Seminararbeit

Simon Egli, Thomas Junghans

**Shopping List**

Einkaufslisten online erstellen und verwalten.

Datum: 09.04.2010

Abgabedatum: 07.07.2010

Studienjahr: 3/4

Fach: PHP und MySQL

Schule: Hochschule für Technik Zürich

Betreuer: Matthias Bachmann

# Vorwort

Die Hochschule für Technik sieht für jedes Fach im 3. Studienjahr eine Seminararbeit im zeitlichen Rahmen von 50 Stunden pro Student vor. Im Fach „PHP und MySQL“ geht es darum eine Website oder Webapplikation mittels den beiden genannten Technologien zu realisieren. Wir haben uns entschlossen eine Webapplikation namens „Shoppinglist“ zu entwickeln. Shoppinglist ermöglicht den Benutzer Einkaufslisten online zu verwalten. Weiter sollen mehrere Personen auf die gleiche Einkaufsliste zugreifen und Gegenstände hinzufügen beziehungsweise als gekauft abhaken können. Diese Szenario entsteht vor allem in einer Wohngemeinschaft mit gemeinsamen Kühlschrank und gemeinsamer Kasse.

Die Applikation soll für Mobilgeräte optimiert werden und wenn möglich ohne Seiten-Refresh auskommen.

# Management Summary

Dieses Dokument ist Bestandteil einer Seminararbeit zum Thema „PHP und MySQL“. Es beschreibt das Vorgehen bei der Planung und Entwicklung und zeigt Probleme und Learnings auf. Zur Planung gehören die Aufgabenstellung, Use Cases und Kriterien. Es werden auch die Werkzeuge aufgelistet, die für die Entwicklung und Zusammenarbeit eingesetzt wurden, wie zum Beispiel Skype und Google Wave für die Kommunikation im Team, Pivotaltracker für die Erfassung von Use Cases und Google Code für die Versionskontrolle.

Das Zusammenspiel zwischen PHP und MySQL soll Anhand einer für Mobilgeräten optimierten Webapplikation gezeigt werden. Die Architektur und verwendeten Technologien werden erwähnt und anhand von Klassen- und ER-Diagramm verdeutlicht.

Am Schluss werden die Hindernisse und Learnings aufgelistet.

Inhaltsverzeichnis

Vorwort 2

Management Summary 2

Aufgabenstellung 4

Pflichtenheft (Simon) 4

Use Cases 4

Use Case 1 4

Kriterien 4

Muss 4

Soll 5

Kann 5

Abgrenzung 5

Planung 5

Milestones 5

Umsetzung 6

Tools 6

Collaboration 6

Kommunikation 6

Applikationsentwicklung 6

Testing und Bugfixing 6

Versionkontrolle – Google Code 6

Hosting 6

Pivotaltracker 6

Technologien (Simon + TJ) 7

Software-Archtiktur 8

Klassendiagram (Simon) 8

ER Diagramm (Simon) 8

GUI und Navigation 8

Kommunikation zwischen View und Model 12

Tests 13

Selenium IDE 13

Probleme 14

DAO Flexibilität (Simon) 14

Polling 14

Autorisierung (Simon) 14

Navigation 14

Learnings 15

AJAX mit PHP und JSON 15

Datenkbank Persistenzlayer (Simon) 16

Datenbankabstraktion (Simon) 16

Quellenverzeichnis 16

# Aufgabenstellung

Es folgt ein Auszug der wichtigsten Punkte aus dem Reglement für Seminararbeiten der HSZ-T.

„Der Aufwand des einzelnen Studierenden für die Bearbeitung der Seminararbeit selber

soll ca. 50 Stunden betragen.

Die Arbeit wird in Form eines kurzen technischen Berichtes auf Papier (A4, weiss, einseitig

bedruckt) abgegeben. Die Sprache ist Hochdeutsch, bei Zustimmung durch den

Betreuer kann die Arbeit auch in Englisch verfasst werden. Zusätzlich werden die Arbeit

als PDF-File und die erstellten weiteren Arbeitsergebnisse auf einem geeigneten

Medium abgegeben (vgl. auch Richtlinie zum erfolgreichen Verfassen einer Diplomarbeit).

Die Arbeit wird mündlich in Hochdeutsch präsentiert. Diese Präsentation trägt zur Beurteilung bei. Anwesend sind dabei: alle am Seminar teilnehmenden Studierenden, der

Dozent, interessierte Personen aus dem Kreise der Dozenten und Studenten des

Fachstudiums Informatik, sowie ggf. weitere Personen auf Einladung der Leitung des

Studiengangs Informatik. Die eigentliche Präsentationszeit soll 20-30 Minuten nicht

übersteigen. Anschliessend an die Präsentation können seitens des Dozenten oder der

Seminarteilnehmer Fragen zur Arbeit gestellt werden. Zur Präsentation ist eine einseitige

Zusammenfassung für alle Seminarteilnehmer mitzubringen.“

# Pflichtenheft (Simon)

## Use Cases

### Use Case 1

## Kriterien

### Muss

Das Produkt muss folgende Anforderungen erfüllen:

* Benutzerregistrierung
* Benutzerauthentifizierung
* Erstellen von Haushalte
* Löschen von Haushalte
* Anzeigen von Haushalte
* Erstellen von Einkaufslisten für einen ausgewählten Haushalt
* Löschen von Einkaufslisten für einen ausgewählten Haushalt
* Anzeigen von Einkaufslisten für einen ausgewählten Haushalt
* Erstellen von Produkten für eine ausgewählte Einkaufsliste
* Löschen von Produkten für eine ausgewählte Einkaufsliste
* Anzeigen von Produkten für eine ausgewählte Einkaufsliste

### Soll

Die Erfüllung dieser Kriterien wird angestrebt:

### Kann

Die Erfüllung ist nicht unbedingt notwendig und sollte nur angestrebt werden, falls noch ausreichende Kapazitäten vorhanden sind.

* Erstellen eines Budgets für eine bestimmte Zeitperiode (z.B. Monat)
* Ausgaben (Summe Items für jeden Einkaufszettel) mit Budget vergleichen
* Der Benutzer sieht ob er zu viel ausgegeben hat oder wieviel er noch ausgeben kann
* Live Update der Shoppinglist GUI. Wenn ein Produkt geändert wurde, sehen das andere User, die zur gleichen Zeit dieselbe GUI betrachten.
* RESTfulness. Ein authentifizierter Benutzer kann eine Ansicht direkt über die URL aufrufen.

### Abgrenzung

Diese Kriterien sollten bewusst nicht erreicht werden:

* Crossbrowserkompatibilität

# Planung

## Milestones

|  |  |  |
| --- | --- | --- |
| Datum  (initial Planung) | Datum  (Realität) | Milestones |
| Mi 17.03.2010 | Mi 17.03.2010 | Kick-Off und Einreichung der Projektidee |
| Sa 10.04.2010 | Sa 10.04.2010 | Planung, Inhaltsangabe, Aufgabenverteilung, Use Cases, Einrichtung Entwicklungsumgebung (IDE, SVN, Pivotaltracker), DokuSkeleton. |
| Sa 26.06.2010 | So 20.6.2010 | Entwicklung fertig |
| Mi 16.06.2010 | Mo 05.07.2010 | Test, Vorwort und Schlusswort |
| Mi 23.06.2010 | Mo 12.07.2010 | Präsentationsvorbereitung |
| Mi 30.06.2010 | Sa 10.07.2010 | Korrekturen, Probedurchlauf |
| Mi 07.07.2010 | Mi 07.07.2010 | Abgabe |
| Mi 14.07.2010 | Mi 14.07.2010 | Präsentation |

# Umsetzung

## Tools

### Collaboration

Für das Ablegen von Notizen haben wir Google Wave eingesetzt. Mit Google Wave können Diskussionen (Waves) geführt werden, die neben Text auch Links, Bilder und andere Medien enthalten können.

### Kommunikation

Für die Kommunikation haben wir Skype verwendet. Mit Skype können Telefonkonferenzen abgehalten werden. Skype ermöglicht auch Desktop-Sharing.

### Applikationsentwicklung

Als IDE für die PHP und Javascript Entwicklung wurde PHPStorm[[1]](#footnote-2) von JetBrains eingesetzt. Ruby On Rails Entwickler kennen RubyMine von JetBrains. RubyMine ist die zurzeit wohl beste IDE erhältlich für die Entwicklung von Rails-Applikationen. PHPStorm basiert auf RubyMine.

### Testing und Bugfixing

Für Navigation- und GUI-Tests wurde Selenium-IDE (Firefox Plugin) eingesetzt. AJAX-Requests- und Javascript-Debugging wurden mittels Firebug (Firefox Plugin) durchgeführt.

### Versionkontrolle – Google Code

Die Versionskontrolle mit SVN wird bei Google Code gehostet[[2]](#footnote-3). Die Versionierung vereinfacht die Zusammenarbeit und dient auch als Backup.

### Hosting

Die Applikation läuft zu Testzwecken unter einer privaten Subdomäne http://shlist.junghans.co.za.

### Pivotaltracker

Pivotaltracker ist eine Onlinetool[[3]](#footnote-4) mit dem Stories (Use Cases) wie man sie aus Scrum kennt auf einem virutellen Story Board verwaltet werden können. Die Stories werden in der *Icebox* erstellt. Hier liegen alle Stories, ob sie nun angenommen werden oder nicht. Angenommene Stories landen im *Backlog*. Stories im Backlog können nach Priorität und Komplexität sortiert werden und in die *Current* Bahn via Drag & Drop gezogen werden. Stories landet letztendlich in der *Done*-Bahn.

Pivotaltracker ist alleine schon wegen der einfachen Bedienung sehr praktisch.



Abbildung GUI von Pivotaltracker mit den vier Bahnen: Icebox, Backlog, Current und Done.

## Technologien (Simon + TJ)

* HTML
* CSS
* Javascript
* PHP
* JSON
* AJAX
* Frameworks
  + jQuery
  + PHP DAO

# Software-Archtiktur

## Klassendiagram (Simon)

## ER Diagramm (Simon)

## GUI und Navigation

|  |  |
| --- | --- |
| :screenshots:IMG_0308.PNG  Abbildung Login | :screenshots:IMG_0309.PNG  Abbildung Registrierung |
| :screenshots:IMG_0310.PNG  Abbildung Übersicht der Haushalte | :screenshots:IMG_0311.PNG  Abbildung Einkaufslisten im Haushalt |
| :screenshots:IMG_0312.PNG  Abbildung Inhalt der Einkaufsliste |  |
|  |  |

Die grafische Oberfläche ist für die Nutzung auf Mobilgeräten optimiert. Die Navigierbarkeit ohne Page-Reloads gibt dem Benutzer das Gefühl, er müsse nie warten bis eine Seite geladen ist. Man spricht von eine sogenannten Single-Page-App[[4]](#footnote-5). Das Aufrufen von anderen Seiten und Nachladen von Inhalten erfolgt über AJAX. Dafür wurde eine Funktion geschrieben, welche die URL der PHP Seite und eine Callback-Funktion als Parameter annimmt.

Definition der Funktion:

Shoppinglist.load\_page = function (options) {

var config = null,

defaults = null;

defaults = {

'beforeLoad' : function () {},

'afterLoad' : function () {},

'page' : 'page.login.php',

'data' : null

};

config = $.extend({}, defaults, options);

config.beforeLoad();

$.ajax({

'url' : config.page,

'type' : 'get',

'data' : config.data,

'dataType' : 'html',

'success' : function (data) {

$('#page').html(data);

config.afterLoad();

location.href = '#page\_' + config.page.split('.')[1];

//log.info('loaded: ' + config.page);

}

});

};

Aufruf der Funktion:

Shoppinglist.load\_page({

'page' : page\_url,

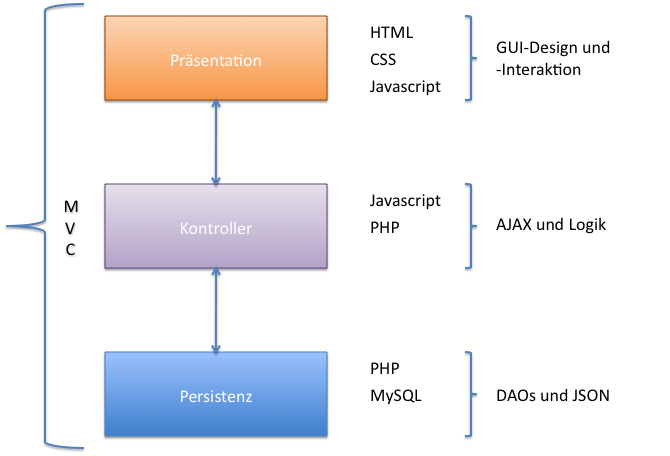
'afterLoad' : function () {

Shoppinglist[module\_name].init();

}

});

## Kommunikation zwischen View und Model

Die Kommunikation zwischen View (Präsentationsschicht) und Model (Persistenzschicht) findet via AJAX-Requests statt. Die View wird mit HTML und CSS dargestellt. Javascript ist für die Interaktivität der GUI und den Empfang der JSON-Daten zuständig. Die Kontrollschicht besteht aus Javascript und PHP. Letzteres wandelt die verarbeiteten Daten aus dem Model in JSON um. 

JSON ist im Vergleich zu XML viel schlanker. Das bedeutet weniger Datenverkehr wird verursacht und die Reaktionsfähigkeit der Webapplikation wird gesteigert. Da JSON eine offizielle Schreibweise von Javascript-Objekten ist, entsteht kein Zusatzaufwand bei der Verarbeitung mit Javascript.

Der Aufbau der AJAX-Request sieht immer gleich aus:

**PHP**

Nachdem die Daten mit PHP verarbeitet wurden, werden sie in JSON umgewandelt. Danach wird der JSON-Code mit den entsprechenden Header ausgegeben.

PHP:

**// Convert to JSON**

$json = json\_encode($data);

**// Set content type**

header('Content-type: application/json');

**// Prevent caching**

header('Expires: 0');

**// Send Response**

print($json);

Javascript:

fetch\_items\_by\_shoppinglist\_id = function (callback) {

$.ajax({

'url' : 'controller\_proxy.php?controller=fetchitems&sid=' + Shoppinglist.selected\_sid,

'type' : 'get',

'dataType' : 'json',

'success' : function (data) {

if (data.items) {

callback(data);

config.onFetch();

} else if (data.message && data.type === 'error') {

callback(data);

config.onError(data);

}

}

});

};

jQuery[[5]](#footnote-6) wird für den Javascript-Part eingesetzt. Die Ajax-Abfragen konnten somit relativ einfach erstellt werden.

# Tests

## Selenium IDE

Unit Tests wurden nicht geschrieben, dafür wurden verschiedene Szenarien wie beispielsweise „Login“ oder „Registration“ mit Selenium IDE[[6]](#footnote-7) aufgenommen. Selenium IDE kann die Aktivitäten des Benutzers simulieren und so ist es einfach einen fehlgeschlagenen oder erfolgreichen Login mit einem Klick durchlaufen zu lassen. Da die Applikation nicht RESTful ist, wurde Selenium DIE auch eingesetzt um mit einem Klick zum Beispiel an die Einkaufsliste zu gelangen, die normalerweise mehrere Klicks via Login, Auswahl des Haushalts und Auswahl der Einkaufsliste benötigt. Auch Formulare können mit Beispieldaten abgefüllt werden, anstatt jedes mal die Daten von Hand eingeben zu müssen.

# Probleme

### DAO Flexibilität (Simon)

### Polling

Die Daten der Itemauflistung werden alle 3 Sekunden aktualisiert. Eine AJAX-Abfrage hohlt die aktuellen Daten in der Datenbank. Wenn zwei oder mehrere Personen zur selben Zeit die selbe Einkaufsliste vor sich haben, sieht jeder die Änderungen des anderen ohne etwas tun zu müssen. Mit der Funktion setTimeout ist dieses Verhalten schnell realisiert. Dabei gibt es ein paar kritische Punkte, welche beachtet werden müssen:

1. Ein GUI-Update darf nicht stattfinden, wenn gerade ein Feld editiert wird. Diese Problematik wurde gelöst in dem die Mausaktivität überwacht wird. Befindet sich der Kursor im Bereich der Eingabemaske wird das Update unterdrückt.
2. Wenn Person A ein Feld editiert, darf Person B dieses Feld nicht gleichzeitig editieren. Dies kann sonst zu Race Conditions[[7]](#footnote-8). Dieser Punkt erfordert eine Sperrung der Ressource, die gerade bearbeitet wird.
3. Es dürfen nicht mehrere AJAX-Abfragen gleichzeitig laufen. Dies kann passieren, wenn eine Antwort länger als 3 Sekunden dauert und setTimeout die Abfrage ein weiteres Mal ausführt bevor die andere beendet wurde. In diesem Fall spricht man ebenfalls von Race Conditions. Lösen kann man das Problem, in dem der setTimeout-Handler immer zurückgesetzt wird (clearTimeout) bevor der nächste Aufruf stattfindet.

### Autorisierung (Simon)

### Navigation

Da es sich um eine single-page-app handelt können einzelne Ansichten nicht direkt abgerufen oder als Favourit gespeichert werden. Nach einem Refresh der Seite landet der Benutzer wird beim Login. Um das RESTful-Verhalten zu erzeugen werden Hashes in der URL verwendet. An diesem Punkt wurde klar, dass ein Einsatz eines Frameworks sinnvoll gewesen wäre, da nicht mehr Zeit für eine eigene Implementation verfügbar gewesen wäre. Lösungsansätze gibt es bereits:

* <http://ajaxian.com/archives/emulating-onhashchange-without-setinterval>
* <http://ceit.uq.edu.au/content/onhashchange-html5-feature-which-can-allow-urls-containing-video-offsets>

# Learnings

## AJAX mit PHP und JSON

Für die Kommunikation zwischen View und Controller wird ausschliesslich AJAX genutzt. Als Datenstruktur eignet sich JSON wegen dem geringen Overhead und dem Zusammenspiel mit Javascript sehr gut.

Für die Javascript und PHP Schnittstelle haben wir uns an das Beispiel 8-5 auf Seite 85 aus dem Buch "Developing Large Web Applications" (Oreilly und Yahoo! Press) angelehnt.

<?php

**// Include libraries**

...

**// Handle the inputs via POST / GET**

$user\_id = $\_SESSION['user']->userId;

$shoppinglist\_id = $\_GET['sid'];

...

**// Assemble data. In this case, retreive data from model.**

if(isset($\_GET['sid']) AND $\_GET['sid'] >= 0) {

$items = DAOFactory::getItemDAO()->queryAllByUserIdAndShoppinglistIdNotClosed($user\_id, $shoppinglist\_id);

} else {

$items = DAOFactory::getItemDAO()->queryAllByUserIdNotClosed($user\_id);

}

if (count($items) > 0) {

$data = array(

'items' => $items

);

}

...

**// Convert to JSON**

$json = json\_encode($data);

**// Set content type**

header('Content-type: application/json');

**// Prevent caching**

header('Expires: 0');

**// Send Response**

print($json);

exit;

## Datenkbank Persistenzlayer (Simon)

## Datenbankabstraktion (Simon)

# Quellenverzeichnis

1. Siehe http://www.jetbrains.com/phpstorm/ [↑](#footnote-ref-2)
2. Siehe http://code.google.com/p/hszt-shoppinglist/ [↑](#footnote-ref-3)
3. Siehe http://www.pivotaltracker.com [↑](#footnote-ref-4)
4. Siehe http://en.wikipedia.org/wiki/Single-page\_application [↑](#footnote-ref-5)
5. Siehe http://api.jquery.com/jQuery.ajax/ [↑](#footnote-ref-6)
6. Siehe http://seleniumhq.org/projects/ide/ [↑](#footnote-ref-7)
7. Siehe http://de.wikipedia.org/wiki/Race\_Condition [↑](#footnote-ref-8)