutzersitzung mit der Nummer 17 erstellt.
3. Der Browser verarbeitet die HTML-Seite und lädt zusätzliche statische Elemente wie CSS-Stylesheets und JavaScript-Code vom Server.
4. Die HTML-Seite wird generiert, JS-Code wird ausgeführt. Die interaktive Sitzung hat begonnen. Die ist auf der Abbildung 22 angezeigt. In dem Zustand der FSM *session_started* wird zuerst keine Information gegeben, wer der Studierende ist und mit welchem Board gearbeitet ist. Die Webseite zeigt dem Studierende nur die Hinweise, die ihn das Prozess der Ausleihe-/Rückgabe erleuchten können.

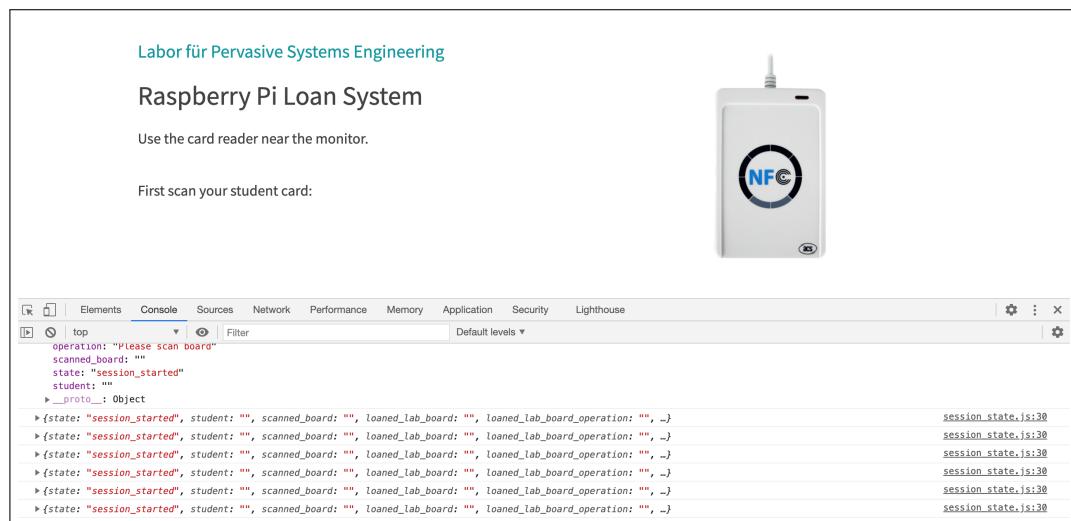


Abb. 22: Webseite für eine neue Benutzersitzung mit dem Console-Log

4.3.3 Interaktive Benutzersitzung mit JavaScript

Nachdem die Benutzersitzung gestartet ist. dient die folgenden Bedingungen:

- Der Sitzungsstatus wird auf dem Server geändert
- Der Benutzer interagiert mit einem Elementen der angezeigten Webseite

Da das HTTP-Protokoll keine bidirektionale Kommunikation bietet und die Möglichkeiten des Servers, Daten proaktiv an einen Client zu übertragen, sehr begrenzt sind, muss regelmäßig nach einer Änderung des Sitzungsstatus gefragt werden. Um dies zu erreichen, legt der Javascript-Code eine regelmäßige Aktualisierung des Sitzungsstatus

fest durch Registrieren eines Timer-Ereignisses mit dem Aufrufen einer setInterval()-Funktion.

```
function session_started(body) {
    console.log("Started session with id " + body.id)
    active_session_id = body.id
    session_refresh_timer = setInterval(refresh_session_state,
        1000)}
```

Listing 4.31: function session_started()

Die Entwicklung mit einfachen Javascript-APIs ist umständlich und fehleranfällig. Um die Codebasis zu vereinfachen und später zu machen Unterstützung und Änderungen einfacher, eine Reihe von Frameworks wurden erstellt. Ziel eines solchen Frameworks ist es, Entwickler von zu isolieren interne Komplexität von reinen JS-APIs und bieten browserübergreifende Kompatibilität, wenn verschiedene Browser implementiert werden bestimmte API auf andere Weise. Dieses Projekt wurde mit einer plattformübergreifenden Javascript-Bibliothek namens jQuery erstellt. In Quellcode unten wird mit `jQuery.ajax(url[, settings])` eine asynchrone HTTP-Anforderung (Ajax) ausgeführt.

```
function refresh_session_state() {
    $.ajax({url: "api/sessions/" + active_session_id, method: "GET"
    })
    .done(handle_session_event)
    .fail(function() {
        clearInterval(session_refresh_timer)
        alert("Timeout/Canceled. Please start again!")
        window.location.replace("/loan")
    })}
}
```

Listing 4.32: Function refresh_session_state()

Die Funktion `handle_session_event` wird nach erfolgreichem Abschluss einer Ajax-Anforderung ausgerufen und hat die folgenden Funktionsweise:

- Die Funktion verwendet die XMLHttpRequest() nach Browser-API, um eine asynchrone HTTP-Anforderung an den Server zu senden
- Die asynchrone HTTP-Anforderung wird so eingerichtet, dass eine vom Benutzer bereitgestellte Funktion aufgerufen wird, wenn der Server eine erfolgreiche Antwort liefert.
- Basierend auf dem Inhalt der Antwort wird der Status des DOM angepasst, um den Sitzungsstatus widerzuspiegeln und den Benutzer über die erforderlichen Aktionen zu informieren. Die Fehlermeldungen werden auch von dieser Funk-

tion über Status des DOM angepasst, z.B. in folgenden Quellcode wird es die Fehlermeldung gegeben über Verbot einen Board nach Hause auszuleihen:

```
if (body.state == "home_loan_disabled") {
    $("#student_name").text("You can not loan board
        to work a home")
    $("#operation_info").text("Home loan is disabled
        . Ask teaching assistant or administrator for
        a help")
    $("#warning_picture").show()
    decorate_terminate_state(body)}
```

Listing 4.33: Änderung des DOM im Fall des Verbot der Home-Ausleihe

- Der neue Sitzungsstatus wird im Client zur späteren Bezugnahme gespeichert.

4.3.4 Client-Server-Kommunikationsprotokoll

Beim regelmäßigen Abrufen von Sitzungsaktualisierungen vom Server erwartet der Client, dass der Server mit JSON-Nachrichten antwortet. Die Clientanwendung analysiert diese Nachrichten und vergleicht sie mit dem vorherigen bekannten Status. Hier wird es auch auf die UIDs der Studentenkarten und RFID-Tags der Rapsi-Boards erwartet.

```
if input_type not in ["card", "tag", "cancel_button", "
    terminate_button", "get_rfid_status", "return_scanned_board_button
    ", "loan_scanned_board_button", "finish_button"]:
    return HttpResponseBadRequest()

...
elif input_type == "cancel_button":
    session.session_canceled()
elif input_type == "finish_button":
    session.session_finished()
elif input_type == "get_rfid_status":
    session.get_rfid_status()
elif input_type == "return_scanned_board_button":
    session.loaned_board_returned()
elif input_type == "loan_scanned_board_button":
    session.loan_active_board()
elif input_type == "terminate_button":
    session.session_terminated()
```

Listing 4.34: Sitzungsaktualisierungen mit JSON

Einem Client bekannten Status der endliche Zustandsmaschine wird dem im Kapitel 3.5.3 und 4.2.5 beschriebenen Sitzungsstatus zugeordnet. Beispielweise in dem Anwendungsfall "Ausleihe des Home-Loan Boards" (Kapitel 3.3.2) gibt es einen Übergang aus dem Zustand *rfid_state_active* nach entweder erfolgreichem Zustand

loaned oder einem von mehreren vordefinierten fehlerhaften Zustände. Der Auslöser des Übergangs ist der Druck der Taste "Loan Board".

```
@transition(field=state, source='rfid_state_active', target=
    RETURN_VALUE('loaned', 'home_loan_disabled', 'error', 'maximum_boards_reached', 'same_bord_type'))
def loan_active_board(self):
    result = self.board_loaned()
    if result == 'loaned':
        Board.loan_board(self.raspi_tag)
return result
```

Listing 4.35: Übergang aus dem Zustand rfid_state_active

Während des Beispielübergangs der FSM mithilfe der Funktion *board_loaned* wird der aktive Studierende aus der Sitzung mit dem Aufruf der Funktion *get_active_student* abgerufen, um seine Rechte zum Home-Ausleihe zu überprüfen und eine Liste der zugewiesenen ihm Boards zu bekommen. Nachdem wird es festgestellt, dass der Studierende den Board ausleihen darf, wird die neue Aktion in Datenbank erstellt. Operation wird während des Ausführen der Funktion abhängig vom dem Typ des Boards bestimmt und kann entweder *Operation.HOME_LOAN* für den Board *BoardType.HOME_LOAN* oder *Operation.HOME_LOAN* für den Board *BoardType.HOME_LOAN* sein. Endlich wird der Status des Boards auf "Loaned" gesetzt.

```
Board.objects.filter(raspi_tag=raspi_tag).update(board_status=
    BoardStatus.LOANED)
```

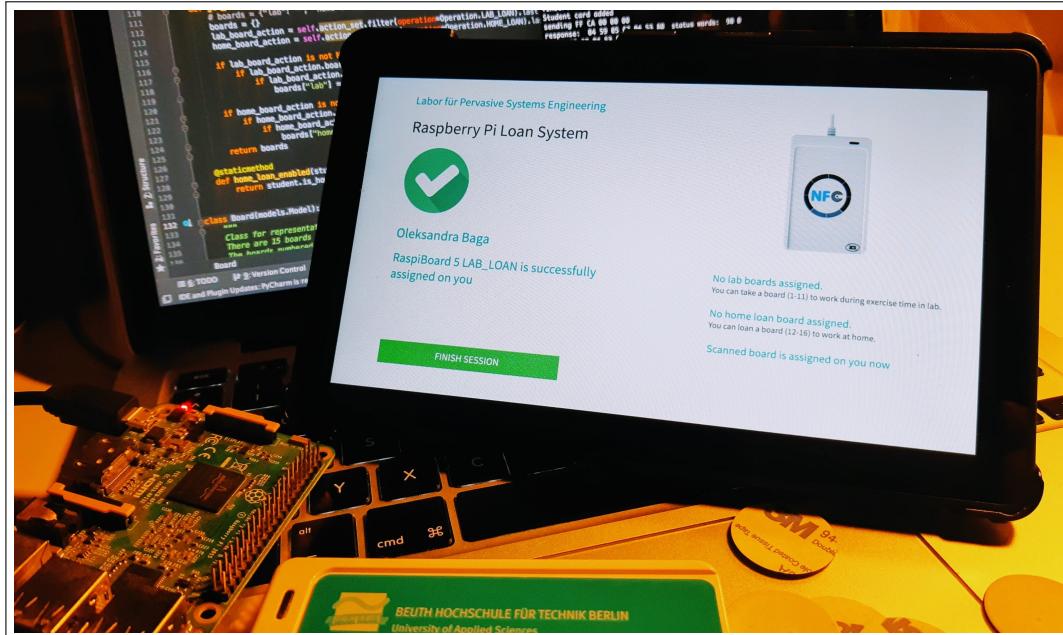


Abb. 23: Webseite für einen Endzustand nach der Ausleihe des Boards

Die endliche Zustandsmaschine in einen Zustand *loaned* gewechselt. Dann wird es mit der nächsten regelmäßigen Abrufen von Sitzungsaktualisierungen wieder den Status der DOM angepasst, um eine erfolgreiche finale Zustand zu stylen. So werden mittels JavaScript die Tasten "Loan Board" und "Cancel" Tasten verborgen und die neue Taste "Finish" angezeigt, beim deren Drucken wird FSM in einem Endzustand gewechselt. Somit wird der Benutzersitzung beendet, da der Zustand "finished" ein von mehreren Endzuständen ist, mit denen FSM beenden darf und somit die neue Sitzung angefangen werden darf. Den Display-Client mit einer Webseite nach dem erfolgreichen Ausleihe eines Boards ist auf der Abbildung 23 gezeigt.

4.4 Automatisierten Tests

Django bietet ein Testframework mit einer kleinen Hierarchie von Klassen, die auf der unittest-Python-Standardbibliothek aufbauen. Trotz des Namens eignet sich dieses Testframework sowohl für Unit- als auch für Integrationstests. Das Django-Framework fügt API-Methoden und -Tools hinzu, mit denen das Web- und Django-spezifische Verhalten getestet werden kann. Mit diesen können Sie Anforderungen simulieren, Testdaten einfügen und die Ausgabe Ihrer Anwendung überprüfen [conal].

Für Testing muss eine Datenbank mit Datensätzen gefühlt werden. In dem Fall des acaLoan-Project bedeutet es, sowohl notwendige UIDs der Studentenkarten als auch die Namen, Vornamen und sogar Matrikelnummern sollen bei jedem Testen erzeugt werden. Das gilt auch für die UIDs des Boards mit den verschiedenen Status und Typen. Alle Zustände die endliche Zustandsmaschine sollen auch getestet werden, dafür muss nicht nur FSM in einem Zustand gesetzt werden sondern auch die notwendigen Datensätze in der Tabelle "Aktion" und "Session" vorbereiter werden. Jedes Mal es manuell wiederholen wäre zu aufwändig. Deshalb werden die *factory_boy – Bibliothek* (nach dem Vorbild von Factory Girl in Rails) benutzt, um die Daten für Tests zu generieren. In Django wurden Daten für Tests als Fixtures bezeichnet, die aus Dateien im Code geladen wurden. Beispielweise in dem Quellcode wird so viel wie notwendig "korrekten" UID generiert (sie sind nur aus der Datensicht korrekt, da sie nicht von offiziellen Herstellen gegeben wurden und sind somit nicht einzigartig)

```
class FuzzyUid(factory.fuzzy.BaseFuzzyAttribute):
    def __init__(self, length):
        self.length = length
        super().__init__()

    def _segment(self):
        x = randgen.randint(0, 255)
        return "{:02X}".format(x)
```

```

def fuzz(self):
    return ' '.join([self._segment() for _ in range(self.length)])

```

Listing 4.37: Factory für UIDs

Da eine Studentenkarte muss aus 7 Segmenten bestehen und ein RFID-Tag aus nur 4 Segmenten, ist der Rest der Struktur gleich und die FuzzyUid-Klasse kann für die Generierung sowohl UIDs der Studentenkarten als auch RFID-Tags verwendet werden: es wird nur den Parameter *length* angepasst.

```

class StudentCardFactory(DjangoModelFactory):
    class Meta:
        model = StudentCard
    atr_hex = ATRCardType.STUDENT_CARD_ATR
    uid = FuzzyUid(length=7)

```

Listing 4.38: StudentCardFactory

Factory_boy verwendet *FakeFactory(Faker)*, um zufällige Werte zu generieren. Das ist besonders nützlich für die Generierung von Namen, Vornamen, Adressen, IP4-Adressen und andren Sätzen, die in realen Datenbanken verwendet werden können. Benutzung der *Localede_DE* zulässt die deutschen Namen und Städten zu generieren. Im Beispiel unten werden "die Studierende" generiert, für denen die Matrikelnummern und HRZ-Nummern werden als zufälligen Werten in Rahmen vordefinierten Bereich ausgewählt.

```

class StudentFactory(DjangoModelFactory):
    class Meta:
        model = Student
    student_card = factory.SubFactory(StudentCardFactory)
    first_name = factory.Faker('first_name')
    second_name = factory.Faker('last_name')
    matricul_no = factory.Faker('random_int', min=850000, max=950000)
    hrz_no = factory.Faker('random_int', min=65000, max=69000)
    group = StudentGroup.A_GROUP
    is_home_loan_enabled = True

```

Listing 4.39: StudentFactory

Während der mehreren Ausführungen der Tests es hat sich ein paar Mal Fehler aufgetreten: von Factory mehr als ein Student mit der gleichen Matrikelnummer generiert wurde. In Django Model "Student" ist das Feld *matricul_no* mit *unique = True* markiert. Die Tests müssen einfach erneut geladen werden, dann tritt höchstwahrscheinlich der Fehler nicht wieder auf.

Der Django-Test ist eine Funktion, die mit einem vorbereiteten Inhalt die Änderungen in der Datenbank macht und dann vergleicht das Ergebnis mit einem erwarteten Zustand. Sodass prüft folgende Funktion, ob einem Studierende, der von Factory erzeugt wurde und dem den Flag `home_loan_enabled` nicht gesetzt wurde, eine Home-Ausleihe gelingen kann:

```
def test_home_loan_disabled(self):
    session = Session.objects.create(state='rfid_state_active',
                                      student_card=self.student_home_disabled.student_card,
                                      raspi_tag=self.home_board_active.raspi_tag)
    self.assertEqual(session.board_loaned(), 'home_loan_disabled')
)
```

Listing 4.40: Test für einen Student

```
[→ acaLoanRaspiBoard git:(master) ✘ python manage.py test loan.tests.test_lab_loan_board
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 6 tests in 0.137s

OK
Destroying test database for alias 'default'...
[→ acaLoanRaspiBoard git:(master) ✘ python manage.py test loan.tests.test_home_loan_board
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 8 tests in 0.156s

OK
Destroying test database for alias 'default'...
[→ acaLoanRaspiBoard git:(master) ✘ python manage.py test loan.tests.test_fsm_transitions
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 40 tests in 0.553s

OK
Destroying test database for alias 'default'...
[→ acaLoanRaspiBoard git:(master) ✘ python manage.py test loan.tests.test_api
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 44 tests in 0.942s

OK
Destroying test database for alias 'default'...
+ acaLoanRaspiBoard git:(master) ✘
```

Abb. 24: Terminal nach der Ausführung aller Tests

Sodass wurde die REST API, die endliche Zustandsmaschine und alle Anwendungsfälle des acaLoan-Systems fehlerfrei getestet. Insgesamt 98 Testfunktionen wurden geschrieben und ausgeführt.

Analyse

Bei der Entwicklung benutzerdefinierter Webanwendung "acaLoan" wurde Python einer anderen Programmiersprache bevorzugt, um ihre einfache und ausdrucksstarke Syntax zu nutzen. Da Django in der Programmiersprache Python geschrieben ist, wird es auch sinnvoller, die Abschlussarbeit in Python Code zu erstellen, indem die alle Komponenten Syntaxregeln von Python nutzen. Django stellt die Ressourcen bereit, die während der Implementierung der analysierte Software sofort benötigt wurden: allgemeine Vorgänge wie Datenbankmanipulation, HTML-Vorlagen, URL-Routing, Sitzungsverwaltung und Sicherheit. Mit der MVC-Prinzip (Model-View-Controller) sind die Benutzeroberfläche (UI) und die Geschäftslogikebenen des acaLoan-Systems getrennt. Django unterstützt verschiedene Betriebssysteme wie Windows, Linux und MacOS, damit es keine strenge Softwareabhängigkeiten bei acaLoan-System während sowohl der Bereitstellung des Djangos mit ASGI als auch in der Zukunft gegeben werden sollten.

Für sowohl eine Markierung der Raspi-Boards als auch für eine Authentifizierung der Studierenden werden RFID-Tags benutzt. Da RFID Funkwellen zur Kommunikation verwendet werden, müssen sich RFID-Tags nur innerhalb des Lesebereichs des Lesegeräts befinden. Das lässt die kleinen NFC-Tags auf der Rückseite des jeden Boards unter dem Schutzschirm zu kleben, sodass der Board am RFID-Leser ohne präzise Positionierung auf dem Lesefeld angelesen werden kann. Da die Studentenkarte der Beuth Hochschule mit MIFARE DESFire kontaktlose Chipkartentechnik hergestellt sind, wird es möglich, in acaLoan-System nur mit einem RFID-Leser zu arbeiten und sowohl die Daten der Studierende als auch die Raspi-Boards abzulesen.

An dieser Stelle muss es gesagt werden, dass die RFID-Tags bestimmte Nachteile haben, die wurden zwischen den Mitarbeiter und Autorin der Abschlussarbeit diskutiert. Es wurde jedoch die Entscheidung getroffen, dass RFID-Tags für die vorgesehenen Zwecken des Verfolgen des Raspi-Boards (welchen Raspi-Board von welchem Student am welchen Tag ausgeliehen wurde und bis zum welchen Tag zurückgegeben muss) die Sicherheitserwartungen der PSE-Labor Mitarbeiter erfüllen. RFID-Tags sind aus mehreren Gründen im Vergleich nicht ideal. Da ein RFID-Tag nicht zwischen Lesegeräten unterscheiden kann, können die Informationen von fast jedem gelesen werden, sobald sie die ursprüngliche Lieferkette verlassen haben. Weil RFID-Lesegeräte so tragbar sind und die Reichweite einiger Tags so groß ist, können

Betrüger Informationen sammeln, auf die sie sonst keinen Zugriff hätten. Dies bedeutet, dass jeder ohne Wissen einer Person potenziell sensible Informationen sammeln kann. Diese Nachteile der RFID-Tags wurden vernachlässigt, da sowohl Studentenkarte als auch geklebte auf den Raspi-Boards RFID-Tags keine sensible Information behalten und eine kleine Reichweite von bis zu 10 cm haben. Es gibt trotzdem ein Gefahr, dass entweder die Studentenkarte geklont von einem Täter wird, um sich für einen Student ausgeben und ein Raspi-Board stehlen zu können, oder ein Raspi-Board geklont wird, um ein Datensatz in der Datenbank zu erzeugen, dass schon ausgeliehenen Board quasi zurückgegeben wurde, obwohl in der Realität den Raspi-Board nie ins Labor zurückgekommen war. Gegen das Klonen des Raspi-Tags wurde es besprochen, dass in der Zukunft im PSE-Labor im Schrank mit den Raspi-Boards jeden Platz für jeden entsprechenden Raspi-Board mit einem Gewichtssensor ausgerüstet werden wird und acaLoan-System auf der Erscheinung des bestimmten Gewicht erwarten wird. Dies ist aber nicht der Teil bestehenden Abschlussarbeit und von Mitarbeiter des PSE-Labor als eine spannende Aufgabe für die andere Abschlussarbeit vorgesehen ist. Gegen das Klonen der Studentenkarte wurde es zuerst entschieden, dass ein bestehenden Zugang zu einem Schrank mit Raspi-Boards entlang die beide Arbeitstischen der Mitarbeiter des PSE-Labor eine bestimmte Sicherheit gewährleisten könnte. Die anderen Lösungen werden nach der Lieferung der Software diskutieren und liegen ebenfalls außer den Rahmen der bestehenden Abschlussarbeit.

Obwohl oben die Nachteilen der RFID-Tags erwähnt wurde, es lässt sich zusammenfassend sagen, dass die neue Studentenkarte, die an der Beuth Hochschule ab Sommersemester 2018 verwendet wurden, sind eine zuverlässige und zeitgemäße Lösung. Die Karte beinhaltet keine elektronischen persönlichen Daten der Studierenden und die Campus-Automaten alle persönlichen Daten anhand eines Pseudonyms online abrufen müssen (d.h. liegen in den Automaten auch keine persönlichen Daten vor). Sodass im Fall des Verlusts die persönliche Daten von den Unberechtigte nicht ausgelesen werden können [TBb]. Das stand im Fokus der Entscheidung, eine Studentenkarte als einzige elektronischer Identifizierungsmittel beim Ausleihe/Rückgabevorgänge im PSE-Laboz zu benutzen.

In dieser Abschlussarbeit wurde nachgewiesen, dass das Verleihprozedere für die Raspi-Boards (Lab und Home) mit gewählten Mittel für die Verwendung im PSE-Labor erfolgreich automatisiert wurden.

Zusammenfassung

Das zu Beginn der Arbeit gesetzte Ziel der Entwicklung von Software für PSE-Labor wurde erfolgreich erreicht. Die Implementierung alle Bestandteilen und ein Zusammenspiel unter der Verwendung des Entwicklungsservers wurde erfolgreich den Mitarbeitern des PSE-Labors präsentiert. Mit dem acaLoan-System werden die Studierende in die Lage versetzt, einen Raspi-Board selbständig im PSE-Labor für die Übungen auszuleihen und später zurückzugeben. Damit wird ein Arbeitsszeitverbrauch für die Verwaltung der Bordstandortbestimmung reduziert und die Mitarbeitern können sich auf weitere neuen geistliche wissenschaftlichen Herausforderungen konzentrieren und damit in der Weiterentwicklung des acaLab-Projekte hineinbringen. Der Funktionsumfang der acaLoan-System erlaubt es den Studierenden, den Raspberry Board entweder für die Übung im Labor oder für die selbständige Arbeit zu Hause auszuleihen.

Darüber hinaus wird der Kommunikation der Studierende mit dem acaLoan-System auf notwendigen Minimum reduziert, sodass wird ein Ausleihe- / Rückgabevorgang nur mit drei Tastendruck erledigt. Zuerst wird die Taste "Loan / Return Board" gedrückt, um eine neue Benutzersitzung zu starten. Das Ablesen der Studentenkarte und Raspi-Boards geschieht ohne weitere Tastendrucken nur über das Register-Client, der die abgelesene Daten dem Server in den Endpunkt der REST API schickt. Es wird abhängig von dem Typ des Boards bestimmt, ob abgelesenen Board zum Ausleihen oder Rückgabe ist. Mit dem zweiten Tastendruck bestätigt der Studierende seinen Wunsch, der Board auszuleihen oder zurückzugeben. Mit dem dritten Tastendruck nach dem erfolgreichen Beenden der Operation, enden der Studierende seine Sitzung und gibt acaLoan-System dem nächsten Studierenden wieder frei.

Für die Implementierung acaLoan-System ist ein ausführliches Software Engineering in Form der objektorientierten Analyse und des Designs der endlichen Zustandsmaschine durchgeführt worden. Das acaLoan-System konnte ohne Server nicht realisiert werden, da Benutzerdaten länger als eine Sitzung gespeichert werden müssen: die Studentenkarten und die Datensätze alle Ausleihvorgänge müssen mindestens für ein laufenden Semester gespeichert werden, damit die Mitarbeiter des PSE-Labor immer einen Zugang zu allen gespeicherten vorherigen Leihvorgangs (von der Ausleihe bis zur Rückgabe eines Boards) bekommen können. Der Server ist erfolgreich mit Django Webframework realisiert. Es ist auch vorgesehen, dass am Ende des Semester

nach dem letzte Rückgabe eines Boards die Datensätzen des zu Ende gegangen Semesters gelöscht werden. Der Zugriff von Server auf Display-Client wird durch entsprechende Schnittstellen mithilfe der RESTful Kommunikation ermöglicht. In der realisierten Abschlussarbeit wurde auch ein notwendigen Teil namens Register-Client entwickelt, an dem ein RFID-Leser angeschlossen wurde. Der Register-Client verfügt selbst über keinen Datenbank und darf nur die abgelesene Daten dem Server schicken. Es geht um eine Simplex-Verbindung, da ein Nachrichtenverkehr asymmetrisch ist, weil der Register-Client keine Daten vom Server zurückbekommen darf und über den erfolgreiche oder gescheiterte Leihvorgang nicht wissen muss. Für die Implementierung des Register-Clients wurde uComputer Raspberry Pi benutzt, der schon zur Verfügung im PSE-Labor stand und ergänzend nicht geliefert werden muss.

Glossar

AJAX

AJAX (asynchrones Javascript und XML) ist der allgemeine Name für Technologien, mit denen asynchrone Anforderungen (ohne erneutes Laden von Seiten) an den Server gestellt und Daten ausgetauscht werden können. Da die Client- und Serverteile der Webanwendung in verschiedenen Programmiersprachen geschrieben sind, müssen zum Austausch von Informationen die Datenstrukturen (z. B. Listen und Wörterbücher), in denen sie gespeichert sind, in das JSON-Format konvertiert werden.

ASGI

Asynchronous Server Gateway Interface ermöglicht nicht nur mehrere eingehende und ausgehende Ereignisse für jede Anwendung, sondern auch eine Hintergrundkoutine, damit die Anwendung andere Aufgaben ausführen kann.

BGA

BGA oder Ball Grid Array ist eine Art oberflächenmontiertes Gehäuse, das in elektronischen Produkten zur Montage integrierter Schaltkreise wie Mikroprozessoren, FPGAs, WiFi-Chips usw. verwendet wird. Die Anschlüsse liegen in Form von Lötkugeln vor, die in einem Gitter angeordnet wie Muster auf der Unterseite des Gehäuses sind, um den für die Verbindungen verwendeten Bereich zu vergrößern.

Daemon

Ein Unix-Daemon ist ein Programm, das "im Hintergrund" ausgeführt wird, ohne dass die Steuerung über ein Terminal erforderlich ist, und dem Benutzer die Möglichkeit bietet, andere Prozesse "im Vordergrund" auszuführen. Der Dämon kann entweder von einem anderen Prozess gestartet werden, z. B. von einem der Systemstartskripte, ohne auf ein Steuerterminal zuzugreifen, oder vom Benutzer von einem beliebigen Terminal aus. In diesem Fall "entführt" der Dämon das Terminal jedoch nicht, während es ausgeführt wird.

DOM

DOM (Document Object Model) ist die Struktur einer HTML-Seite. Bei der Arbeit mit dem DOM werden HTML-Tags (Elemente auf einer Seite) gefunden, hinzugefügt, geändert, verschoben und entfernt.

GPU

Die Grafikverarbeitungseinheit ist ein programmierbarer Prozessor, der auf das Rendern aller Bilder auf dem Computerbildschirm spezialisiert ist. Eine GPU bietet die schnellste Grafikverarbeitung, und für Gamer ist die GPU eine eigenständige Karte, die an den PCI Express (PCIe) -Bus angeschlossen ist. GPUs wurden ursprünglich entwickelt, um Bilder für Computergrafik zu erstellen, jedoch seit Anfang 2010 können GPUs auch verwendet werden, um Berechnungen mit großen Datenmengen zu beschleunigen.

HOST

Host - Dies der Name der IP-Adresse für den Webserver, auf den zugegriffen wird. Dies ist normalerweise der Teil der URL, der unmittelbar auf den Doppelpunkt und zwei Schrägstriche folgt.[RR03, p.31]

HTML-Vorlage

Eine HTML-Vorlage ist eine intelligente HTML-Seite, die Variablen anstelle bestimmter Werte verwendet und verschiedene Operatoren bereitstellt: if (if-then),

for-Schleife (Durchlaufen einer Liste) und andere. Das Abrufen einer HTML-Seite aus einer Vorlage durch Ersetzen von Variablen und Anwenden von Operatoren wird als Vorlagenrendering bezeichnet. Die resultierende Seite wird dem Benutzer angezeigt. Falls einen anderen Abschnitt zu öffnen ist, muss ein anderen Musters geladen werden. Wenn andere Daten in der Vorlage verwendet werden müssen, werden sie vom Server angefordert. Alle Formularübermittlungen mit Daten sind auch AJAX-Anforderungen an den Server.

HTTP

HTTP steht für HyperText Transfer Protocol, Hypertext Transfer Protocol". HTTP ist ein weit verbreitetes Datenübertragungsprotokoll, das ursprünglich für die Übertragung von Hypertextdokumenten vorgesehen war (Dokumente, die möglicherweise Links enthalten, mit denen Sie den Übergang zu anderen Dokumenten organisieren können). Die Basis dieses Protokolls ist eine Anforderung von einem Client (Browser) an einen Server und eine Serverantwort an einen Client.

JSON

JSON (JavaScript Object Notation) ist ein universelles Format für den Datenaustausch zwischen einem Client und einem Server. Es ist eine einfache Zeichenfolge, die in jeder Programmiersprache verwendet werden kann.

PORT

Dies ist ein optionaler Teil der URL, der die Portnummer angibt, die der Zielwebserver abhört. Die Standardportnummer für HTTP-Server ist 80, einige Konfigurationen sind jedoch so eingerichtet, dass sie eine alternative Portnummer verwenden. In diesem Fall muss diese Nummer in der URL angegeben werden. Die Portnummer wird direkt mit einem Doppelpunkt, der unmittelbar auf den Servernamen oder die Adresse folgt, eingegeben.[RR03, p.31]

RFID

RFID (englisch radio-frequency identification oder „Identifizierung mit Hilfe elektromagnetischer Wellen“) bezeichnet eine Technologie für Sender-Empfänger-Systeme

und wird bei der Abschlussarbeit verwendet, um die vorhandenen zur Ausleihe Raspberry Pi Boards zu markieren und identifizieren.

URI

Uniform Resource Identifier ist ein Pfad zu einer bestimmten Ressource (z.B. einem Dokument), für die eine Operation ausgeführt werden muss (z. B. bei Verwendung der GET-Methode bedeutet dies das Abrufen einer Ressource). Einige Anforderungen beziehen sich möglicherweise nicht auf eine Ressource und in diesem Fall kann der Startzeile anstelle des URI ein Sternchen (Symbol "*") hinzugefügt werden.

Literaturverzeichnis

- [Ado] Adobe. *Understand web applications*. <https://helpx.adobe.com/dreamweaver/user-guide.html/dreamweaver/using/web-applications.ug.html>. abgerufen am 29. September 2020.
- [Chi14] Ugo Chirico. *Smart Card Programming. A comprehensive guide to smart card programming*. lulu.com London, 2014. ISBN: 978-1-291-61050-5.
- [cona] MDN contributors. *Django Tutorial Part 10: Testing a Django web application*. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing>. abgerufen am 12. November 2020.
- [conb] MDN contributors. *Introduction to the DOM*. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. abgerufen am 7. November 2020.
- [conc] Net informations.com contributors. *JavaScript: Client side Vs. Server side*. <http://net-informations.com/js/iq/side.htm>. abgerufen am 8. November 2020.
- [CR] David Corcoran and Ludovic Rousseau. *pcscd(8) - Linux man page*. <https://linux.die.net/man/8/pcscd>. abgerufen am 05. Oktober 2020.
- [Dix] Shubhang Dixit. *Beginners Guide to Client Server Communication*. <https://medium.com/@subhangdxt/beginners-guide-to-client-server-communication-8099cf0ac3af>. abgerufen am 5. November 2020.
- [Edi] Perdita Stevens. School of Informatics. University of Edinburgh. *Class diagrams and architectural design*. <http://www.inf.ed.ac.uk/teaching/courses/inf2c/lectures/SE/architectureclassHO.pdf>. abgerufen am 17. Oktober 2020.
- [Foua] Django Software Foundation. *The Django admin site*. <https://docs.djangoproject.com/en/3.1/ref/contrib/admin/>. abgerufen am 2. November 2020.
- [Foub] Django Software Foundation. *Writing your first Django app, part 1*. <https://docs.djangoproject.com/en/3.1/intro/tutorial01>. abgerufen am 12. Oktober 2020.

- [Fouc] Django Software Foundation. *Writing your first Django app, part 2.* <https://docs.djangoproject.com/en/3.1/intro/tutorial02>. abgerufen am 13. Oktober 2020.
- [Foud] Django Software Foundation. *Writing your first Django app, part 3.* <https://docs.djangoproject.com/en/3.1/intro/tutorial03>. abgerufen am 29. Oktober 2020.
- [Geo17] Nigel George. *Mastering Django: Core. The Complete Guide to Django 1.8 LTS.* Leanpub book, 2017. ISBN: 978-0-9946168-0-7.
- [Hal19] Gareth Halfacree. *The Official Raspberry Pi. How to use your new computer. Beginner's Guide.* raspberry Pi Press; 3rd Revised edition, 2019. ISBN: 1912047586.
- [ION] IONOS. *Komponentendiagramm – Effiziente Systemmodellierung mit Softwaremodulen.* <https://www.ionos.de/digitalguide/websites/webentwicklung/komponentendiagramm>. abgerufen am 18. Oktober 2020.
- [JI] Ralph Jacobi and Josh Wyatt. Texas Instruments. *MIFAREDESFireEV1 AES Authentication With TRF7970A.* <https://www.ti.com/lit/an/sloa213/sloa213.pdf>. abgerufen am 07. Oktober 2020.
- [lib] libusb. *libusb-1.0 API Reference. A cross-platform user library to access USB devices.* <http://libusb.sourceforge.net/api-1.0>. abgerufen am 05. Oktober 2020.
- [Ltd] Advanced Card Systems Ltd. *ACR122U USB NFC Reader.* <https://www.acs.com.hk/en/products/3/acr122u-usb-nfc-reader/>. abgerufen am 23. Oktober 2020.
- [MD] Space Telescope Science Institute. Michael Droettboom. *Understanding JSON Schema.* <https://json-schema.org/understanding-json-schema/reference/array.html>. abgerufen am 10. Oktober 2020.
- [ML] a Salesforce company MuleSoft LLC. *What is a REST API.* <https://www.mulesoft.com/resources/api/what-is-rest-api-design>. abgerufen am 5. November 2020.
- [NXP] NXP. *MIFARE® contactless tag IC family overview.* <https://www.nxp.com/docs/en/product-selector-guide/MIFARE-ICs-linecard.pdf>. abgerufen am 06. Oktober 2020.
- [OG] One Blog: it is all in the name. One Guy. *ACR122U NFC USB READER ON A RASPBERRY PI.* <https://oneguyoneblog.com/2015/09/16/acr122u-nfc-usb-reader-raspberry-pi/>. abgerufen am 05. Oktober 2020.
- [Pod] Mikhail Podgurskiy. *Django friendly finite state machine support.* <https://github.com/viewflow/django-fsm>. abgerufen am 28. Oktober 2020.

- [Pri] Author: SV 2d/3d Printing. *Raspberry Pi programming basics*. <https://api-2d3d-cad.com/microcomputer-programming-basics/>. abgerufen am 03. Oktober 2020.
- [RF] J. Gettys J. Mogul H. Frystyk T. Berners-Lee R. Fielding UC Irvine. *Hypertext Transfer Protocol – HTTP/1.1*. <https://tools.ietf.org/html/rfc1945>. abgerufen am 30. Oktober 2020.
- [RR03] Leon Shklar und Richard Rosen. *Web Application Architecture. Principles, protocols and practices*. John Wiley & Sons Ltd, 2003. ISBN: 0-471-48656-6.
- [SCT] Reiner SCT. *cyberJack® RFID basis. Chipkartenleser für den neuen Personalausweis*. <https://shop.reiner-sct.com/chipkartenleser-fuer-die-sicherheitsklasse-3/cyberjack-rfid-basis>. abgerufen am 04. Oktober 2020.
- [SSG] Sparx Systems Ltd und SparxSystems Software GmbH. *Anwendungsfalldiagramm (Use Case Diagram)*. <https://www.sparxsystems.de/ressourcen/literatur/leseprobe-zu-projektabwicklung-mit-uml-und-enterprise-architect/anwendungsfalldiagramm-use-case-diagram>. abgerufen am 18. Oktober 2020.
- [TBa] Beuth Hochschule für Technik Berlin. *acaLab: Aktivitäten und Abschlussarbeiten*. <https://labor.beuth-hochschule.de/pse/acalab-aktivitaeten-und-abschlussarbeiten>. abgerufen am 21. August 2020.
- [TBb] Beuth Hochschule für Technik Berlin. *Campus-Card*. <https://www.beuth-hochschule.de/campus-card>. abgerufen am 08. Oktober 2020.
- [TBC] PSE-Labor der Beuth Hochschule für Technik Berlin. *acaLoan-Raspi*. <https://labor.beuth-hochschule.de/pse/acalab-aktivitaeten-und-abschlussarbeiten/acaloan-raspi/>. abgerufen am 15. Oktober 2020.
- [TBL] H. Frystyk T. Berners-Lee R. Fielding. *Hypertext Transfer Protocol – HTTP/1.0*. <https://tools.ietf.org/html/rfc1945>. abgerufen am 30. Oktober 2020.
- [Tec] Syrma Technology. *How Radio-Frequency Identification Systems Work*. <https://www.syrmatech.com/rfid-system/>. abgerufen am 09. Oktober 2020.
- [Too] NFC Tools. *ISO/IEC 14443 Type A*. <http://nfc-tools.org/index.php/ISO14443A>. abgerufen am 07. Oktober 2020.