

Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt
Fakultät Informatik und Wirtschaftsinformatik

Dokumentation

StoreMe

**vorgelegt an der Hochschule für angewandte Wissenschaften
Würzburg-Schweinfurt in der Fakultät Informatik und Wirtschaftsinformatik zur
Dokumentation des Projekts „StoreMe“ innerhalb des Moduls
Programmierprojekt**

Florian Beuschel, Christian Braun, Marcel Groß,
Christian Paling, Marvin Therolf, Marcel Waleska

Eingereicht am: 17.07.2015

Inhaltsverzeichnis

1	Einleitung	1
1.1	Danksagung	1
1.2	Ausgangssituation und Motivation	2
1.3	Marktanalyse	2
1.3.1	Zielgruppe	2
1.3.2	Alternative Software	3
1.4	Vorgehen	3
1.4.1	Projektplanung	4
1.4.2	Aufgabenverteilung	4
1.4.3	Projektverlauf	5
1.4.3.1	Einarbeitungsphase	5
1.4.3.2	Erprobungsphase	5
1.4.3.3	Implementierungsphase	6
1.4.3.4	Dokumentationsphase	6
1.4.4	Erkenntnisse	6
2	Grundlagen	8
2.1	Fachliche Anforderungen	8
2.2	Verwendete Technologien und Frameworks	11
2.2.1	HTML5	11
2.2.2	CSS3	11
2.2.3	LESS	12
2.2.4	JavaScript	12
2.2.5	jQuery	13
2.2.6	Bootstrap	13
2.2.7	Node.js	14
2.2.8	CouchDB	14
2.2.9	Ractive.js	15
2.2.10	JSDocs	16
2.2.11	Android	16
2.3	Tools	16
2.3.1	JIRA	16
2.3.2	Stash (Git)	17
2.3.3	Webstorm	17
2.3.4	Android Studio	18

2.3.5	Google Drive	18
2.3.6	Google Docs	18
2.3.7	Draw IO	18
2.3.8	LaTeX	19
3	Architektur	20
3.1	Technische Anforderungen	20
3.1.1	UML-Diagramme	21
3.2	Kommunikationsmodell	21
3.3	Problemstellung Datenmodell	21
4	Implementierung	24
4.1	Konzept	24
4.2	Front End	26
4.2.1	Web Client	26
4.2.2	Android Applikation	30
4.3	Back End	34
4.3.1	Authentifizierung	34
4.3.1.1	Registrierung	34
4.3.1.2	Login	35
4.3.2	Datenstruktur	35
4.3.3	Datenbank-Verbindung	36
4.4	Sicherheit	36
4.5	Session IDs	36
4.6	Verschlüsselung	37
4.7	Testing	37
5	Evaluation	39
5.1	Das kann StoreMe	39
5.2	Das kann StoreMe nicht	39
6	Fazit	41
7	Ausblick	43
7.1	Problembehandlung	43
7.1.1	Container editieren	43
7.1.2	Mehrbenutzerbetrieb	43
7.1.3	Benutzerverwaltung	44
7.1.4	Performance	44
7.1.5	Constrains	44
7.2	Features	44
7.2.1	Inventur	44
7.2.2	Lagerplatzvorschlag	45
7.2.3	Rechteverwaltung	45

7.2.4	Alertsystem für Administratoren	45
8	Tutorial	46
8.1	Installation	46
8.1.1	StoreMe	46
8.1.2	Node.js Server	46
8.1.3	CouchDB	47
8.1.4	Android Applikation	48
8.2	Bedienung	48
8.2.1	Login und Signup	48
8.2.2	Das Dashboard	49
8.2.3	Management Ansicht	49
8.2.4	Container hinzufügen und löschen	50
8.2.5	Stammdaten anlegen, bearbeiten und löschen	50
8.2.6	Items einlagern und auslagern	51
8.2.7	Nach Items suchen	51
	Verzeichnisse	53
	Listings	55
	Literatur	56

1 Einleitung

Im Folgenden möchten wir Ihnen StoreMe vorstellen.

Wir, das sind Florian Beuschel, Christian Braun, Marcel Groß, Christian Paling, Marvin Therolf und Marcel Waleska.

Und StoreMe? StoreMe ist unsere universell einsetzbare und intuitiv verständliche Lösung für die Verwaltung Ihrer Lagerorte. Dabei spielt es keine Rolle, ob Sie Werkzeuge und Material in Ihrer Hobbywerkstatt, Warenpaletten im Hochregallager Ihrer Firma oder Joghurtbecher in Ihrem Kühlschrank organisieren möchten. Im Grunde ist man auch nicht auf die Standard-Einlagerungsfälle beschränkt. Für StoreMe ist es kein Problem Mitarbeiter in einem Unternehmen “einzulagern”. StoreMe bietet die notwendigen Feature Ihr Lager so realitätsnah wie möglich abzubilden und ist dabei in Design und Bedienung ansprechend, übersichtlich und funktional konzipiert.

1.1 Danksagung

An [3] dieser Stelle möchten wir unseren besonderen Dank gegenüber den Menschen Ausdruck bringen, die uns während der Planung und Durchführung der Projekts beraten und unterstützt haben. Wir bedanken uns bei Herrn Prof. Dr. Frank Hennermann für die umfassende Beratung zum Thema Lagerverwaltung und -logistik. Außerdem bedanken wir uns bei Herrn Vitaliy Schreibmann, der uns als Betreuer im gesamten Projektverlauf mit Rat und Tat zur Seite stand, sodass wir in der Lage waren, StoreMe in einer konstruktiven und produktiven Teamkultur zu entwickeln. Besonders bedanken möchten wir uns bei Herrn Patrick Mennig, der zu jeder Zeit ein offenes Ohr für unsere Fragen bezüglich JavaScript hatte und nicht müde wurde sein Wissen über die Entwicklung von Webanwendungen mit uns zu teilen. Ohne diese drei Personen wäre die Umsetzung des Projekts nicht möglich gewesen.

1.2 Ausgangssituation und Motivation

Im Rahmen der Veranstaltung “Programmierprojekt” war es unsere Aufgabe eine Softwarelösung zu konzipieren, zu implementieren und den Prozess sowie die Software selbst zu dokumentieren. Bei der Themenfindung und -ausarbeitung wurde uns dabei grundsätzlich freie Hand gelassen. Zur Bestimmung einer Projektidee hat jeder von uns ein bis zwei Projektideen vorbereitet und im Team vorgestellt. Anschließend haben wir uns in zwei offenen Wahlgängen für das Thema Lagerverwaltung entschieden. Unser Grundsatz dabei war, dass die Software unabhängig von der Größe des Lagers anwendbar sein sollte. Ein Privatanwender sollte nicht gezwungen werden, sich mit umfangreichen Detailkonfigurationen aufzuhalten. Professionelle Anwender hingegen sollten nicht auf wichtige Funktionalitäten verzichten müssen. Mit diesen Grundüberlegungen sind wir in die Projektplanung eingestiegen.

1.3 Marktanalyse

Auf dem Markt gibt es zwei verschiedene Arten von “Lager”-Software. Einerseits die Lagerverwaltungssoftware, in dessen Sparte auch StoreMe treten will, bei der es sich im engeren Sinne um eine Lagerbestandsverwaltungssoftware handelt. Andererseits gibt es Warehouse Management Systeme, die eher für Steuerung, Kontrolle, Optimierung komplexer Lager- und Distributionssysteme, Disposition und viele weitere Aufgaben gedacht sind. Hinter Warehouse Management Systeme (nachfolgend WMS genannt) steckt demzufolge eine komplexe Logik zur Optimierung und Führung von Lagern. Einer der bekanntesten Vermarkter von WMS ist die Firma SAP. Da WMS aufgrund ihrer Aufgabenbereiche und darausfolgend großen Komplexität eher für große Konzerne in Frage kommen, werden wir uns nachfolgend mit Lagerverwaltungssoftware beschäftigen.

Sucht man nach einer passenden Lagerverwaltungssoftware, wird man schnell fündig mit passenden Plug-ins für bestehende Softwareprodukte (z.B. für Microsoft Access) oder Testversionen für Standalone Programme. Schaut man sich die Leistungsmerkmale eben jener Lagerverwaltungssoftware an, bemerkt man schnell diverse Einschränkungen oder Barrieren die wir bei der Entwicklung von StoreMe möglichst vermeiden wollen. Eine Beispielanalyse ist im Bereich 1.3.2 Alternative Software zu finden.

1.3.1 Zielgruppe

Unsere Software richtet sich vor allem an Privatpersonen und kleinere Unternehmen, die eine leicht zu bedienende Software benötigen um ihre Lagersituation möglichst

wirklichkeitsgetreu digital abzubilden.

1.3.2 Alternative Software

Um die Marktanalyse nochmals zu verdeutlichen, werden wir anhand von einem Konkurrenzprodukt[2] die Unterschiede aufzeigen.

Sage

- Sage bietet eine kostenpflichtige Lösung zur Verwaltung eines Lagers.
- Es besteht auch die Möglichkeit die Software zu mieten. Hier wird monatlich pro Anwender bezahlt.
- Durch ein zusätzliches Paket ermöglicht die Software dem Benutzer die Verwaltung in verschiedenen Lagertechniken wie z.B. Hochregallager, Flächenlager, usw.
- Sage verfügt über die Möglichkeit eigene Lager anzulegen.
- Desweiteren besitzt die Software eine Funktion für die Inventur.
- Durch viele Zusatzfeatures, die unsere Zielgruppe nicht benötigt ist die Bedienung der Software sehr gewöhnungsbedürftig und nicht intuitiv.
- Auch das Design hält sich sehr an Microsoft Produkte und ist nicht sonderlich ansprechend.

1.4 Vorgehen

In diesem Kapitel möchten wir erläutern, wie wir den Projektauftrag Schritt für Schritt umgesetzt haben. Dabei wollen wir zunächst auf die Planung, anschließend auf die Umsetzung eingehen, die Aufgabenteilung, sowie unsere persönlichen Erkenntnisse beschreiben.

1.4.1 Projektplanung

In einer ersten Phase haben wir uns ausführlich damit beschäftigt, wie wir die Zeit, die uns für die Bearbeitung zur Verfügung steht, sinnvoll aufteilen, welche Arbeitsgruppen zu bilden sind und welche Technologien sich zur Implementierung anbieten. Um einen Überblick über die Fachthematik zu bekommen, haben wir uns an Professor Hennermann gewandt. Wir haben uns dafür entschieden SCRUM als Entwicklungsmodell einzusetzen. An SCRUM gefällt uns vor allem, dass man auch während der Entwicklung sehr flexibel gegenüber Anforderungsänderungen bleibt. Als uneingespieltes 6er-Team ohne Projekterfahrung erschien uns agiles Vorgehen als die einzige sinnvolle Variante. Nach der Ideenfindung stand die Frage im Raum, was wir als Teilnehmer von der gemeinsamen Arbeit am Projekt erwarten. Jeder von uns hat Stellung dazu genommen, ob er lieber mit bekannten Technologien (Java, Tomcat, MySQL) arbeiten möchte um bereits Gelerntes im Rahmen einer größeren Anwendung zu vertiefen, oder ob er mit neuen Technologien arbeiten will um seinen Horizont zu erweitern. Wir haben uns einstimmig für die zweite Variante entschieden. In Absprache mit Herrn Schreibmann und Herrn Mennig haben wir uns darauf geeinigt, dass eine Webanwendung unseren Anforderungen entspricht. Vorteilhaft an Webanwendungen ist, dass sie auf allen Geräten und Betriebssystemen, die über einen Internetbrowser verfügen, ausgeführt werden können. Nachteil: Keiner von uns hatte Erfahrung mit Webentwicklung. Wie bereits erwähnt, haben wir diesen vermeidlichen Nachteil bewusst in Kauf genommen um uns in Richtung neuer Technologien und Frameworks weiterzubilden. Mit dieser Einstellung haben wir uns letztlich für den Einsatz von JavaScript, einem Node.js Server sowie einer CouchDB entschieden. Näheres dazu im Kapitel 2.2. Im Anschluss an die Technologieauswahl ging es an die konkrete Planung. Wir haben die fachlichen und die sich daraus ergebenden Anforderungen zusammengetragen und daraus einen Backlog für die Funktionalität einer Prototyp-Version erstellt. Weiterhin machten wir uns Gedanken über das Design und die Frontend-Backend-Kommunikation. In dieser Phase bekam das Projekt auch seinen Namen: StoreMe. Schließlich wurden die Aufgaben verteilt und der erste Sprint begonnen. Thema: Einarbeitung in JavaScript, Suchen nützlicher Frameworks und Gewöhnung an JIRA. Von diesem Punkt aus haben wir die Phase der Planung verlassen und den Rest des Projektes im agilen SCRUM-Prozess durchgeführt.

1.4.2 Aufgabenverteilung

Unser Ansatz bei der Bearbeitung des Projekts war es, möglichst viel neues zu lernen. Selbstverständlich gab es zu Beginn eine grobe Aufgabenverteilung. Aufgrund der Tatsache, dass wir den Umgang mit den neuen Technologien erst während des Projektes erlernt haben, hat sich die Realität der Aufgabenverteilung immer wieder verändert. So haben sich immer wieder kleine Arbeitsgruppen gebildet um gemeinsam an Problemen zu arbeiten. Dabei hat jeder von uns selbständig darauf geachtet, Einblick in alle für ihn

1 Einleitung

interessanten Teilbereiche zu nehmen. Die Zuordnung der Hauptverantwortung für die einzelnen Teilbereiche sah wie folgt aus:

Frontend: Christian Braun, Christian Paling
Programmlogik: Marvin Therolf, Marcel Groß
Backend (Node): Marcel Groß
Datenbank-Einrichtung: Florian Beuschel
Datenbank-Zugriff: Marcel Waleska
Android-App: Marcel Groß
Product-Owner: Marvin Therolf
Scrum Master: Christian Braun
Testing: Alle
Hauptverantwortlicher Dokumentation: Florian Beuschel
Dokumentation: Alle

1.4.3 Projektverlauf

Der Projektverlauf lässt sich grob in vier Phasen einteilen: Einarbeitung, Erprobung, Implementierung und Dokumentation.

1.4.3.1 Einarbeitungsphase

Ca. 2 Wochen nach dem Projektstart begann für uns die Einarbeitungsphase. Hier galt es, sich mit den uns unbekannten Technologien vertraut zu machen. Keiner von uns hatte viel Erfahrung mit der Entwicklung von Webanwendungen. So haben wir damit begonnen uns in HTML, JavaScript und jQuery einzuarbeiten. Sehr hilfreich waren dabei die Tutorials auf Codecademy.com [6]. Gleichzeitig haben wir uns mit Themen wie QR-Code-Generierung, NoSQL-Datenbanken und Serverlösungen beschäftigt. Die Einarbeitung hat ungefähr drei Wochen Zeit in Anspruch genommen.

1.4.3.2 Erprobungsphase

Die Erprobungsphase hatte den Zweck, das Team auf einen Stand in Sachen Technologieverständnis zu bringen. In dieser Zeit wurden zu den meisten verwendeten Technologien und für die wichtigsten Anwendungsfälle kleine Beispielapplikationen erstellt und im Team präsentiert. So haben wir gelernt z.B. gelernt, wie man JavaScript-Files in einer

HTML-Seite miteinander kombiniert, wie man mit Bootstrap das Design einer HTML-Seite bearbeiten kann oder wie man Daten in eine CouchDB schreibt und diese wieder auslesen kann. Für der Erprobung der neu gelernten Inhalte haben wir uns weitere drei Wochen Zeit genommen.

1.4.3.3 Implementierungsphase

Während der fünfwöchigen Implementierungsphase haben wir StoreMe entwickelt. Parallel wurden Front- und Backend aufgesetzt, die Programmlogik entworfen und codiert. In dieser Zeit haben wir JIRA mehr und mehr als Tool zur schnellen Kommunikation schätzen gelernt. Wir standen im engen Kontakt zu Herrn Schreibmann und Herrn Mennig. Das JIRA-Board bot uns einen guten Überblick über Arbeitspakete die noch zu erledigen sind. Häppchenweise konnte sich jeder die Aufgaben herausuchen, die ihn interessiert haben. Fehler und neue Anforderungen konnte in Form von Vorgängen schnell an die betroffenen Entwickler durchgereicht werden. Was wir nicht über JIRA kommunizieren konnten, haben wir mit einem geteilten Google Drive Ordner und einer WhatsApp-Gruppe organisiert. Während der Implementierung gab es immer wieder Diskussionen darüber ob bereits geplante Inhalte in der vereinbarten Form umsetzbar sind. Wir haben erst im Laufe dieser Zeit gelernt, was es wirklich bedeutet agil zu entwickeln. Neben der Entwicklung stand auch das Testen neu implementierter Methoden auf dem Programm. Im Bereich der Datenstruktur ist es uns sogar gelungen einen überwiegend testgetriebenen Entwicklungsstil an den Tag zu legen.

1.4.3.4 Dokumentationsphase

In der letzten Juniwoche haben wir uns - neben dem Verbessern von Fehlern - hauptsächlich der Dokumentation zugewandt. Die meisten Entwicklungsschritte haben wir bereits während der vorherigen Phasen ausführlich dokumentiert. Ziel war es nun, alle Informationen in einem Dokument zusammenzutragen, den Text in eine einheitliche Form zu bringen und ansprechend aufzubereiten. Die grobe Gliederung haben wir von Herrn Schreibmann erhalten, die Detailgliederung, in Absprache mit ihm, eigenständig erstellt.

1.4.4 Erkenntnisse

Die Liste der Erkenntnisse, die wir während des Programmierprojektes gewinnen konnten ist wohl endlos lang. Sie beginnt mit der Tatsache, dass ein agiles Entwicklungsmodell tatsächlich funktioniert und wesentlich weniger inhaltliche Vorausplanung erfordert als

1 Einleitung

man intuitiv vermuten würde. Eine weitere Erkenntnis liegt darin, dass eine gute Kommunikation, insbesondere bei Konflikten, der einzige Weg ist um ein frustfreies gemeinsames Arbeiten zu ermöglichen. Wir haben gelernt, dass man bei der Entwicklung im Web immer wieder auf die gleichen Probleme stößt. Findet ein regelmäßiger Austausch zwischen den Teammitgliedern statt, können diese von den Lösungen der anderen profitieren. Man macht die selben Fehler nicht zwei Mal. Dazu kommen natürlich persönliche Erkenntnisse. Jeder von uns hat erfahren, wie es ist an guten, aber auch an schlechten Tagen als Team zu arbeiten, Konflikte auszutragen, undankbare Aufgaben zu übernehmen und sich zusammen zu reißen. Wenn wir beim nächsten Mal etwas grundsätzlich anders machen würden, dann würden wir von Anfang an wirklich agil entwickeln und uns Tools wie JIRA noch mehr zunutze machen. Tatsächlich waren zu starre Planungen in der Anfangsphase letztlich unsere größte Hürde bei der Entwicklung.

2 Grundlagen

Die Beschreibung der Grundlagen ist wichtig, um die fachlichen Hintergründe, die hinter der Entwicklung von StoreMe stehen, zu begreifen. Zu den Grundlagen gehört außerdem die Beschreibung der von uns verwendeten Technologien, Frameworks und Tools. Wir werden dabei auf alle Aspekte eingehen, die aus Sicht unseres bisherigen Studienverlaufs als “neue Inhalte” zu bezeichnen sind.

2.1 Fachliche Anforderungen

Die Kernanforderung an StoreMe war die Idee eines universellen Lagers. Universell bezieht sich hierbei auf die Möglichkeit jedes beliebige Item zu erstellen, jede erdenkliche Lagerstruktur abzubilden und jedes erstellte Item in besagte Lagerstruktur zu integrieren. Zusätzlich soll der Nutzer seine Lager geräteübergreifend kontrollieren können. Ein universelles Lagersystem sollte also Lagerzugriff und -bearbeitung von mehreren Geräten gleichzeitig erlauben. Dabei dürfen Bildschirmauflösung und Betriebssystem der genutzten Geräte keine Rolle spielen. Besonders ist zu erwähnen, dass wir kein Lagersystem entwickeln wollen, welches dem Nutzer in irgendeiner Form einschränkt oder Bedingungen vorschreibt. Der Nutzer hat zu jedem Zeitpunkt die volle Kontrolle.

Die Anforderungen zum Erstellen eines Items sollen bei StoreMe über einen Stammdaten-Screen abgebildet werden. Dabei muss jedes Item mindestens eine ID und einen Namen tragen. Um jeden beliebigen Gegenstand abbilden zu können, ist der Nutzer in der Lage einem Item eine unbestimmte Anzahl zusätzlicher beliebiger Attribute hinzuzufügen. Ein solches besteht aus einem Attributnamen, einem Attributwert, einer Einheit und einem Attributtypen. Dieser kann entweder vom Typ Boolean oder Number sein und ist für die zum Einlagern erwünschte Constraint-Prüfung essentiell. Um bereits erstellte Attribute nicht in mehreren unterschiedlichen Schreibweisen in der Datenbank mitzuführen, ist beim Tippen des Attributnamens eine Vorschlagsliste bekannter Attribute erstrebenswert. Wenn erwünscht, soll einem Item auch noch eine vorher erstellte Kategorie zugewiesen werden können. Dadurch lässt sich zum einen eine filigrane Item-Suche als auch eine bessere Übersicht über Item-Gruppen verwirklichen. Nach dem Erschaffen eines Items müssen Item-Name und Attributwerte nachbearbeitet und bei Bedarf auch das gesamte Item gelöscht werden können. Eine weitere fundamentale Funktion des Stammdaten-Screens ist

2 Grundlagen

Attributname	Attributwert	Attributeinheit	Attributtyp
Gewicht	300	kg	Number

Tabelle 2.1: Nummern Attribute

Boolean-Attribut			
Attributname	Attributwert	Attributeinheit	Attributtyp
Flüssig	true	-	Boolean

Tabelle 2.2: Boolean Attribute

das Erstellen von den Items zugehörigen QR-Codes. Nur so kann ein schnelles Einlagern und Auslagern der Items sinnvoll realisiert werden.

Um auch bei der Lagererstellung einen möglichst abstrakten Weg zu gehen nehmen wir von einer spezifischen Bezeichnung von Regalen und Lagerräumen Abstand. Jeder Behälter wird in StoreMe als Container bezeichnet. Er sollte eine selbständig generierte ContainerID einen Namen und analog zu den Stammdaten eine dynamische Liste freiwählbarer Attribute besitzen. Grundsätzlich soll jeder Container wieder eine beliebige Anzahl an Containern und Items beherbergen können. So soll eine Verschachtelte Struktur ähnlich zu einer Matroschka realisiert werden können. Das explizite Verbieten von speziellen Items oder Containern soll über eine Compulsory-Attribut-Markierung erfolgen. Diese Attribute werden im Folgenden als Compulsory-Attribute bezeichnet. Jeder Container oder jedes Item welches unter oder innerhalb eines Containers mit einem oder mehreren Compulsory-Attributen liegt muss diese Attribute als ein eigenes tragen. Ist ein Compulsory-Attribut vom Typ Number, so soll geprüft werden ob die Summe des Attributs aller Container, bereits eingelagerten Items und der einzulagernden Items innerhalb des angegebenen Wertes liegt. Bei einem Compulsory-Attribut vom Typ Boolean muss jedes einzulagernde Item besagtes Attribut mit dem gleichen Wert halten. Um die Nachbildung eines Lagers weiter zu vereinfachen soll der Nutzer im Stande sein mehrere Container gleichzeitig zu erstellen oder zu löschen.

Hier können nur noch Container und Items eingelagert werden solange die Summe ihres Gewichtes unterhalb oder genau 300 Kilogramm beträgt.

Hier können nur noch Items eingelagert werden welche das Attribut "flüssig" mit dem Wert "true" tragen.

Attributname	Attributwert	Attributeinheit	Attributtyp
Gewicht	300	kg	Number

Tabelle 2.3: Number-Compulsory-Attribute

Attributname	Attributwert	Attributeinheit	Attributtyp
Flüssig	true	-	Boolean

Tabelle 2.4: Boolean-Compulsory-Attribute

Durch die Verwaltung von Stammdaten und das Abbilden der Lagerstruktur in in-sich verschachtelte Container ist der Grundstein für das Einlagern und Auslagern von Items gelegt. Items sollen auf zwei Arten ein- bzw. ausgelagert werden können. Entweder navigiert der Nutzer auf die entsprechende Container-Ebene oder es werden Container- und Item-ID durch eine native Android-Anwendung eingescannt. Das aufeinander folgende Scannen der Container- bzw. Item-ID gibt einen realistischen Vorgang in einem Lager wieder. Zusätzlich soll bei jedem Einlagerungsvorgang ein optimaler Lagerplatz angeboten werden. Dieser soll entweder über zuvor eingelagerte Items des gleichen Typs oder über die Attribute des Items ermittelt werden. Wählt der Nutzer selbst einen Container für die Einlagerung des Items aus, so soll er bei der Verletzung von Compulsory-Attributen gewarnt werden. Möchte er dennoch einlagern wird er jedoch nicht daran gehindert.

Für die Verwaltung und den Überblick über den Lagerstand ist ein Lagerspiegel erforderlich. Hier sollen alle Items eines Lagers aufgelistet werden. Der Nutzer soll nach Name, ID, Kategorie und allen Attributen der Items suchen können. Eine zusätzliche Filterung soll das Aussondern von Items mit speziellen Attributen ermöglichen. Da die Funktionalität eines Lagerspiegels der einer Inventur schon sehr Nahe kommt, soll auch diese hier realisiert werden. Genauso wichtig wie eine detaillierte Übersicht über den Lagerbestand ist die Aufstellung aller Ein- und Auslagerungen innerhalb eines Lagers und wer diese getätigt hat. Auch hier soll StoreMe dem Nutzer einen geordneten Überblick über Mitarbeiteraktionen liefern.

Zusätzlich zu allen lagerspezifischen Funktionen sollte ein Lagerverwaltungssystem administrative Funktionen implementieren. Hierzugehört das Anlegen oder Genehmigen von Mitarbeitern zu einem spezifischen Lager, so wie eine feinstufige Rechteverwaltung. Hier soll eingegrenzt werden welche Nutzer nur Ein- und Auslagern, welche die Lagerstruktur bearbeiten oder welche die Stammdatenverwaltung verändern können. Zusätzlich sollte Sicherheit in Bezug auf unternehmensinterne Daten bereitgestellt werden. Da dieses Feld nahezu endlos ist, ist für StoreMe nur ein zeitlich begrenzter Login relevant.

2.2 Verwendete Technologien und Frameworks

2.2.1 HTML5

HTML (Hypertext Markup Language) ist eine Markup Sprache um Inhalte elektronischer Dokumente, vorwiegend im World Wide Web, zu erstellen und anzuzeigen. HTML5 spezifiziert die fünfte Fassung dieser Sprache und wurde 2014 von World Wide Web Consortium (W3C) fertiggestellt.

HTML5¹ erweitert HTML4 um Funktionalitäten wie:

- Darstellung von Audio, Video, 3D-Grafiken mit HTML Tags (vorher wurde dazu Adobe Flash benötigt)
- Canvas Element für Zeichnungen, Schatten, ...
- Die Strukturierung der Seite mit neuen HTML Tags wie nav, header, footer, ...

Zu welcher Version ein HTML Dokument gehört, wird mit DOCTYPE am Anfang des HTML Dokument festgelegt. In HTML5 wird das DOCTYPE durch ein einfaches

Listing 2.1: HTML Dokumentspezifizierung

```
1  <!DOCTYPE HTML>
```

spezifiziert, was auch am Anfang jeder HTML Seite von StoreMe zu finden ist. Ansonsten wird auf neue Funktionalitäten der HTML5-Spezifikation in StoreMe weitestgehend verzichtet.

2.2.2 CSS3

Cascading Style Sheets ist eine Layout-Sprache und wurde entwickelt vom World Wide Web Consortium (W3C). Zwar liefert HTML grundlegende Funktionalitäten um das Layout festzulegen, jedoch kommt HTML dabei schnell an seine Grenzen und es sollte darauf verzichtet werden.

CSS3 bietet einen riesigen Umfang an sogenannten Rules, denen ein Wert zugeordnet wird. Weiterhin kann eine Rule einem DOM-Element per Selektor zugeordnet werden. Dadurch kann man eine Website bis zum kleinsten Element nach belieben gestalten. CSS kommt

¹<http://www.w3.org/TR/html5/>

in StoreMe in Verwendung bei den eigenen Style Sheets Files für die unterschiedlichen HTML-Seiten von StoreMe.

Als Programmierer erscheint CSS nicht logisch und die Arbeit mit der Sprache gestaltet sich als äußerst unangenehm. Jedoch ist CSS eine Technologie, auf die bei einer Webplattform nicht verzichtet werden kann. Das reine CSS kann jedoch heutzutage außen vor bleiben und stattdessen ein Präprozessor wie LESS oder Sass verwendet werden, der die Arbeit mit CSS um einiges erleichtert.

2.2.3 LESS

LESS² ist ein, in Javascript geschriebener Präprozessor, welcher CSS um viele Funktionalitäten erweitert.

LESS erweitert CSS um Features wie:

- Variablen
- Eingebaute Funktionen wie saturate, desaturate, convert
- Geschachtelte CSS Rules
- Mathematische Operationen

Dadurch das Twitter Bootstrap in LESS geschrieben ist und StoreMe ein angepasstes Bootstrap verwendet, ist viel LESS Code in das Design von StoreMe eingeflossen. Heutzutage gibt es keinen Grund mehr auf das normale CSS zurückzugreifen. Mit CSS Präprozessoren wie LESS, Sass uvm. werden Stylesheets modularer, leichter wartbar und funktionaler. Der Code muss Compiliert werden. Durch LESS wurde die Arbeit am Design von StoreMe erheblich vereinfacht. Mithilfe von Variablen können beispielsweise Farbschema oder Schriftart von StoreMe mit wenigen Code-Änderungen komplett umgestellt werden. Für uns hat sich die Arbeit mit einem CSS Präprozessor wie LESS als unverzichtbar erwiesen.

2.2.4 JavaScript

JavaScript[6] ist eine Skriptsprache, welche ursprünglich für dynamische Inhalte in Webbrowsern entwickelt wurde. Heutzutage ist JavaScript jedoch in allen Bereichen der

²<http://lesscss.org>

Entwicklung zu finden. Ein großer Grund dessen ist die große Beliebtheit von Node.js. StoreMe selbst ist fast komplett in JavaScript implementiert. Das Frontend sowie das Backend setzt auf JavaScript, sowie die Speicherung der Daten, welche in JavaScript Object Notation (JSON) erfolgt. Dadurch ist die Kommunikation von Programmlogik zwischen den einzelnen Bereichen von StoreMe erleichtert worden. Für uns gab es bei der Arbeit mit JavaScript viel Begeisterung aber gleichzeitig auch Kopferbrechen. Auf der einen Seite gibt JavaScript dem Entwickler viele Freiheiten beim Schreiben des Codes und kommt mit einem großen Umfang an interessanten Funktionalitäten. Vieles in JavaScript wird vollkommen anders gelöst als in anderen heutzutage relevanten Programmier- und Skriptsprachen. Jedoch haben uns Dinge wie, das Debuggen, die Asynchronität oder das feste Call-By-Reference Prinzip oft Ärger beschert.

2.2.5 jQuery

jQuery [6] ist eine freie JavaScript Bibliothek, welche einfache Funktionen zur DOM-Manipulation in JavaScript bereitstellt. Dazu benutzt sie Selektoren, welche mit den Cascading Stylesheets übereinstimmt. Dadurch hat sich jQuery zu einer der am häufigsten benutzten JavaScript Bibliotheken etabliert. Während anfangs in StoreMe noch viel mit jQuery gearbeitet wurde, wurde über die Dauer des Projektes immer mehr auf die internen Funktionalitäten von Ractive.js gesetzt. Es erschien uns als keinen guten Stil jQuery Code mit Ractive.js Code zu mischen. jQuery selber ist eine hervorragende Bibliothek welche sich als unverzichtbar erwiesen hat.

2.2.6 Bootstrap

Bootstrap[3] ist ein Front-End Framework, welches vorgefertigte Cascading Stylesheet Klassen bietet um sein Front-End schnell und simpel aufzubauen. Dabei ist Bootstrap Responsive, wie beispielsweise das Grid-System. Im Grid-System kann der Entwickler seine Seite beliebig in Zeilen einteilen und Inhalte bestimmten Felder des “Grids” zuordnen. Diese Zeilen können je nach Bildschirmgröße unterschiedlich sein und erwirken dadurch, dass das Front-End Responsive wird. Auch in StoreMe wurde das Grid-System intensiv verwendet. Bootstrap wird von Twitter entwickelt und ist aktuell eines der meistgenutzten Projekte von Github (Hostingdienst für Repositories), u.a. wird Bootstrap von der NASA verwendet. Bootstrap ist modular aufgebaut und enthält Stylesheets die in LESS geschrieben sind (siehe auch LESS).

Generell wurde in StoreMe der Quellcode von Bootstrap angepasst um StoreMe ein eigenständiges Thema zu geben. Hierbei handelt es sich bei den Abänderungen größtenteils um Farb- und Textveränderungen.

2.2.7 Node.js

Node.js³ [8] ist eine leichtgewichtige auf JavaScript basierende Plattform für Netzwerkanwendungen. Dabei eignet es sich besonders gut für Webserver. Die Plattform arbeitet ereignisorientiert und in einem Thread. Das bedeutet, dass dadurch ein Webserver sehr leichtgewichtig wird und nicht viel Arbeitsspeicher benötigt. Jedoch stößt Node.js durch diesen Ansatz an seine Grenzen, falls umfangreiche Berechnungen durchgeführt werden müssen. Durch einen eingebauten Paketmanager (Node Package Manager, kurz: npm) können sich leicht weitere Module zur eigenen Node.js Plattform hinzufügen lassen. Seit der Veröffentlichung erfreut sich Node.js einer großen Beliebtheit, welches sich auch an der umfangreichen Anzahl der verfügbaren Module sehen lässt. Wir entschieden uns für Node.js, da die Plattform eine aktuelle Technologie ist und wir unsere Anwendung möglichst leichtgewichtig halten wollten. Weiterhin ermöglicht Node.js, dass auch das Backend von StoreMe in JavaScript geschrieben ist und dadurch die Kommunikation zwischen Back- und Frontend erleichtert wird.

2.2.8 CouchDB

CouchDB⁴ [1] [4] ist ein Datenbankmanagementsystem nach dem Vorbild einer dokumentenorientierter Datenbank. Sie wurde von Damien Katz entwickelt und in ERLANG geschrieben. Als Abfragesprache kommt Javascript zum Einsatz. Verwendet wird die Datenbank hauptsächlich für Webseiten, Facebook Applikationen und andere Softwareprojekte. CouchDB steht für “Cluster of unreliable commodity hardware Data Base” und bietet eine Schemalose Datenbankdarstellung, d.h sie verwendet keine Tabellen, Spalten und Zeilen sondern Dokumente die JSON Dateien beinhalten. Zugriffe auf die Datenbank erfolgen über eine HTTP-Schnittstelle. Abfragen werden von der Datenbank nach dem von Google entwickelten Map-Reduce Prinzip verarbeitet.

CouchDB bietet diverse Funktionalitäten:

- Futon als JavaScript Applikaton zur Nutzer- und Datenbankverwaltung
- CouchApps als Integriertes JavaScript Framework mit clientseitiger jQuery-Bibliothek und Erweiterungen
- Merge-Replikation zum Betreiben mehrerer verteilter CouchDB Instanzen

Wir verwenden CouchDB in der aktuellen Version 1.6.1.

³<https://nodejs.org/>

⁴<http://couchdb.apache.org/>

2.2.9 Ractive.js

Ractive.js⁵ [9] ist eine Javascript Bibliothek zur Entwicklung eines lebendigen, interaktiven Front-Ends. Ursprünglich wurde Ractive.js für die britischen Tageszeitung The Guardian entworfen. Mit diesem Hintergrund wurde die Bibliothek entwickelt um in möglichst kurzer Zeit interaktive und performante Oberflächen zu erstellen. Ractive.js hat sich im mobilen- und Desktop Bereich leistungsstark und zuverlässig verhalten. Ractive.js bindet Daten, welche als Objekt in einem Ractive-Element definiert sind, an die Oberfläche. Dies hat zur Folge, dass ein Update der Daten, ein sofortiges Update der Oberfläche mit sich führt.

Ractive.js bietet unter anderem folgende Funktionalitäten:

- on-click, on-tap (für mobile Geräte) oder on-change um Events an DOM-Elemente zu binden
- Animationen und sogenannte Transitions, mit der man das Erscheinen von Elementen auf der Oberfläche dynamisch gestalten kann
- Schleifen, welche über Arrays iterieren um beispielsweise eine datengebundene Tabelle zu erstellen
- Bedingungen, um beispielsweise das Anzeigen von DOM Elementen an Konditionen zu knüpfen
- Components für eine Strukturiertere HTML Seite die mehrfach verwendet werden können. Vorteile für die Modularität und Wartbarkeit

Der größte Vorteil von Ractive.js für StoreMe ist die Bindung der Daten an die Oberfläche. Dadurch wird mit einer Datenbankabfrage, welche die Daten aktualisiert, die Oberfläche automatisch neu gerendert. Weiterhin ist die Iterierung über Arrays an vielen Stellen im StoreMe Code von Nöten (z.B.: die Tabelle des Dashboards oder das Container Panel auf dem Main Screen). Nach unseren Erkenntnissen ist Ractive.js eine tolle Bibliothek für das Front-End, welche jedoch auch seine Grenzen hat. Möchte man beispielsweise eine größere Anzahl von Containern auf dem Main Screen erstellen, merkt man das Ractive.js beim Rendern der DOM an seine Grenzen kommt. Ob dies jedoch an der Javascript-Engine des Browsers liegt oder an der Bibliothek selbst können wir an dieser Stelle nicht beurteilen. Weiterhin ist das Einbinden von weiteren Javascript Plugins z.B. jQuery nicht immer trivial. Aufgrund des Renderns von Ractive.js kommt es gelegentlich beim Verwenden von Funktionalitäten anderer Plugins zu Konflikten.

⁵www.ractivejs.org

2.2.10 JSDocs

JSDocs [**jQueryDoc**] ist eine Markup Sprache die benutzt wird um JavaScript Quellcode zu kommentieren. Mithilfe dieser Annotationen können diverse Tools daraus Dokumentation im HTML- oder RTF-Format erstellt werden.

Solche Tags können wie folgend aussehen:

- @author
- @constructor
- @private

etc.

Wir haben JSDocs verwendet um die im Projekt verwendete Datenstruktur zu dokumentieren.

2.2.11 Android

Android⁶ ist ein Betriebssystem für smalldevices (Smartphones, Kameras, Tablets, Kleinrechner) und baut auf einen Linuxkernel auf. Da es keine Linux Distribution im klassischen Sinne ist gehört es somit zu den Embedded Linux Distributionen. Google kaufte 2003 die Firma die Android entwickelte auf und ist aktuell der Schirmherr in der Android Entwicklung (in der sog. Open Handset Alliance). Jedoch ist Android unter GPLv2 (Opensource) zertifiziert und kann von jedem benutzt und angepasst werden.

2.3 Tools

2.3.1 JIRA

JIRA ist eine webbasierte Software im Bereich des Projektmanagements, welche von Atlassian entwickelt wurde. Dabei unterstützt JIRA agile Vorgehensmodelle wie SCRUM und Kanban. Die Entwickler wollten sich an das bekannte BugZilla anlehnen und gaben ihm somit als Anlehnung den japanischen Namen Gojira (Godzilla). JIRA wurde von

⁶<https://www.android.com/>

uns verwendet um die Aufgaben zu zuteilen und die verschiedenen Bearbeitungsphasen, in der diese Aufgaben aktuell sind, ersichtlich zu machen. Dabei benutzen wir die Agile-Funktion von JIRA.

Zwar benötigt die Einrichtung und Eingewöhnung an JIRA eine gewisse Zeit, jedoch bietet JIRA eine große Unterstützung beim verteilten Arbeiten, sowie eine bessere Übersichtlichkeit, falls das Projekt eine gewisse Komplexität erreicht.

2.3.2 Stash (Git)

Stash ist eine webbasierte Git-Repository Software die in Java geschrieben wurde. Entwickler ist die australische Firma Atlassian, welche auch für JIRA verantwortlich ist. Stash verfügt u.a. über ein feingranulares Berechtigungssystem, das für die Projekt-, Repository- und Branchebene eingerichtet werden kann. Die Funktionalität von Stash kann mithilfe von Add-ons erweitert werden, welche auch selbst geschrieben werden können.

Obwohl Stash einer proprietären Lizenz unterliegt kann es u.a. für Schulklassen oder Open-Source Projekte kostenlos genutzt werden. Wir haben Stash als Online-Repository verwendet, um eine zentrale Stelle für unser Git-Repository zu besitzen auf welche das ganze Team zugreifen kann.

2.3.3 Webstorm

Webstorm⁷ ist ein IDE für Websprachen wie HTML, JavaScript, CSS und ähnliche Sprachen. Obwohl in Websprachen viele Facetten an unterschiedlichen Syntax existieren, funktioniert die Code Vervollständigung in Webstorm erstaunlich gut. Weiterhin besitzt Webstorm praktische Features, wie das Vorschlagen von Datentypen bei JavaScript Parametern und einen eigenen Server zum Ausprobieren der Applikation.

Die Arbeit mit Ractive.js und Webstorm erwies sich jedoch gelegentlich als etwas ärgerlich. Da ein Template in Ractive ein einfacher String ist, zeigt Webstorm gelegentlich Fehler im Code, obwohl diese nicht vorhanden sind. Dies ließ sich oft mit einem Zurücksetzen der Änderungen erst wieder beheben.

⁷<https://www.jetbrains.com/webstorm/>

2.3.4 Android Studio

Android Studio⁸ ist ein IDE für die Programmierung nativer Android Applikationen der Firma JetBrains. Google hat jeglichen Support für andere IDEs eingestellt. Auf der offiziellen Android Developer Seite wird primär Android Studio zum Download angeboten. Alle Tutorials und Beispiel Anwendungen, sowie deren Beschreibungen sind auf Android Studio ausgelegt. Ein weiterer Vorteil ist das man die Android SDK nicht extra in die IDE einbinden muss. Man muss diese lediglich in der IDE herunterladen. Android Studio bindet automatisch das Build-Tool Gradle in jedes Projekt mit ein. Wegen der geringen Bedeutung von Gradle in unserem Projekt werden wir hier nicht weiter darauf eingehen. Weitere Informationen zu Gradle finden Sie unter folgender Internetseite: <https://gradle.org/> .

2.3.5 Google Drive

Google Drive ist ein Webservice von Google, der einem Benutzer Kostenlos 15 GB Webspace zur Verfügung stellt. Per Drag & Drop können Dateien in eine Ordner-Struktur hochgeladen werden. Da Google Drive im Browser läuft, ist es möglich, Dateien plattformübergreifend zu verwalten. Diesen Service haben wir genutzt um schnell und bequem Daten auszutauschen. Insbesondere Beispielapplikationen, Frameworktutorials und Dokumentationsinhalte ließen sich so über den gesamten Projektverlauf gut zusammentragen.

2.3.6 Google Docs

Google Docs ist ein weiterer kostenloser Webservice von Google. Er bietet die Möglichkeit online Dokumente anzulegen und mit anderen Benutzern zu teilen. Wir haben Google Docs eingesetzt um unsere Dokumentation zu schreiben. Dabei haben wir uns zu Nutzen gemacht, das es möglich ist ein Dokument gleichzeitig mit mehreren Benutzern zu bearbeiten. Auf diese Weise mussten wir keine Zeit für das Zusammenführen verschiedener Dokumente aufbringen.

2.3.7 Draw IO

draw.io⁹ ist das ausgereifte Diagramming-Plugin für Confluence und JIRA, das von JGraph produziert wird. Mit draw.io lassen sich im Confluence-Wiki und im JIRA-

⁸<https://developer.android.com/sdk/index.html>

⁹www.draw.io

Tracking-System über eine intuitive und responsive Oberfläche nahtlos Diagramme aller Art entwickeln: Flow-Charts, Netzwerkdiagramme, Org-Charts, UML-Diagramme, Mindmaps und viele mehr. Die zugrundeliegende Kerntechnologie mxGraph wird bereits seit 2005 entwickelt und ist entsprechend ausgereift; 2012 ist daraus eine Anwendung entstanden, die inzwischen als Erweiterung auch auf dem Atlassian Marketplace verfügbar ist. Zusätzlich zu dieser Erweiterung kann man draw.io auch ganz simpel aus dem Browser heraus bedienen.

2.3.8 LaTeX

Das Programm TeX wurde von Donald E. Knuth, Professor an der Stanford University, entwickelt. Leslie Lamport entwickelte Anfang der 1980er Jahre darauf aufbauend LaTeX, eine Sammlung von TeX-Makros. Der Name ist eine Abkürzung für Lamport TeX. Lamports Entwicklung von LaTeX endete gegen 1990 mit der Version 2.09. Der aktuelle Nachfolger LaTeX 2 wurde ab 1989 von einer größeren Zahl von Autoren um Frank Mittelbach, Chris Rowley und Rainer Schöpf entwickelt. Wesentliche Erweiterungen von Lamports Versionen bestanden in einem „vierdimensionalen“ Mechanismus für das Umschalten zwischen Zeichensätzen („New Font Selection Scheme“, „NFSS“), in einem komplexeren Mechanismus für das Einlesen von Zusatzpaketen und in einer entsprechenden „Paketschreiberschnittstelle“ für die Erstellung und Dokumentation auf LaTeX aufbauender Makropakete. Die beiden Entwicklungsstufen von LaTeX sind nicht kompatibel. LaTeX 2 ist seit Mitte der 1990er Jahre die am weitesten verbreitete Methode, TeX zu verwenden.

3 Architektur

3.1 Technische Anforderungen

Zu den technischen Anforderungen ist zu sagen, dass wir uns zur Entwicklung einer plattformunabhängigen Anwendung darauf geeinigt haben, eine Web-Applikation mit HTML und Javascript zu implementieren. Dafür benötigen wir einen Webserver, über den die Anwendung erreichbar sein soll. Die Webseite sollte sich selbständig an verschiedene Bildschirmgrößen anpassen (responsive) und ansprechend gestaltet sein. Deshalb wird zusätzlich zu den JavaScript Bibliotheken jQuery und Ractive.js noch ein Frontend-Framework benötigt. Hier haben wir uns für Bootstrap entschieden.

Des Weiteren wird für die Steuerung des Screenflows und die Authentifikation ein Backend benötigt. Dieses wird durch einen Node-Server realisiert. Wir haben uns für einen Node-Server aufgrund der Einheitlichkeit der Programmiersprache in Backend und Frontend entschieden.

Um die persistente Speicherung aller notwendigen Daten sicherzustellen wird außerdem eine Datenbank benötigt. In unserem Fall die NoSQL-Datenbank CouchDB. Aufgrund unserer Anforderung, dass man sein Lager flexibel verwalten kann (z.B. beliebige Anzahl von Attributen für Container), entschieden wir uns für den NoSQL-Ansatz. Damit kann auch gewährleistet werden, dass eine beliebige Anzahl von Containern ineinander verschachtelt werden kann. Um die Handhabung der kompletten Datenstruktur zu vereinfachen, arbeiteten wir ebenfalls mit einem relationalen Ansatz. So speichern wir zum Beispiel Items in einem eigenen Dokument ab. Dadurch können Stammdaten separat von den eingelagerten Daten verwaltet werden. Da wir unseren Usern die Möglichkeit geben möchten, QR-Codes einzuscannen, haben wir uns dafür entschieden zusätzlich zur Webanwendung einen Android Client zu entwickeln. Im Android-Client kann die Webanwendung genau wie im Browser verwendet werden, mit dem Zusatz-Feature, dass man QR-Codes scannen kann.

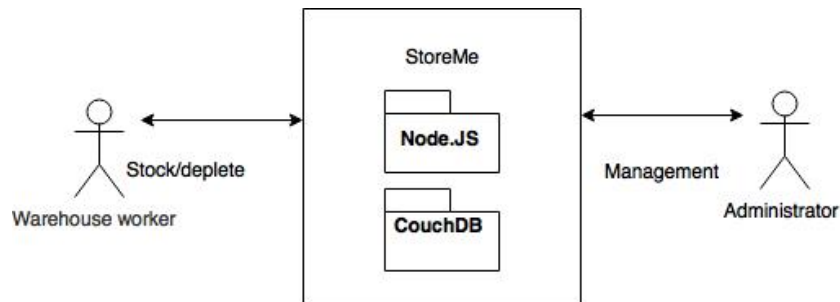


Abbildung 3.1: Grobübersicht über den Architektonischen Aufbau von StoreMe in einem Diagramm

3.1.1 UML-Diagramme

Anhand dieses Use-Case Diagramms soll der Grundaufbau von StoreMe veranschaulicht werden. Es gibt zwei Benutzer, die entweder produktiv oder administrativ auf das System zugreifen. Während der Administrator direkt am Rechner arbeitet, kann der Lagerarbeiter mit einem mobilen Gerät im Lager arbeiten. Innerhalb von StoreMe arbeitet ein Node-JS Server in Verbindung mit einer CouchDB, welche ihre Daten an den ausführenden Browser senden.

3.2 Kommunikationsmodell

Anhand unserer Anforderungen haben wir das folgende Kommunikationsmodell erarbeitet. Dieses Modell hat uns über die Phasen der Implementierung und Planung des weiteren Vorgehens sehr geholfen, da wir den Projektfortschritt stets auf einen Stand in diesem Modell beziehen konnten. Es unterstützte uns dahingehend, dass jeder mit einem Blick auf das Modell erkennen konnte, wie der logische Ablauf in unserer Anwendung konzipiert war. In diesem Dokument ist grob beschrieben, was bei einem Klick auf einen Button passieren soll, welcher Screen als nächstest angezeigt werden soll und welche Vorgänge dabei im Hintergrund ablaufen.

3.3 Problemstellung Datenmodell

Durch unsere Forderung, StoreMe möglichst universell zu gestalten, standen wir vor dem Problem, ein möglichst dynamisches und performantes Datenmodell zu entwickeln. Um die beliebig tiefe Verschachtelung zu realisieren, haben wir uns gedanklich vom relationalen Ansatz der Datenhaltung entfernt, hin zu einem NoSQL-Ansatz. Nun blieb

3 Architektur

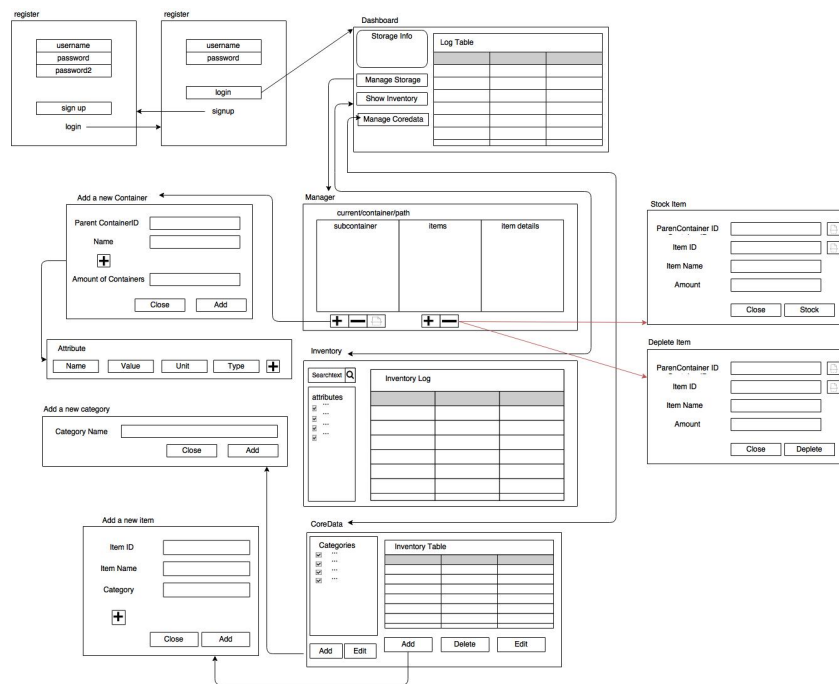


Abbildung 3.2: Endgültiger Screenflow von StoreMe

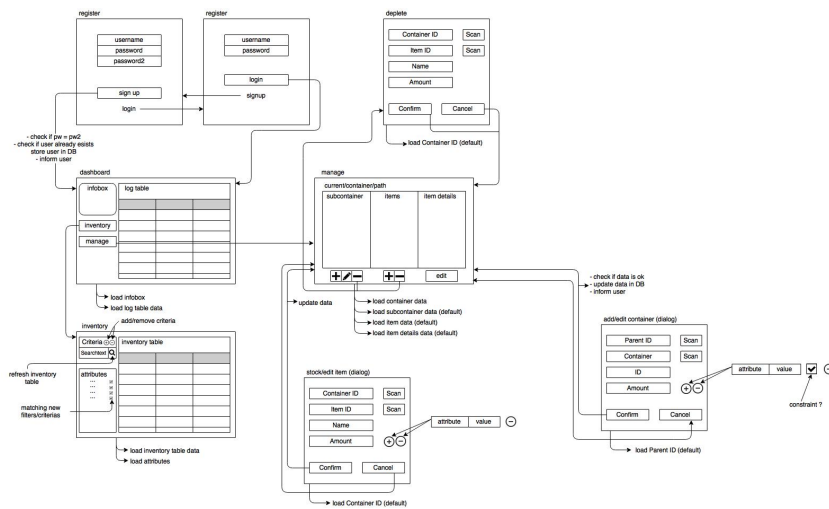


Abbildung 3.3: Erstentwurf für das Kommunikationsmodell auf den die weitere Entwicklung aufbaut

noch der Faktor der Performance zu berücksichtigen. Bezüglich dieses Gesichtspunktes haben wir ein Datenmodell entwickelt, welches das Lager als Baum abbildet. Weiterhin haben wir uns überlegt, wie wir einzelne Items speichern können, sodass ein User ein Item einmalig anlegt und dieses dann beliebig oft einlagern kann.

Die genaue Umsetzung und das Zusammenspiel zwischen Items und Container kann unter dem Punkt 4.3.2 Datenstruktur nachgeschlagen werden.

4 Implementierung

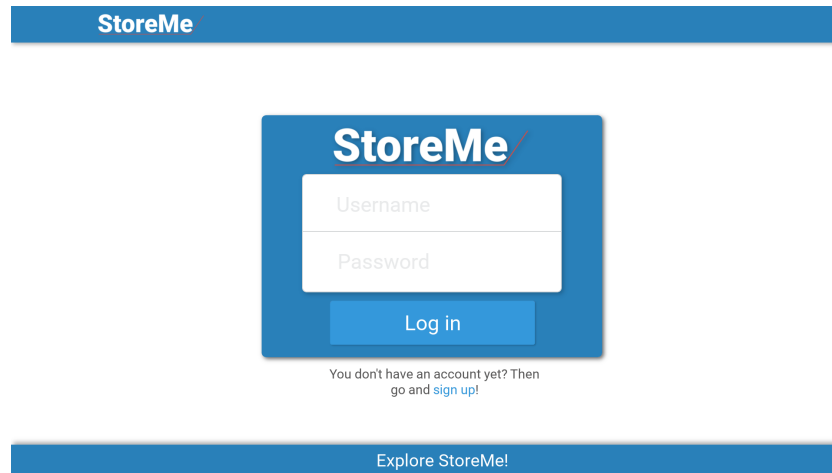
Dieser Abschnitt soll darüber informieren, wie wir die fachlichen und technischen Anforderungen umgesetzt haben. Dabei wollen wir zunächst auf die Konzeptionierung eingehen, anschließend auf die Entwicklung von Front- und Backend.

4.1 Konzept

Zu Beginn unseres Projektes stand eine lange Konzeptphase. Nach einiger Überlegung kamen wir also auf bereits beschriebene dynamische Lagerstruktur. Um sicher zu gehen, dass wir so eine realitätsgetreue Lagerabbildung mit allen nötigen Funktionen realisieren können hielten wir Rücksprache mit Herrn Professor Hennermann. In diesem Zwiegespräch bestätigte sich zunächst, dass unsere dynamische Lagerstruktur durchaus praxistauglich ist. Außerdem kristallisierten sich die unbedingt benötigten Funktionen für eine Lagerverwaltung heraus. Zusätzlich zu Lagerspiegel, Log-Screen und dem Ein- und Auslagern wurden wir auf zusätzliche Features wie das Informieren von Gabelstaplerfahrern über neue Aufgaben im Lager aufmerksam.

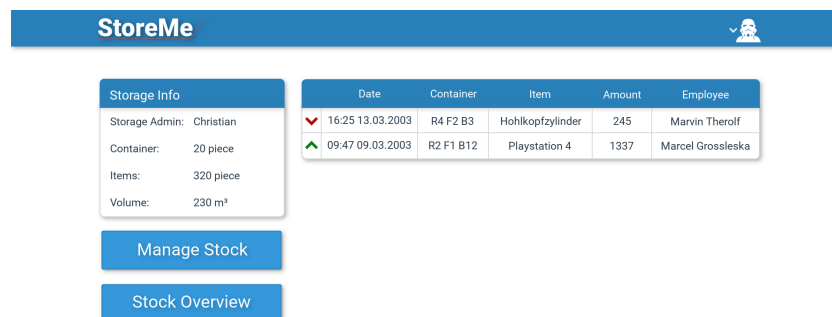
Nachdem die geforderten Basisfunktionalität geklärt waren wurden alle benötigten Screens entworfen und in Beziehung zueinander gesetzt. Dabei war uns die Benutzerfreundlichkeit und Übersichtlichkeit sehr wichtig. In mehreren Iterationen wurden so Screens um Funktionen bereichert, beschnitten oder sogar als Ganzes entfernt. Ergebnis war das unter ?? beschriebene ?. Für eine bessere Ausarbeitung der Oberflächen wurde in Gimp eine erste grafische Idee der Screens umgesetzt. An dieser und dem Kommunikationsmodell wurde sich während des gesamten Prozesses orientiert.

4 Implementierung



The mockup shows a blue header bar with the 'StoreMe' logo. Below it is a central login box with a blue border. Inside the box, the 'StoreMe' logo is repeated. There are two input fields: 'Username' and 'Password'. Below these is a blue 'Log in' button. At the bottom of the login box, there is a link: 'You don't have an account yet? Then go and [sign up!](#)'. Below the login box is a blue bar with the text 'Explore StoreMe!'.

Abbildung 4.1: Erstentwurf für den Login Screen auf den die weitere Entwicklung aufbaut



The mockup shows a blue header bar with the 'StoreMe' logo and a user profile icon. Below the header, there is a 'Storage Info' sidebar on the left and a main content area on the right. The sidebar contains the following information:

Storage Info	
Storage Admin:	Christian
Container:	20 piece
Items:	320 piece
Volume:	230 m³

Below the sidebar are two blue buttons: 'Manage Stock' and 'Stock Overview'.

The main content area contains a table with the following data:

	Date	Container	Item	Amount	Employee
✓	16:25 13.03.2003	R4 F2 B3	Hohlkopfzylinder	245	Marvin Therolf
▲	09:47 09.03.2003	R2 F1 B12	Playstation 4	1337	Marcel Grossleska

Abbildung 4.2: Erstentwurf für den Dashboard Screen auf den die weitere Entwicklung aufbaut

4.2 Front End

4.2.1 Web Client

Die Entscheidung für eine Webanwendung fiel einerseits durch ein großes Interesse des Teams im Bereich Webapplikationen und andererseits durch den Wunsch eine plattformübergreifende Applikation zu schreiben, welche auf Desktop sowie mobilen Geräten funktionieren sollte.

Grundsätzlich ist die Webapplikation in verschiedene HTML-Dateien unterteilt. Der Name des HTML-Files entspricht dem des Screens. Dabei ist die funktionelle Basis jeder HTML Datei die gleiche. Das Haupt-Skript, in unseren Fall `app.js`, wird gestartet und das passende Template geladen. Dieses Template besteht aus sogenannten `Ractive.js` Components, welche Unterelemente der Seite beinhalten.

Eine unserer Programmierkonventionen war die Anzahl der hart-gecodeten Strings innerhalb der Web-Anwendung auf ein Minimum zu reduzieren. Aufgrund dessen wurde in `StoreMe` eine `strings.js` Datei eingefügt, welche unter Benutzung des `Json-Formates` die meisten Strings verwaltet. Leider ist uns das aufgrund des Zeitmangels gegen Abschluss des Projektes nicht überall gelungen.

Listing 4.1: Zusammenstellen der Components u. Aufruf von `loadTemplate`

```
1 var components =
2 {
3   coredataContainer: coredataContainer,
4   navbarContainer: navbarContainer
5 };
6 loadTemplate( templates.coredata, components, strings.coredata );
```

Dieses Code-Snippet zeigt beispielhaft für `coredata.html` den Aufruf der Funktion `loadTemplate`, welche das der Seite zugehörige `Ractive.js` Objekt erstellt. Die passenden Components, welche im `HEAD`-Abschnitts des `HTML-Dokuments` hinzugefügt wurden, werden hier gesetzt und an `loadTemplate` übergeben. Weiterhin wird der Pfad aus `strings.js` zum passenden Template übergeben. Als letzten Parameter werden Zeichenketten übergeben, aus dem sich der Screen zusammensetzt, zum Beispiel der Titel eines Panels.

Listing 4.2: Erstellen des `Ractive.js` Objekts als `loadTemplate` Funktion

```
1 function loadTemplate( template, components, hardStrings )
2 {
3   $.ajax(
4     {
```

```

5      url: template,
6      async: true
7    }).done( function ( data )
8    {
9        window.app = new Ractive(
10        {
11            el: $('#container'),
12            template: data,
13            components: components,
14            data: hardStrings
15        });
16    });
17 }

```

Die Funktion `loadTemplate` setzt einen AJAX GET Request ab, lädt das Seiten-Template und setzt es auf das Template Objekt des Ractive Objekts. Weiterhin werden die übergebenen Components eingetragen und die Daten aus der Strings-Datei an das data-Objekt von Ractive gebunden.

Listing 4.3: Aufbau des Haupt-Templates

```

1 <navbarContainer></navbarContainer>
2 <coredataContainer></coredataContainer>

```

Das Template selbst besitzt nicht mehr als die erstellten Components, welche man in Ractive.js wie HTML-Tags benutzen kann.

Listing 4.4: Erstellung einer Ractive.js Component

```

1 var coredataContainer = Ractive.extend(
2 {
3     template: '\
4     <div class="container">\
5     <!-- ... -->
6     </div>\ ',
7
8     data:
9     {
10         edit_enabled: false
11     },
12     oninit: function()
13     {
14         this.refreshCategories();
15         this.refreshItems();

```

```

16     window.currentRactive = this;
17   },
18   refreshCategories: function()
19   {
20       //...
21   }
22 });

```

Ein Ractive.js Component wird durch die Ractive.extend - Methode erstellt. Dort wird ein eigenes Template-Objekt übergeben welches als String den HTML-Code definiert. Weiterhin kann das Data-Objekt erweitert werden, falls Ractive-Daten im Component benötigt werden. Im vorliegenden Coredata Screen kümmert sich die Variable edit_enabled beispielsweise darum, dass im Edit-Item Modal das Editieren des Items deaktiviert oder aktiviert ist.

Schließlich können vordefinierte Ractive.js Methoden verwendet werden wie oninit, welche beim Initialisieren des Components aufgerufen wird, sowie eigene Methoden geschrieben werden. Diese können als Events für Buttons im Component dienen. Auf diese Weise sind die Pages von StoreMe modular aufgebaut, leicht wartbar und erweiterbar.

Im Folgenden soll auf Funktionalitäten von Ractive.js für die Erstellung des Templates eingegangen werden. Ractive.js benutzt Mustache für den Aufbau des Templates. Eine Mustache Expression erkennt man an doppelt geschweiften Klammern: {{EXPRESSION}}.

In StoreMe wird Mustache sehr vielfältig verwendet.

Listing 4.5: Zugriff auf ein Element des Ractive.js Data-Objekt per Mustache

```

1 <div class="panel-heading">\
2 {{panel.title.category}}\
3 </div>\

```

Hier wird der Titel des Category-Panels auf dem Coredata Screen aus dem Data-Objekt von Ractive gelesen.

Listing 4.6: Schleife über Ractive.js Data-Objekt Array per Mustache

```

1 {{#each items:i}}\
2 <tr id="item_{{i}}" class="table-entry" on-click="selectItem(i)">\
3   <td>{{id}}</td>\
4   <td>{{value.name}}</td>\
5   <td>{{value.category_id}}</td>\
6       //...
7 </tr>\

```



```
8 {{/each}}\
```

Auf diese Weise werden dynamisch Tabellen mit Hilfe von Mustache erstellt. Für jedes Item aus dem Items Array wird eine Table-Row hinzugefügt. In diesem Code-Block kann auf die einzelnen Elemente eines Item-Elements zugegriffen werden. Dadurch gibt es ein Tabellenfeld für die ID, eines für den Namen usw.

Die Items werden aus der Datenbank gelesen und auf das Items Array gesetzt. Bei der Aktualisierung des Items Arrays wird dadurch ebenfalls die Item Tabelle geupdated.

Diese Funktionsweise wird bei jedem dynamischen Element von StoreMe verwendet, welches abhängig von der Datenbank ist.

Weiterhin stellt Ractive.js Bedingungen welche die Oberfläche beeinflussen zur Verfügung.

Listing 4.7: Ractive.js beherrscht ebenfalls Bedingungen

```
1 <button ... {{#unless edit_enabled}} disabled {{/unless}}>\
2     //...
3 </button>\
```

Hier wird das vorher angesprochene edit_enabled Boolean verwendet um den Button auf disabled zu setzen.

Listing 4.8: Bindung eines Elements des Ractive.js Data-Objekts an ein Oberflächenelement

```
1 <input checked="{{edit_enabled}}" type="checkbox">\
```

Der Wert des Boolean selbst wird über eine Checkbox verwaltet. Falls die Checkbox gesetzt ist, ist der Wert von edit_enabled TRUE. Andernfalls FALSE. Variablen des Data-Objekts kann man auf viele weitere Attribute eines DOM-Elements setzen. Beispielsweise auf das value-Attribut einer Input Box. Auch dies wird in StoreMe an einigen Stellen verwendet.

Als letzte Implementierungsvorgehensweise werden Ractive Events vorgestellt. Jede Aktion in StoreMe wird durch Ractive Events implementiert. Als Beispiel dient hier ein Button:

Listing 4.9: Verknüpfung eines Oberflächen-Elements mit einer Eventmethode

```
1 <button class="btn btn-primary btn-sm" ←
    on-click="removeAttributeRow(i)">\
2     //...
3 </button>\
```

Hier wird das Ractive Event on-click verwendet und mit der Funktion `removeAttributeRow` verbunden. Als Parameter wird der Index der Attribute Row übergeben, diese Code-Zeile befindet sich in einer each-Loop. Es gibt weitere Ractive Events die man mit DOM-Elementen verknüpfen kann, beispielsweise: on-change, on-tap (für mobile Geräte). Im Ractive Component kann nun eine Methode erstellt werden, welche die Funktionalität des Buttons bei einem Klick implementiert. In diesem Fall wäre es folgende:

Listing 4.10: Methode die nach Event geladen wird

```
1 removeAttributeRow: function(index)
2 {
3     window.currentRactive.splice( 'newItem.attributes', index, 1 );
4 }
```

Auf diese Weise sind alle Interaktionen des Users mit der DOM implementiert.

Im Laufe des Projektes wurde unsere Implementierung stetig besser und sauberer. Dies kann man auch an Qualitätsunterschieden des Codes je nach Screen sehen. Eine große Implementierungsschwäche von StoreMe ist die Verknüpfung des Front-Ends mit der Datenbank. Um den Datenaustausch möglichst einfach zu halten wird nach jeder Veränderung an den Daten der Datenbank im Front-End, das komplette Dokument aus der CouchDB geladen. Falls sich der Lagerumfang in Grenzen hält stellt dies kein Performance-Problem dar, jedoch ist StoreMe dadurch nicht wirklich geeignet ein großes Lager performant zu verwalten. Weiterhin gibt es in StoreMe noch einige Validierungsschwächen. Durch das Projekt haben wir bemerkt das Validierung ein sehr großes Thema bei der Erstellung einer Oberfläche ist und in diesem Bereich viel Zeit eingeplant werden muss. Ansonsten waren wir von StoreMe überrascht, da es sich im Normalbetrieb als performant erwiesen hat. Beim Verwalten eines Lagers für den Heim- oder Kleinunternehmenegebrauch konnten wir keine Performanceprobleme feststellen.

4.2.2 Android Applikation

Zur Unterstützung unserer Webanwendung haben wir uns für eine native Android Applikation entschieden, welche den User bei der Eingabe der ContainerID und ItemID mit Hilfe eines QR-Code-Scanners unterstützen soll. In unserer Applikation verwenden wir deshalb eine Webview, welche unsere Webanwendung anzeigt. Das Mobile Endgerät ist primär aber nur zur Unterstützung nicht zur Hauptbedienung der Anwendung gedacht. Nach dem Klick auf den Button Scan wird eine Scan-Activity gestartet, welche über die ZXing-Library die Kamera am Mobile Endgerät startet und den QR-Code ausliest und das Ergebnis in das zugehörige Input-Form einträgt. Wir haben uns für die ZXing-Library entschieden, weil wir bewusst keine 3rd.-Party-Applikation voraussetzen wollten.(z.B. barcoo, ZXing als native App, usw.) Aufgrund der guten Dokumentation von ZXing

haben wir uns für diese Library entschieden und nicht für einer der vielen anderen. Um eine Alternative zu nennen, gäbe es da ZBar. Besonderheiten bei der Implementierung, stellt die Schnittstelle zwischen Android und Javascript da. Dies wird Android-Seitig durch ein JavaScriptInterface umgesetzt. Um in unserer Anwendung zum Beispiel den Button-Klick in der Webview in der nativen Applikation zu nutzen, haben wir folgendes JavaScriptInterface implementiert.

Listing 4.11: Button Click Event in Android

```

1 browser.addJavascriptInterface( new Object()
2 {
3     @JavascriptInterface
4     public void performClick()
5     {
6         Intent intent = new Intent( ResultActivity.this, ↵
ScannerActivity.class );
7         intent.putExtra( "Scan", 1 );
8         intent.putExtra( "result1", result1 );
9         intent.putExtra( "result2", result2 );
10        intent.putExtra( "url", getIntentUrl( DEplete, STOCK, ↵
MODAL ) );
11        startActivity( intent );
12    }
13 }, "scan" );

```

Im JavaScript muss dem Button folgende onclick Methode hinterlegt werden.

Listing 4.12: Button Click Event in JavaScript

```

1 onclick="scan.performClick()"

```

Wird nun auf den Button geklickt startet Android ein neues Intent und übergibt ihm mögliche frühere Scan-Ergebnisse. Dieses Intent startet eine neue Activity, welche das Scannen des QR-Codes übernimmt. Um die Ergebnisse wieder in das Input-Form in der WebView einzubinden wird folgende Methode benötigt.

Listing 4.13: Befüllen des Input Forms in Android

```

1 browser.setWebViewClient( new WebViewClient()
2 {
3     @Override
4     public void onPageFinished( WebView view, String url )
5     {
6         super.onPageFinished( view, url );

```

```

8      if ( !MODAL.isEmpty() )
9      {
10         if ( !result1.isEmpty() )
11         {
12             view.loadUrl( "javascript:getScanResult( \"\" + ↵
result1 + "\", \"\" + getInputID( MODAL )[0] + "\" )" );
13         }
14         if ( !result2.isEmpty() )
15         {
16             view.loadUrl( "javascript:getScanResult( \"\" + ↵
result2 + "\", \"\" + getInputID( MODAL )[1] + "\" )" );
17         }
18         if ( !result1.isEmpty() !result2.isEmpty() )
19         {
20             result1 = "";
21             result2 = "";
22         }
23     }
24     setTempurl( url );
25 }
26 } );

```

Das JavaScript-File muss neben nachfolgender Methode je ein Input-Form mit der ID „container-id-stock“, „item-id-stock“ und „container-id-deplete“ und „item-id-deplete“ besitzen. Da der Render-Zeitpunkt von Ractive nicht genau vorhergesagt werden kann, muss bei der Ausführung der Methode sichergestellt werden, dass das Manager-Ractive vollständig geladen ist. Hierzu wird ein EventListener auf das Event „ractiveLoaded“ registriert. Besagtes Event wird in der oncomplete Methode des Manager-Ractives geworfen sobald dieses komplett berechnet wurde.

Listing 4.14: Befüllen des Input Forms in JavaScript

```

1  function getScanResult( val, id )
2  {
3      window.inputID = val;
4      if ( id.indexOf( "item" ) != -1 )
5      {
6          window.itemInputValue = val;
7      }
8      else
9      {
10         window.containerInputValue = val;
11     }
12     if ( !window.ractiveLoaded )

```

```

13  {
14      document.addEventListener( "ractiveLoaded", function()
15      {
16          if( window.itemInputValue )
17          {
18              window.currentRactive.set( ←
19              'stockItemStructure._id', window.itemInputValue );
20          }
21          if( window.containerInputValue )
22          {
23              window.currentRactive.set( ←
24              'stockItemStructure.containerID', window.containerInputValue );
25          }
26          window.addItemRactive.loadItem( ←
27          window.itemInputValue );
28          }, false );
29      }
30      else
31      {
32          if ( window.itemInputValue )
33          {
34              window.currentRactive.set( 'stockItemStructure._id', ←
35              window.itemInputValue );
36          }
37          if ( window.containerInputValue )
38          {
39              window.currentRactive.set( ←
40              'stockItemStructure.containerID', window.containerInputValue );
41          }
42      }
43  }

```

Beim Aufrufen der `browser.setWebViewClient()` wird der Webview ein `WebViewClient` zugewiesen, welcher wartet bis die Seite fertig geladen wurde und ruft dann die JavaScript-Methode `getScanResult()` auf. Diese bekommt neben dem Scan-Ergebnis auch die ID mit in welches Input-Form diese geschrieben werden soll.

4.3 Back End

Unser Backend hat im wesentlichen 3 Hauptaufgaben. Zum einen die Handhabung der GET-Requests, welche die benötigten Dateien an das Front-End sprich den Client liefern. Die zweite Aufgabe ist es beim starten des Node-Servers zu prüfen ob alle benötigten Datenbanken in der laufenden Couch-Datenbank existieren. Sollte dies nicht der Fall sein, werden diese automatisch angelegt. Für dieses automatische Erstellen der Datenbanken, müssen diese mit dem gewünschten Namen in strings.js stehen.

Listing 4.15: JSON Objekt mit Datenbanken

```
1
2 database:
3 {
4     container:"storemecontainer",
5     user:"storemeusers",
6     items:"storemeitems",
7     category:"storemecategory",
8     attributes: "storemeattributes",
9     log:"storemelog"
10 },
```

Die letzte Aufgabe ist Registrierung und Authentifizierung von Usern. Dies kann unter dem Punkt 4.3.1 Authentifizierung nachgelesen werden.

4.3.1 Authentifizierung

Die Authentifizierung des Nutzers mit StoreMe erfolgt erstmalig beim Login und von diesem Punkt aus bei jedem Seitenaufruf. Da die Nutzerdaten mit der Datenbank abgeglichen werden, müssen Nutzer sich zuvor registrieren.

4.3.1.1 Registrierung

Das Front End sendet via POST-Request den Usernamen und das Passwort des neuanzulegenden Users im Authorization-Header mit. Beides ist mit storeMeEncrypt (Beschreibung im Anhang) verschlüsselt. Zusätzlich wird das Passwort mit SHA1 gehashed. Die Hashfunktion kommt aus der Library von <https://code.google.com/p/crypto-js/>.

4.3.1.2 Login

Mittels eines GET-Request, in dem die Userdaten im Authorization-Header analog zum POST-Request der Registrierung mitgesendet werden, wird die Datenbank nach den übergebenem Usernamen und Passwort durchsucht und verglichen. Sollte es eine Übereinstimmung geben und somit der Login-Vorgang erfolgreich sein, wird mit Hilfe des Usernamens und des Passworts eine neue Session erzeugt, welche für 10 Minuten gültig ist. (Beschreibung der Methode zur Erzeugung neuer Sessions im Anhang unter sessions/session_handler) Diese Session wird im Response mitgegeben und an die URL angehängt. Versucht ein Nutzer nun auf eine geschützte Seite zuzugreifen wird die in der URL mitgegebene SessionID auf Gültigkeit geprüft. Ist diese noch gültig, wird der Timeout wieder auf 10 Minuten gesetzt und die angeforderte Seite wird aufgerufen. Sollte ein User ohne SessionID oder mit abgelaufener/ungültiger SessionID versuchen eine geschützte Seite aufzurufen wird dieser automatisch auf den Login-Screen weitergeleitet.

4.3.2 Datenstruktur

Unsere Datenstruktur ist insbesondere ein Baum. Der Container, der das Lager repräsentiert kann dabei als Root-Knoten angesehen werden, dessen Kinder die einzelnen Strukturelemente des Lagers rekursiv abbilden. Wird die Lagerverwaltung erstmals geöffnet wird der Benutzer daher aufgefordert, ein Lager anzulegen. Hier wird der Root-Container erstellt. Um die Baumstruktur abzubilden besitzt jeder Container ein Array mit seinen Subcontainern. Weitere Parameter sind die Container-ID, ein Name, ein Reihe an Container-Attributen sowie ein Array der im Container befindlichen Items. Die Container-ID verweist dabei auf den Lagerort. Sie wird automatisch vergeben, wenn der Container angelegt wird. Der Root-Container bekommt die ID 0. Sein erster Subcontainer bekommt die ID 0-0, der zweite 0-1 usw. Die Items, welche sich in einem Container befinden werden mittels einer Hilfsklasse abgelegt. Diese heißt treffenderweise ContainerItem. Ein ContainerItem besteht aus einer Item-ID und der eingelagerten Anzahl. Außerdem wird vermerkt, in welchem Container das Item liegt (Rückverweis). Container-Attribute werden beschrieben durch einen Namen, einen Wert, einer Einheit, einen Typ (logischer Wert oder physikalischer Wert). Dazu wird vermerkt, ob ein Attribut verpflichtend von Subcontainern erfüllt werden muss (compulsory). Auch Items werden in einer eigenen Klasse abgebildet. Hier werden alle in der Datenbank abgelegten Item-Informationen dargestellt. Hier sind die Item-Attribute besonders zu nennen. Diese unterscheiden sich von den Container-Attributen vor allem dadurch, dass hier ein compulsory-Verweis fehlt. Zur weiteren Information haben wir eine ausführliche Code-Dokumentation mit JSDocs erstellt. Diese finden Sie im dem Projekt angefügten Ordner.

4.3.3 Datenbank-Verbindung

CouchDB stellt eine eigene jQuery Bibliothek zur Verfügung um mit der Datenbank zu kommunizieren. Diese Kommunikation läuft via HTTP-Requests.

Um einen Eintrag mittels seiner `_id` aus der Datenbank mit Hilfe der zur Verfügung gestellten jQuery Bibliothek zu laden benötigt man folgenden Aufruf.

Listing 4.16: Funktion für GET-Request auf URL

```

1 $.couch.urlPrefix = http://localhost:5984;
2 $.couch.db( "container" ).openDoc( "mainstore",
3 {
4     success: function( data )
5     {
6         //do something with the returned data
7     },
8     error: function( status )
9     {
10    }
11 } );
```

Dieser Methodenaufruf ist gleichzusetzen mit einem GET-Request mit dieser URL: `http://localhost:5984/container/mainstore`

4.4 Sicherheit

Die Entwicklung einer Web-basierten Mehrbenutzeranwendung stellte uns auch vor Sicherheitsfragen: Wie soll gewährleistet werden, dass ein Nutzer eingeloggt bleibt? Wie lange soll man eingeloggt sein? Wollen wir Nutzernamen und Passwort wirklich “nur” Base64-kodiert durch das Internet schicken? Um diese und andere Fragen zu klären haben wir ein Verfahren zur Generierung von Session-IDs sowie einen Verschlüsselungsalgorithmus implementiert.

4.5 Session IDs

Eine Session ID besteht aus dem Benutzernamen sowie einer zehnstelligen Zahl. Diese zehnstellige Zahl hat zwei Bestandteile: Die ersten beiden Ziffern bilden eine Prüfsumme

über den Benutzernamen. Bei den folgenden acht Ziffern handelt es sich um die Session-Nummer. Diese ist ein auf 8 Ziffern reduzierter Hash-Wert auf Grundlage von Nutzernamen, Passwort sowie dem Zeitpunkt der Initialisierung.

4.6 Verschlüsselung

Wir wollten vermeiden, sensible Daten wie z.B. die Kombination von Nutzernamen und Passwort unverschlüsselt zu übertragen. Die gängige Base64 Kodierung erschien uns wenig geeignet. Wir haben daher ein Verfahren namens StoreMeCrypt entwickelt. Dabei handelt es sich um ein simples Text-basiertes Verschlüsselungsverfahren, welches nach dem Rotationsprinzip arbeitet. Das Codewort ist “StoreMe”. Ein Beispiel anhand der Nachricht “HalloWelt”: 4.1 Verschlüsselungstabelle

Erwähnenswert an StoreMeCrypt ist, dass wir nicht die ASCII-Tabelle benutzen. Das liegt insbesondere daran, dass es aufgrund unserer Arbeit mit Query-Parametern ausgeschlossen sein muss, dass “=” in der Session-ID vorkommen. Wir haben also ein eigenes Alphabet genommen. Das hat außerdem den Vorteil, dass wir einen schnellen Überblick über bzw. Zugriff auf die erlaubten Zeichen haben und unser Alphabet im Gegensatz zur ASCII-Tabelle beliebig erweiterbar bleibt. Desweiteren möchten wir festhalten, dass StoreMeCrypt den gleichen Schlüssel zum Ver- und Entschlüsseln verwendet. Das macht die Anwendung sehr unkompliziert. Natürlich ist uns bewusst, dass StoreMeCrypt keinen Anforderungen an eine wirklich sichere Verschlüsselung standhält. Für uns war es jedoch ein lehrreicher Einblick in die Welt der Kryptographie. Wir wollten vor allem zeigen, dass wir uns mit dem Thema Sicherheit auseinandergesetzt haben.

4.7 Testing

Wenn man mit mehreren Menschen in einem Team an einem Code arbeitet, dann muss man sich darauf verlassen können, dass die Funktionen, die ein Teammitglied programmiert hat auch das tun, was sie sollen. Um diesem Anspruch so gut wie möglich gerecht zu

	H	a	l	l	o	W	e	l	t	(Nachricht)
+	S	t	o	r	e	M	e	S	t	(Schlüssel)
=	j	S	Y	b	R	s	H	C	l	(verschlüsselt)
-	S	t	o	r	e	M	e	S	t	(Schlüssel)
	H	a	l	l	o	W	e	l	t	(Nachricht)

Tabelle 4.1: Verschlüsselungstabelle

werden haben wir uns umfassend mit dem Thema Testing befasst. Frei nach dem Ansatz des “Test Driven Developements” haben wir Testfiles für nahezu jedes JavaScript-File angelegt. Innerhalb dieser Testfiles wurden die neue geschriebenen Funktionen anhand von Realbeispielen getestet während sie geschrieben wurden. Dadurch blieb uns häufiges Fehlersuchen weitestgehend erspart.

5 Evaluation

5.1 Das kann StoreMe

Mit StoreMe ist man aktuell in der Lage, ein Lager zu erstellen. Dies ist mittels einer verschachtelten Containerstruktur realisiert. Vergleichbar ist dies mit einer Ordnerstruktur, wodurch ein einfaches und intuitives Navigieren möglich ist. Desweiteren gibt es eine Stammdatenverwaltung. Hier können verschiedene Kategorien erstellt, editiert und gelöscht werden. Desweiteren können hier einzelne Items erstellt, geändert oder gelöscht werden. Außerdem ist möglich jedem Stammitem individuell Attribute zuzuweisen. Bei Bedarf kann ein QR-Code zu einem Item erstellt und ausgedruckt werden. Innerhalb der einzelnen Container kann eine beliebige Anzahl an Items dem jeweiligen Container zugewiesen (einlagern) und wieder entfernt (auslagern) werden. Bei bereits eingelagerten Items können die genaueren Infos zu den Items eingesehen werden. Auch hier ist es möglich einen QR-Code zu einem Container zu erstellen und zu drucken. Weiterhin bietet StoreMe die Funktion sich alle vorhandenen Items inklusive Anzahl anzeigen zu lassen. Hier ist auch sofort ersichtlich wo sich ein Item befindet. Auch eine Filterung der angezeigten Items ist möglich. Über die Log Tabelle kann man jederzeit die Itembewegung nachvollziehen. An dieser Stelle ist sofort ersichtlich, wer, was, wo, wann und wieviel eingelagert oder ausgelagert wurde. Ein Mehrbenutzerbetrieb ist in StoreMe vorhanden, jedoch kann für ein korrektes Verhalten nicht garantiert werden.

5.2 Das kann StoreMe nicht

StoreMe verfügt noch nicht über die Möglichkeit eine Einlagerungsstrategie zu wählen. Daraus folgt, dass es beim Einlagern bestimmter Items keine Vorschläge für einen geeigneten Lagerplatz gibt. Eine Umlagerung wurde nicht explizit implementiert und ist deswegen nur durch die Kombination von Aus- und erneutes Einlagern möglich. Außerdem wurde aufgrund von Komplikationen mit der Datenbank das nachträgliche Ändern von Item Attributen deaktiviert. Dies bedeutet, dass ein Item gelöscht und wieder hinzugefügt werden muss um ein geändertes Item als Stammdatum zu besitzen. Ein weiteres nicht-implementiertes Feature sind Container-Restriktionen. Beispielsweise die Pflicht, dass in einem Container nur flüssige Attribute eingelagert werden können beschreibt eine

Container Restriktion. Eine automatische Kommisionierung beim Auslagern ist mit StoreMe noch nicht realisiert. Auch eine Unterstützung für eine Inventur fehlt bislang. Zuletzt ist es noch nicht möglich einem Benutzer bestimmte Rechte zu vergeben bzw. zu entziehen.

6 Fazit

Bevor wir zum eigentlichen Fazit kommen, wollen wir kurz rekapitulieren, was von uns in diesem Semester erwartet wurde. Gefordert war, dass wir die Jahresleistung eines praxiserfahrenen Programmierers stemmen. Diese entspricht laut der Parallelveranstaltung "Projektmanagement" einer Arbeitszeit von über 1500 Stunden. Diese Forderung ist offenkundig unrealistisch hoch. Das Programmierprojekt wird mit 5 CP bewertet. Diese sollen einen Semesteraufwand von 150 Stunden abdecken. Bei 6 Teammitgliedern bedeutet dies einen Zeitaufwand von ungefähr 900 Stunden. Das Programmierprojekt war auf 17 Wochen angelegt. Das ergibt eine wöchentliche Arbeitsleistung von 8 bis 9 Stunden pro Teammitglied pro Woche. Selbstverständlich hat keiner von uns mit der Stoppuhr am Arbeitsplatz gesessen, eine minutengenaue Dokumentation der Arbeitszeit gibt es nicht. Wir möchten hier jedoch erwähnen, dass es nahezu an jedem Dienstag ein längeres Team-Meeting sowie ein Treffen mit unserem Betreuer Herrn Schreibmann gab. Konkret am Projekt gearbeitet hat dann jede Gruppe im Laufe der Woche. In der Planungsphase sowie zum Ende der Implementierungsphase haben wir uns dann an den Wochenenden getroffen und gearbeitet. Dabei ist es durchaus vorgekommen, dass die Mehrheit des Teams Freitag bis Sonntag täglich 8 Stunden mit dem Projekt beschäftigt war. Die eine oder andere Nachtschicht wurde auch eingelegt. Dazu kam der Einarbeitungsaufwand: Allein die Code-Academy Tutorial zu HTML, JavaScript und jQuery werden von den Herausgebern mit insgesamt 20 Stunden Aufwand beschrieben. Jeder von uns hat diese Tutorials bearbeitet. Alleine in dieser Vorbereitungszeit entstand somit ein Aufwand von ungefähr 120 Stunden. Dabei ist die Einarbeitungszeit in die diversen Tools und Frameworks noch gar nicht berücksichtigt. Wir können mit gutem Gewissen von uns sagen, die geforderte Arbeitszeit während des Semesters erbracht zu haben.

Kommen wir nun zum eigentlichen Fazit. Sicherlich ist StoreMe nicht das, was ein professioneller Webentwickler innerhalb von 1500 Stunden aus der Idee gemacht hätte. Würde man diesen Anspruch tatsächlich an unsere Projektarbeit anlegen, dann würde man dabei einige Hürden übersehen, mit denen ein professioneller Entwickler nicht konfrontiert ist: Dieser kennt die normalerweise die Programmiersprache, hat Erfahrung im Umgang mit Tools und ist mit gängigen Entwicklungskonzepten vertraut. Er ist schlicht deutlich produktiver, als eine Gruppe Studierender ohne Projekterfahrung, die versuchen, eine relativ wage Idee ohne äußere Führung mit ihnen unbekannten Technologien umzusetzen.

Nichtsdestotrotz sind wir sehr stolz auf unsere Leistung. Fachlich hat jeder von uns

durch die Arbeit mit den neuen Technologien und den Umgang mit oft verwirrenden Problemstellungen einen gigantischen Schritt vorwärts gemacht. Die Entscheidung gegen das altbewährte bereuen wir in keinem Fall. Auch die persönliche Entwicklung wurde durch Kommunikationshürden und Teamkonflikte während der vergangenen 17 Wochen stark voran getrieben. Jeder von uns kann heute von sich behaupten, schon einmal ein komplettes Projekt im Team bearbeitet zu haben. Unsere Software erfüllt ihren Zweck, läuft flüssig, ist benutzerfreundlich und sieht gut aus. Die Entwicklung hat uns großen Spaß gemacht, uns stark gefordert und gefördert. In unseren Augen war das Programmierprojekt ein voller Erfolg.

7 Ausblick

Software ist nie fertig. Man hört nur irgendwann auf zu programmieren. Das gilt natürlich auch für StoreMe. Wir konnten eine Menge Features umsetzen, einige haben es noch nicht in die finale Version geschafft. Da wir StoreMe mit einer uns zuvor unbekannten Sprache entwickelt haben, hat StoreMe selbstverständlich auch noch einige Schönheitsfehler, die vor einem Release noch auszubessern wären.

7.1 Problembehandlung

7.1.1 Container editieren

Zur Zeit können Container angelegt und gelöscht werden. Eine nachträgliche Bearbeitung ist allerdings noch nicht möglich. Das liegt insbesondere daran, dass sich das Ändern von Container-Eigenschaften in hohem Maße auf die Prüfung der Constraints auswirken würde. Diese Funktionalität müsste vorher entwickelt werden.

7.1.2 Mehrbenutzerbetrieb

Mehrere Benutzer können zeitgleich mit dem gleichen Lager arbeiten. Das liegt insbesondere an der Webarchitektur, welche mit Requests und Responses arbeitet. Problematisch sind zur Zeit noch Bearbeitungen, die eine transaktionelle Verarbeitung erfordern würde (z.B. lagert jemand Items in einen Container ein, während ein anderer selbigen Container löscht). Im Realbetrieb sind die meisten dieser Problemszenarien allerdings unschädlich, da dem digitalen stets ein realer Lagervorgang gegenüber stünde und viele der problematischen Anwendungsfälle rein theoretischer Natur sind.

7.1.3 Benutzerverwaltung

Die Benutzerverwaltung ist zur Zeit sehr rudimentär und umständlich. Nutzer können sich registrieren und einloggen. Passwörter werden nicht im Klartext abgelegt, sondern vorher mit dem Verfahren SHA1 [5] gehasht. Darüber hinaus sind Anpassungen nur über direkte Datenbankzugriffe seitens eines Admins möglich. Es gibt also Nachholbedarf in Sachen Profil- und Rechteverwaltung.

7.1.4 Performance

In unseren Testfällen lief StoreMe stets zuverlässig und flüssig. Bei der Verwaltung eines sehr großen Lagers mit vielen Benutzer die zeitgleich darauf zugreifen müssen sicher einige Anpassungen bezüglich der Performance vorgenommen werden. Insbesondere die vielen Animationen sowie die Tatsache, dass im aktuellen Betrieb der gesamte Lagerbestand im Hauptspeicher abgelegt wird, bedürfen noch einiger Arbeit.

7.1.5 Constrains

Die Prüfung der Items in Bezug auf Container-Constraints konnten wir zeitlich nicht mehr implementieren. Alle dafür notwendige Eingabefelder sind jedoch bereits in StoreMe vorhanden. Der Implementierungsaufwand beschränkt sich also auf die logische Prüfung sowie ein ansprechendes Warnungssystem.

7.2 Features

7.2.1 Inventur

Ein mächtiges Feature wäre die Unterstützung einer Inventur. Wir stellen uns das so vor, dass man von zentraler Stelle einen Inventurmodus startet. Während des Inventurmodus würden dann alle Lagerbearbeitungen deaktiviert. Gescannte Items würden in die Inventurliste aufgenommen. Nach Abschluss des Inventurmodus wäre es dann möglich, eine Differenzliste zu generieren.

7.2.2 Lagerplatzvorschlag

Als ein weiteres wichtiges Feature haben wir uns eine Lagerplatzempfehlung ausgedacht. Scannt man ein Item über die Einlager-Funktion bevor man einen Container ausgewählt hat, sollte das System in der Lage sein, sinnvolle Lagerplätze vorzuschlagen. Ein Kriterium für den Vorschlag könnten z.B. sein, dass das gleiche Item bereits in einem bestimmten Container gelagert wird, oder das Items gleicher Kategorie typischerweise in einem bestimmten Container gelagert werden. Zuvor würden die Container-Constraints natürlich mit den Item-Attributen abgeglichen werden.

7.2.3 Rechteverwaltung

Im Gespräch mit Professor Hennermann wurde klar, dass unser System auch auf die Benutzung sog. DAUs (Dümmste Anzunehmende User) zugeschnitten sein muss. Das bedeutet, dass ein StoreMe-User nicht in der Lage sein darf, das System über seine Befugnisse hinaus zu verändern (z.B. Löschen der Datenbank). Diese Einschränkung setzt eine konfigurierbare Benutzerverwaltung mit einem individuellen Rollen- und Rechtesystem voraus. Dieses Feature ist für eine zukünftige Version von StoreMe fest eingeplant.

7.2.4 Alertsystem für Administratoren

Zur Verbesserung von Fehlern ist es notwendig, dass Administratoren über ein Werkzeug verfügen, welches es ermöglicht Fehlerquellen schnell auffindig zu machen. Zur Zeit befinden sich zu diesem Zweck einige Konsolen-Logs im Produktivcode von StoreMe. Diese sind in Zukunft durch ein benutzerfreundliches Ticket- und Alertsystem zu ersetzen.

8 Tutorial

Nachfolgend wird auf die Installation, Inbetriebnahme und den Bedienungsablauf eingegangen.

8.1 Installation

Die Installation von StoreMe ist sehr einfach gehalten, um schnell ein Produktivsystem aufziehen zu können.

8.1.1 StoreMe

StoreMe ist eine einfach zu nutzende Lagerverwaltungssoftware, die direkt nach einrichten des NodeJS Server, der CouchDB mit jedem Internetbrowser verwendet werden kann.

8.1.2 Node.js Server

Unter Windows muss der Node.js Server zunächst heruntergeladen ¹ und installiert werden. Ob die Installation fehlerfrei war, kann über die cmd mit dem Kommando “node” getestet werden. Es erscheint eine neue Zeile mit einem Pfeil. Mit Strg + C kann Node.js beendet werden.

Für die Installation auf einem Linux System wird im Folgenden von Ubuntu ausgegangen. Zu Beginn müssen folgende Befehle in der Shell ausgeführt werden um diverse Abhängigkeiten zu installieren:

Listing 8.1: Installation benötigter Abhängigkeiten unter Linux

```
1 sudo apt-get install g++ curl libssl-dev apache2-utils
2 sudo apt-get install git-core
```

¹<https://nodejs.org/download/>

Als nächstes wird das Git-Repository von Node.js geklont und Node.js lokal gebaut und installiert. Im Ordner in dem Node.js installiert werden soll müssen folgende Kommandos ausgeführt werden

Listing 8.2: Node.js Server installation unter Linux

```
1 git clone git://github.com/ry/node.git
2 cd node
3 ./configure
4 make
5 sudo make install
```

Nun ist Node.js installiert und einsatzbereit.

Für die Benutzung von StoreMe wird an dieser Stelle auf das readme-file im Projektordner verwiesen.

8.1.3 CouchDB

Unter Windows muss zunächst CouchDB ² heruntergeladen und installiert werden. Anschließend muss in der ...\\CouchDB\\etc\\couchdb die local.ini angepasst werden um die Same-Origin-Policy aufzuheben. Dazu muss der Server in der cmd mit NET STOP "Apache CouchDB" heruntergefahren werden. In der local.ini wird unter [cors] origins = * und unter [httpd] enable_cors = true hinzugefügt. Zusätzlich sollte unter [log] der Eintrag level auf none umgestellt werden, da mit dem Standardwert debug jegliche Daten ausgelesen werden können. Im Fehlerfall kann dies auch für Debug zwecke wieder auf debug gesetzt werden. Alternativ kann unter http://<IP>:5984/_utils/config.html mit Futon diese Änderungen durchgeführt werden (hierfür ist kein Herunterfahren des Servers notwendig). Anschließend muss der Server wieder mit NET START "Apache CouchDB" gestartet werden. Nun ist der CouchDB-Server einsatzbereit.

Für Linux wird stellvertretend wieder Ubuntu verwendet. Für die CouchDB können die offiziellen Paketquellen genutzt werden.

Durch den Befehl:

Listing 8.3: CouchDB installation unter Linux

```
1 sudo apt-get install couchdb -y
```

wird die CouchDB heruntergeladen und installiert.

²<http://couchdb.apache.org/>

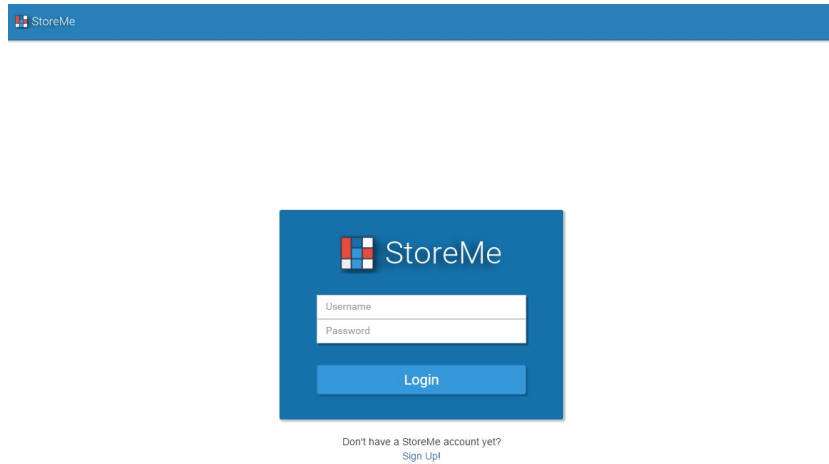


Abbildung 8.1: Login/Sign up Screen als ersteinstieg in die Anwendung

Anschließend muss per `http://<IP>:5984/_utils/config.html` die Futon-Oberfläche aufgerufen werden. Per Klick auf Configuration im rechten Panel werden die Optionen für die CouchDB aufgerufen. Ganz unten können per “Add new Section” die Optionen wie oben im Windows-Teil eingestellt werden.

8.1.4 Android Applikation


Die Applikation herunterladen (von der Startseite) und auf dem Android Gerät installieren. Beim ersten starten der Anwendung muss die Adresse zum Back End in den Client eingetragen werden. Diese kann nach belieben editiert werden. Nach dem Eintragen einer korrekten URL kann die Applikation genutzt werden.

8.2 Bedienung

8.2.1 Login und Signup

Der Login Screen- bzw Sign up Screen ist der Eintrittspunkt in StoreMe. Hier besteht die Möglichkeit sich einen neuen Account anzulegen oder sich in einen bestehenden Account einzuloggen. Sollte noch kein Useraccount erstellt worden sein, kann problemlos über den Sign-Up Link oder die direkte Verlinkung auf der Loginseite darauf zugegriffen werden. In der Sign-Up Ansicht werden Username, Password und eine Password Wiederholung benötigt. Während der Name an keine besonderen Vorgaben gebunden ist, benötigt das

StoreMe



Storage Info

Storage Admin: Christian

Container: 20 piece

Items: 320 piece

Volume: 230 m³

Manage Stock

Stock Overview

Date	Container	Item	Amount	Employee
<div>✓</div> 16:25 13.03.2003	R4 F2 B3	Hohlkopfzylinder	245	Marvin Therolf
<div>▲</div> 09:47 09.03.2003	R2 F1 B12	Playstation 4	1337	Marcel Grossleska

Abbildung 8.2: Überblick über das Dashboard mit Inventarlog

Password mind. 7 Zeichen darunter mind. ein Großbuchstabe, mind. eine Zahl und mind. ein Kleinbuchstabe.

8.2.2 Das Dashboard

Auf den Dashboard Screen kommt man, nachdem man sich eingeloggt hat. Auf dem Dashboard sind alle Einlagerungs- und Auslagerungsvorgänge die auf diesem Lager ausgeführt wurden protokolliert, auch von welchem Benutzer sie durchgeführt wurden. Desweiteren ist ersichtlich welcher User eingeloggt ist und wieviele Container und Items im gesamten Lager verwaltet werden. Von der Dashboard Seite gibt es direkte Weiterleitungen auf die Managementansicht, das Inventar und die Stammdaten mithilfe von Buttons. Mit einem Klick auf das StoreMe Logo kommt man von jeder Ansicht zurück auf das Dashboard.

8.2.3 Management Ansicht

Wird zum erstenmal die Management Seite aufgerufen, wird geprüft ob bereits ein Lager angelegt wurde. Ist dies nicht der Fall kann direkt, einfach über die Namensvergabe, ein neues leeres Lager angelegt werden. Ansonsten wird dieser Schritt übersprungen und man kommt direkt auf die Managementansicht.

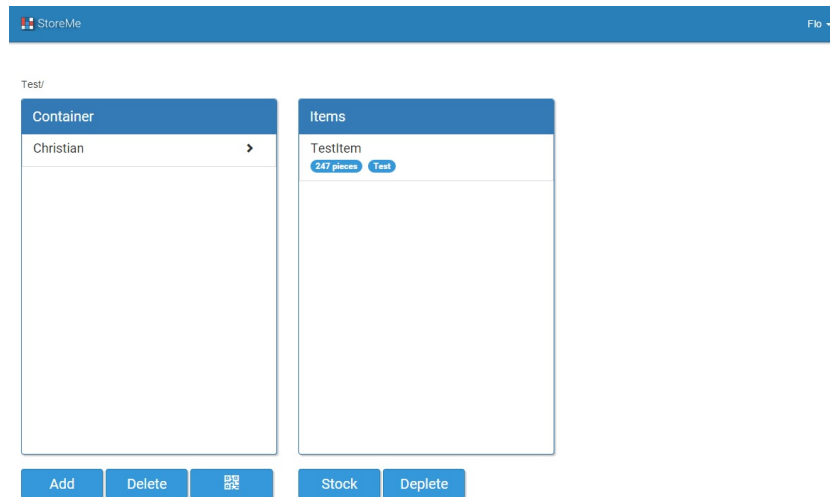


Abbildung 8.3: Management Ansicht mit Container- und Itemsicht und ausgeblendeter Itemdetailsicht

8.2.4 Container hinzufügen und löschen

In der Management Ansicht können u.a. neue Container erstellt werden. Die ContainerID wird automatisch vom System erzeugt. Containernamen müssen vom User vergeben werden. Container können verschiedene Attribute zugeteilt werden, um sie z.b. für Flüssigkeiten zu kennzeichnen. Wird beim erstellen eines Containerattributes der Hacken für Compulsory gesetzt, erhält jeder der darunterliegenden Container auch dieses Attribut. Desweiteren können auch direkt mehrere gleiche Container über die Amount Angabe erstellt werden. Container können nur gelöscht werden wenn ein Container ausgewählt ist, andernfalls ist der delete Button nicht hinterlegt. Durch Mehrfachauswahl können mehrere Container gleichzeitig gelöscht werden. Wird ein Container gelöscht, der noch Items enthält werden dessen Items gelöscht. Desweiteren kann über ausgewählte Container ein Barcode für diesen erstellt werden.

8.2.5 Stammdaten anlegen, bearbeiten und löschen

StoreMe bietet die Möglichkeit Stammdaten im Coredata Screen zu verwalten. Stammdaten sind notwendig um Items einlagern und auslagern zu können. Wenn Stammdaten angelegt werden, muss vom Benutzer eine eindeutige ID, ein Name und ggf. die Zuordnung zu einer Kategorie vergeben werden. Eine Doppelvergabe von ID's ist nicht möglich. Mithilfe der Itemansicht können die Vergebenen Attribute überprüft und/oder mit der Enable Editing Option angepasst werden. In der Coredata Ansicht können Kategorien zu den Items angelegt und geändert werden. Kategorien können nicht gelöscht werden, solange Items noch dieser Kategorie zugewiesen sind. Desweiteren bietet diese Ansicht

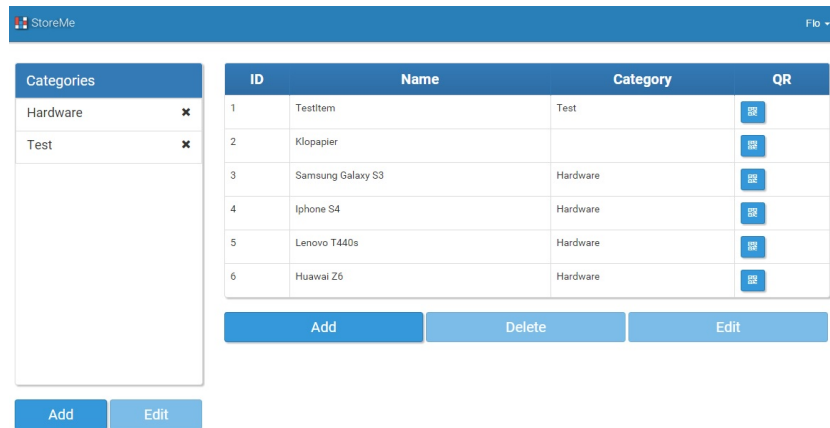


Abbildung 8.4: Coredata Screen mit Kategorien und angelegten Stammitems

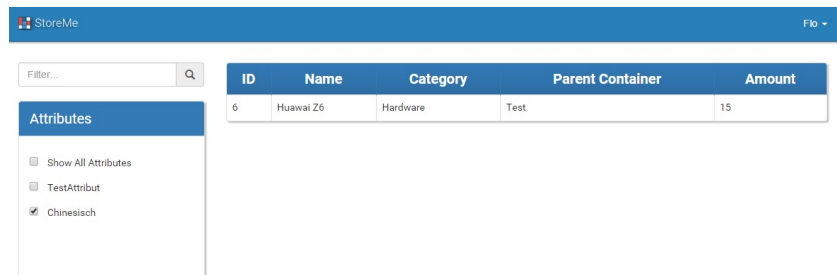
die Möglichkeit direkt QR-Codes von Items zu erstellen und dann zu drucken. Die Option Kategorien oder Items zu löschen ist nur möglich wenn das zu löschende Objekt ausgewählt wurde.

8.2.6 Items einlagern und auslagern

Innerhalb der Management Ansicht finden sich die Funktionalitäten um Items einzulagern oder auszulagern. Dafür muss nur die ItemID eingetragen werden und das System sucht sich selbstständig das passende Item aus den CoreDataas, nachdem die Auswahl des ItemID Feldes aufgehoben wird. Die Dialogfenster von Stock und Deplete bieten auch die Funktionalität für die Android App um mithilfe des QR-Codes Items in Container ein- und auszulagern.

8.2.7 Nach Items suchen

Über die Inventory Screen können Items anhand des Namens, der ID, Kategorie und des Namens des Parentcontainers gesucht werden. Anhand vorher erstellter Attribute können weitere Filterungen durchgeführt werden, z.b. das nur bestimmte Attribute berücksichtigt werden.



The screenshot shows the StoreMe application interface. At the top is a blue header bar with the 'StoreMe' logo on the left and a user profile icon labeled 'Flo' on the right. Below the header, on the left, is a search bar with the placeholder text 'Filter...' and a magnifying glass icon. To the right of the search bar is a table with five columns: 'ID', 'Name', 'Category', 'Parent Container', and 'Amount'. The table contains one data row with the following values: ID '6', Name 'Huawei Z6', Category 'Hardware', Parent Container 'Test', and Amount '15'. To the left of the table is a sidebar titled 'Attributes' which contains three checkboxes: 'Show All Attributes' (unchecked), 'TestAttribut' (unchecked), and 'Chinesisch' (checked).

ID	Name	Category	Parent Container	Amount
6	Huawei Z6	Hardware	Test	15

Attributes

- ☐ Show All Attributes
- ☐ TestAttribut
- ☒ Chinesisch

Abbildung 8.5: Inventory Screen mit aktivem Attributfilter

Abbildungsverzeichnis

3.1	Architekturübersicht StoreMe	21
3.2	Finaler Entwicklungsstand Screenflow	22
3.3	Kommunikationsmodell Konzeptentwurf	22
4.1	Login Screen Konzeptentwurf	25
4.2	Dashboard Konzeptentwurf	25
8.1	Login Screen	48
8.2	Dashboard Screen	49
8.3	Management Ansicht	50
8.4	Coredate Screen	51
8.5	Inventory Screen	52

Tabellenverzeichnis

2.1	Nummern Attribute	9
2.2	Boolean Attribute	9
2.3	Number-Compulsory-Attribute	9
2.4	Boolean-Compulsory-Attribute	10
4.1	Verschlüsselungstabelle	37

Listings

2.1	HTML Dokumentspezifizierung	11
4.1	Zusammenstellen der Components u. Aufruf von loadTemplate	26
4.2	Erstellen des Ractive.js Objekts als loadTemplate Funktion	26
4.3	Aufbau des Haupt-Templates	27
4.4	Erstellung einer Ractive.js Component	27
4.5	Zugriff auf ein Element des Ractive.js Data-Objekt per Mustache	28
4.6	Schleife über Ractive.js Data-Objekt Array per Mustache	28
4.7	Ractive.js beherrscht ebenfalls Bedingungen	29
4.8	Bindung eines Elements des Ractive.js Data-Objekts an ein Oberflächen- element	29
4.9	Verknüpfung eines Oberflächen-Elements mit einer Eventmethode	29
4.10	Methode die nach Event geladen wird	30
4.11	Button Click Event in Android	31
4.12	Button Click Event in JavaScript	31
4.13	Befüllen des Input Forms in Android	31
4.14	Befüllen des Input Forms in JavaScript	32
4.15	JSON Objekt mit Datenbanken	34
4.16	Funktion für GET-Request auf URL	36
8.1	Installation benötigter Abhängigkeiten unter Linux	46
8.2	Node.js Server installation unter Linux	47
8.3	CouchDB installation unter Linux	47

Literatur

- [1] an. *Dokumentation der CouchDB*. <http://docs.couchdb.org/en/1.6.1/> 15.6.2015.
- [2] anonymous. *Alternative Lagerverwaltung herangezogen zur Marktanalyse*. <http://www.sage.de/> 16.06.2015.
- [3] anonymous. *Customizing und Download von Bootstarp*. <http://getbootstrap.com> 02.07.2015.
- [4] anonymous. *Dokumentation der JQuery Referenceimplementierung*. <http://bradley-holt.com/2011/07/couchdb-jquery-plugin-reference/> 25.6.2015.
- [5] anonymous. *Dokumentation und Download des Frameworks fr das SHA1-Hashing*. <https://code.google.com/p/crypto-js/> 20.06.2015.
- [6] anonymous. *Homepage mit Tutorials, in unserem Fall speziell fr JavaScript und JQuery*. <https://www.codecademy.com> 24.04.2015.
- [7] anonymous. *JSDoc Dokumentation*. <http://usejsdoc.org/> 16.07.2015.
- [8] anonymous. *Node Framework Express Dokumentation*. <http://expressjs.com/4x/api.html> 15.06.2015.
- [9] anonymous. *Ractive Dokumentation*. <http://docs.ractivejs.org/latest/get-started> 10.07.2015.
- [10] Mark Wandschneider. *Learning Node.js: A Hands-On Gude to Building Web Applications in JavaScript*. Addison Wesley Verlag, 2013.