Seminararbeit

Simon Egli, Thomas Junghans

ShoppingList

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

1 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 Webapplikation mittels den beiden genannten Technologien zu realisieren. Wir haben uns entschlossen eine Webapplikation namens „Shoppinglist" zu erstellen. 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. Dieses 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.

2 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. Dazu gehören beispielsweise 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 Mobilgeräte optimierte Webapplikation gezeigt werden. Die Architektur und verwendeten Technologien werden erwähnt und anhand von Sequenz- und ER-Diagrammen verdeutlicht. Am Schluss werden die Hindernisse und Learnings aufgelistet.

3 Inhaltsverzeichnis

4 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.“*

5 Pflichtenheft

5.1 Use Cases

5.1.1 Use Case 1 - Login / Logout

Ein Benutzer kann sich bei ShoppingList mit seinem persönlichen Account einloggen bzw. ausloggen. Der Zugriff erfolgt mittels einem gültigen Benutzernamen und einem Passwort.

5.1.2 Use Case 2 - Registrierung

Hat ein Benutzer noch keinen Account bei ShoppingList, kann er selbständig einen mit einer gültigen E-Mail Adresse und Passwort anlegen.

5.1.3 Use Case 3 - Zugriff via iPhone

Die Benutzeroberfläche von ShoppingList ist speziell für den Zugriff via iPhone angepasst. Natürlich ist auch der Zugriff via Browser auf einem Computer möglich.

5.1.4 Use Case 4 - Haushalt

Hat sich ein Benutzer erfolgreich authentifiziert, stehen ihm die Funktionen „Haushalt auswählen", „Haushalt erstellen", „Haushalt löschen" und „Einladung für Haushalt versenden" zur Verfügung. Jeder Haushalt hat einen eindeutigen Namen. Einem Haushalt können 1-n Personen angehören.

5.1.5 Use Case 5 - ShoppingList

Innerhalb eines Haushaltes können beliebig viele Einkaufslisten erstellt werden. Ebenso ist es möglich, bestehende Listen wieder zu löschen. Eine Einkaufsliste ist entweder nur für den Eigentümer der Liste ersichtlich oder für alle Benutzer innerhalb des gleichen Haushaltes.

5.1.6 Use Case 6 - Artikel

Einer Einkaufsliste können beliebig viele Artikel hinzugefügt oder gelöscht werden. Jeder Artikel hat einen Namen, einen Preis und eine Anzahl. Während dem Einkaufen sollen Artikel selektiert werden können und die Gesamtsumme soll für den Benutzer ersichtlich sein. Greift ein anderer Benutzer auf die gleiche Einkaufsliste zu, sind selektierte Artikel auch für diesen Benutzer sofort ersichtlich.

5.2 Kriterien

5.2.1 Muss

Das Produkt muss folgende Anforderungen erfüllen:

- Benutzerregistrierung
- Benutzerauthentifizierung
- Erstellen von Haushalten
- Löschen von Haushalten
- Anzeigen von Haushalten
- 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

5.2.2 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 authentisierter Benutzer kann eine Ansicht direkt über die URL aufrufen.

5.2.3 Abgrenzung

Diese Kriterien sollten bewusst nicht erreicht werden:

- Crossbrowserkompatibilität

6 Planung

6.1 Meilensteine

Datum (Initial-Planung)	Datum (Realität)	Meilensteine
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.

7 Umsetzung

7.1 Tools

7.1.1 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.

7.1.2 Kommunikation

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

7.1.3 Applikationsentwicklung

Als IDE für die PHP und Javascript Entwicklung wurde PHPStorm[1] von JetBrains eingesetzt. Ruby On Rails Entwickler kennen RubyMine von JetBrains. RubyMine ist zurzeit die wohl beste IDE erhältlich für die Entwicklung von Rails-Applikationen und PHPStorm scheint den selben Weg gehen zu wollen (PHPStorm war zur Zeit der Entwicklung noch beta).

7.1.4 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.

7.1.5 Versionkontrolle – Google Code

Die Versionskontrolle mit SVN wird bei Google Code gehostet[2]. Die Versionierung vereinfacht die Zusammenarbeit und dient gleichzeitig als Backup.

7.1.6 Hosting

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

7.1.7 Pivotaltracker

Pivotaltracker ist eine Onlinetool[3] mit dem Stories (Use Cases) wie man sie aus Scrum kennt auf einem virtuellen 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. Wenn sie erledigt sind landen Stories in der *Done*-Bahn.

[1] Siehe http://www.jetbrains.com/phpstorm/
[2] Siehe http://code.google.com/p/hszt-shoppinglist/
[3] Siehe http://www.pivotaltracker.com

Pivotaltracker ist alleine schon wegen der einfachen Bedienung sehr praktisch.

Abbildung 1 GUI von Pivotaltracker mit den vier Bahnen: Icebox, Backlog, Current und Done.

7.2 Technologien

Wir haben in unserer Seminararbeit folgende Technologien verwendet. Die Wahl auf viele der Technologien hat sich bereits durch das Fach automatisch ergeben.

Client-seitig:

- HTML, CSS
- Javascript, jQuery

Server-seitig:

- PHP, PHPDao[4]
- Webserver: Apache HTTPD
- Datenbank: MySQL

Kommunikation:

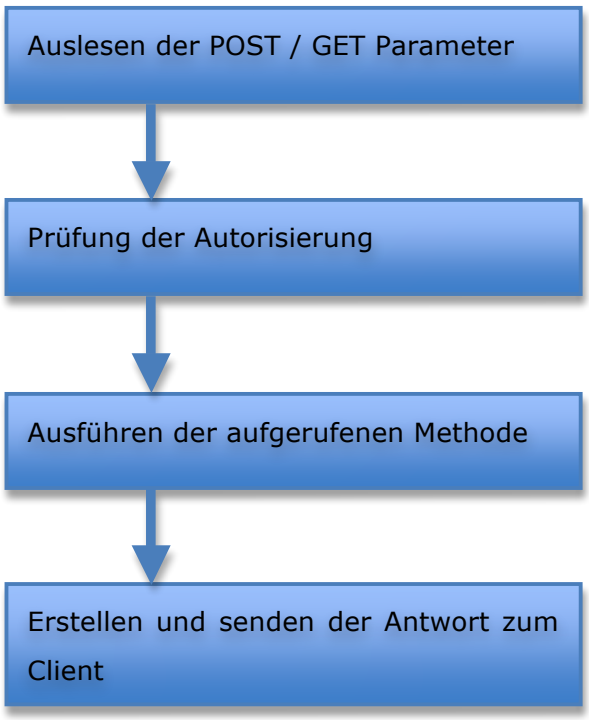
- JSON
- AJAX

[4] Siehe auch: http://phpdao.com/

8 Software-Architektur

Der Aufbau von ShoppingList erfolgt nach den Richtlinien und Prinzipien von Model, View und Control (MVC[5]). Für jede Funktion steht ein separater Kontroller zur Verfügung, welcher den Request, ausgelöst über eine JavaScript Funktion auf dem Client, prozedual bearbeitet. Diese Kontroller greifen auf Persistenzklassen für die Datenbankzugriffe sowie auf diverse Hilfeklassen zu.

Der Ablauf aller Kontroller erfolgt nach folgendem Schema:



Hier ein kleines Beispiel des DeleteItem Controllers:

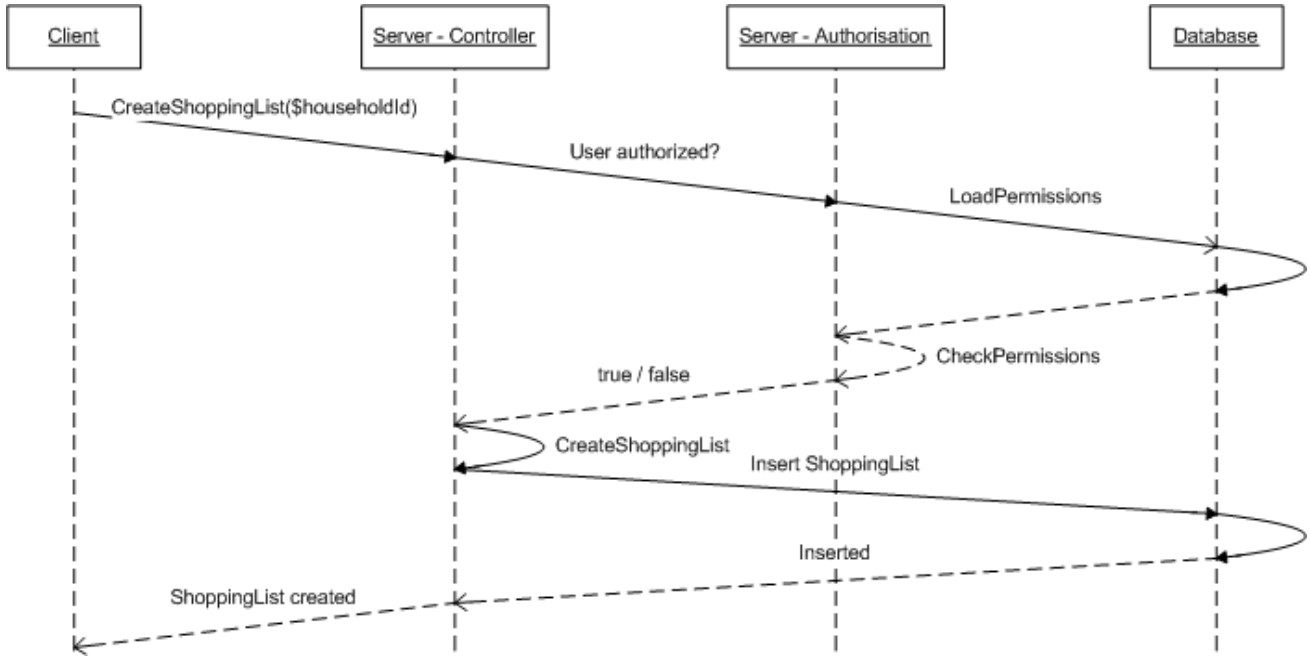
```
<?php
include_once('../config/environment.php');
include_once('lib/include_dao.php');
include_once('lib/message.class.php');
include_once('lib/authorization.class.php');
include_once('controllers/session.controller.php');
include_once('lib/inputvalidation.class.php');

// Read POST / GET variables
$user_id = $_SESSION['user']->userId;
$item_id = $_GET['iid'];

// Check authorization
```

[5] Siehe auch: http://de.wikipedia.org/wiki/Model_View_Controller

```
if(Authorization::auth_delete_item($user_id, $item_id)) {
    // Delete items
    DAOFactory::getItemDAO()->delete($item_id);
    $msg = new Message ('Item deleted', 'info');
    $data = $msg->to_array();
} else {
    $msg = new Message ('Not authorized to delete this item!', 'error');
    $data = $msg->to_array();
}
// 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;
```

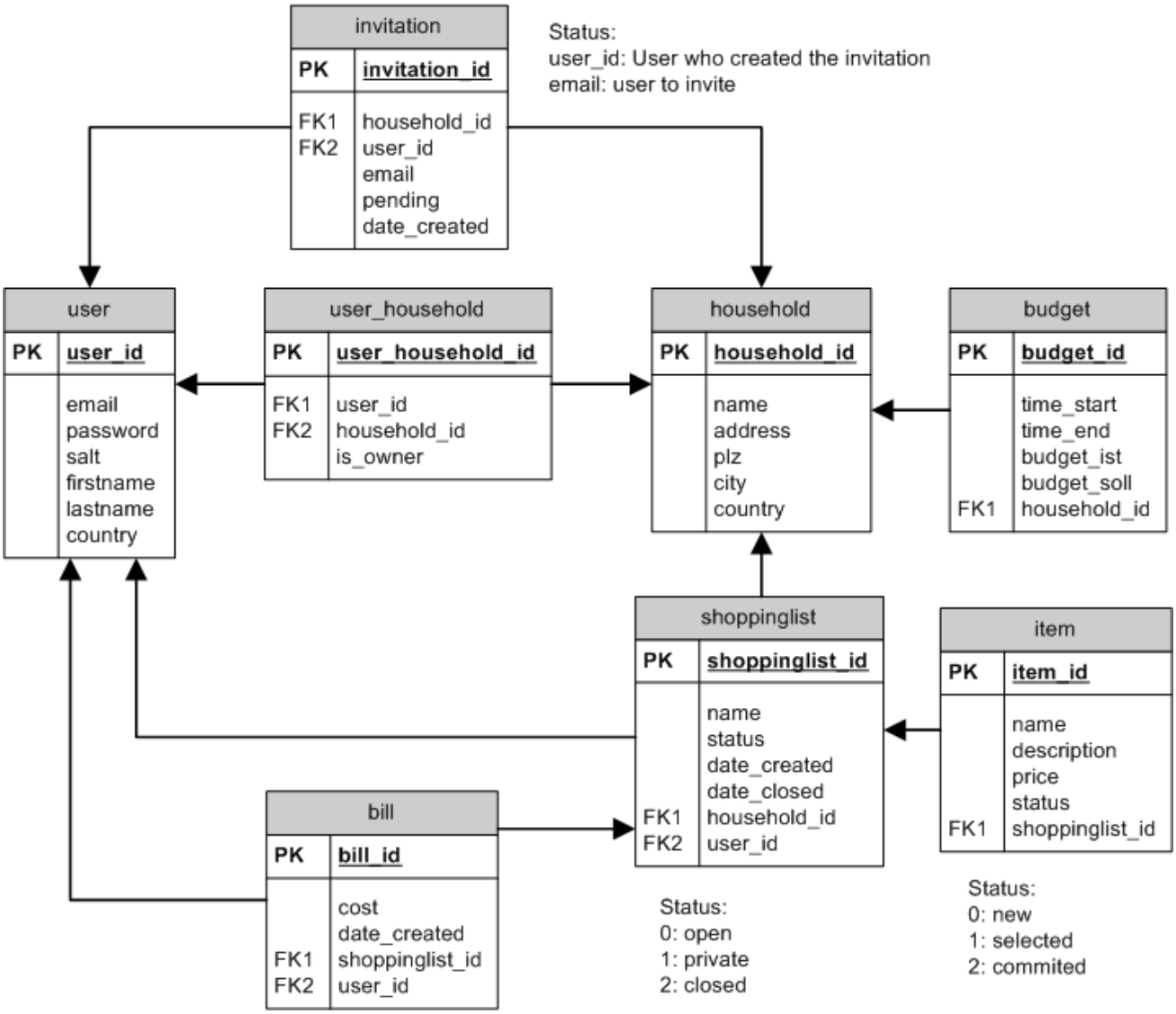
8.1 Sequenz-Diagramm

Folgendes Sequenz-Diagramm beschreibt den Ablauf der Funktion „CreateShoppingList". Es sind alle Funktionen auf dem Server nach gleichem Schema aufgebaut.



8.2 ER-Diagramm

Das Datenbank-Modell von ShoppingList ist wie folgt aufgebaut:



8.3 GUI und Navigation

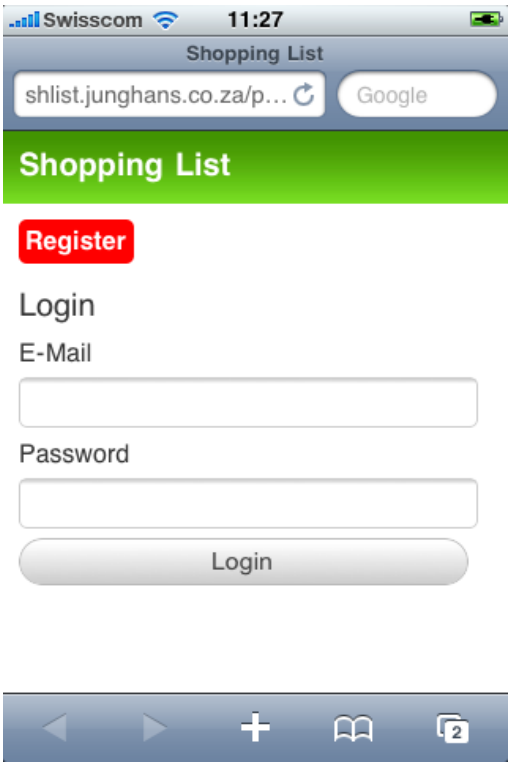


Abbildung 2 Login

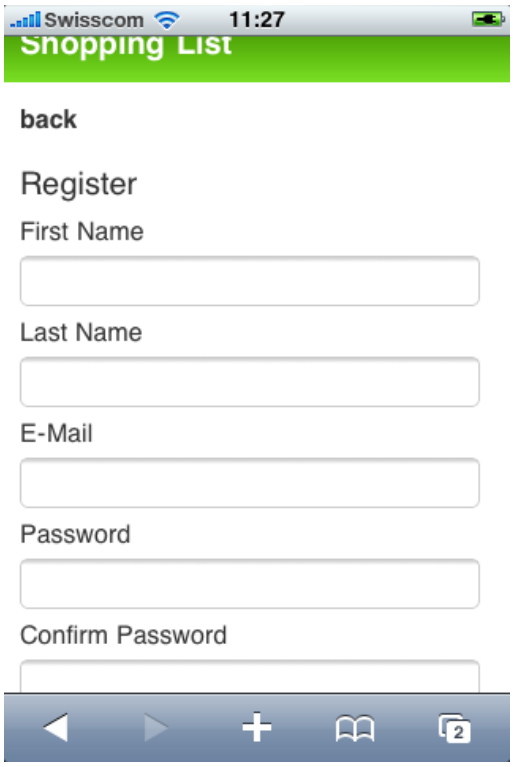


Abbildung 3 Registrierung

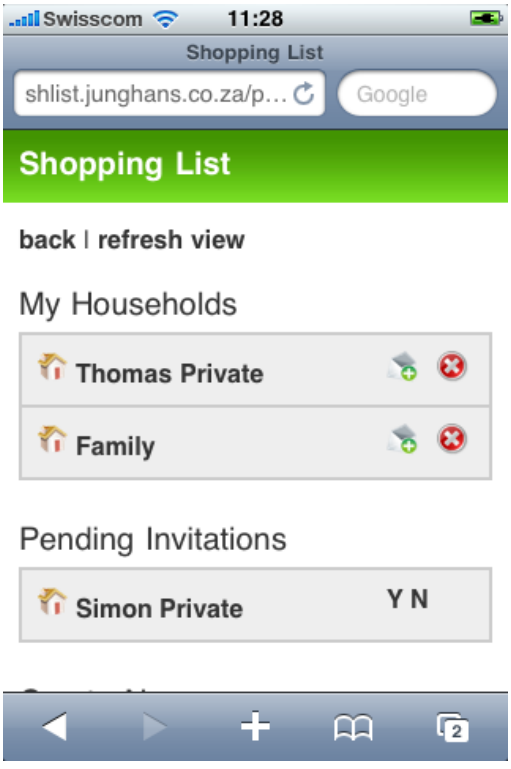


Abbildung 4 Übersicht der Haushalte

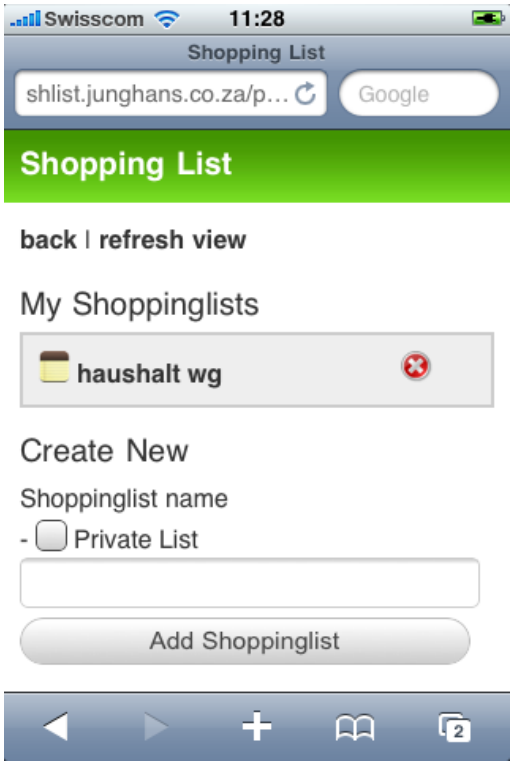


Abbildung 5 Einkaufslisten im Haushalt

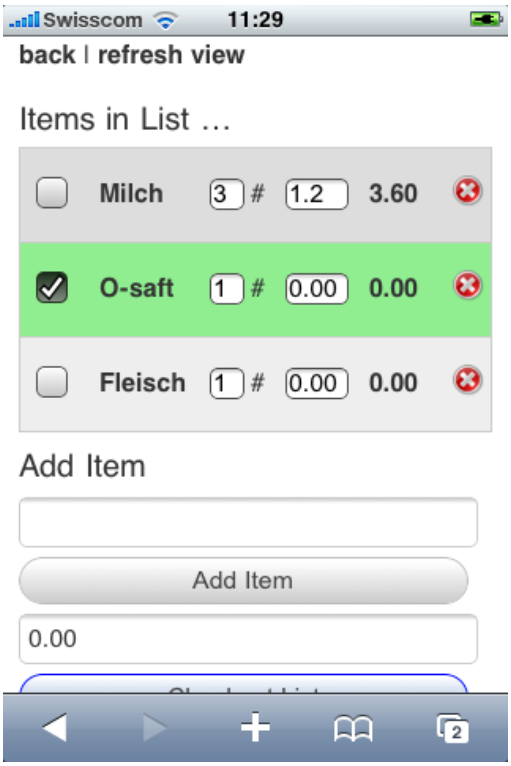


Abbildung 6 Inhalt der Einkaufsliste

Die grafische Oberfläche ist für die Nutzung auf Mobilgeräten optimiert. Die Navigierbarkeit ohne Page-Reload gibt dem Benutzer das Gefühl nie warten zu müssen bis eine Seite geladen ist. Man spricht von einer sogenannten Single-Page-App[6]. 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({
```

[6] Siehe http://en.wikipedia.org/wiki/Single-page_application

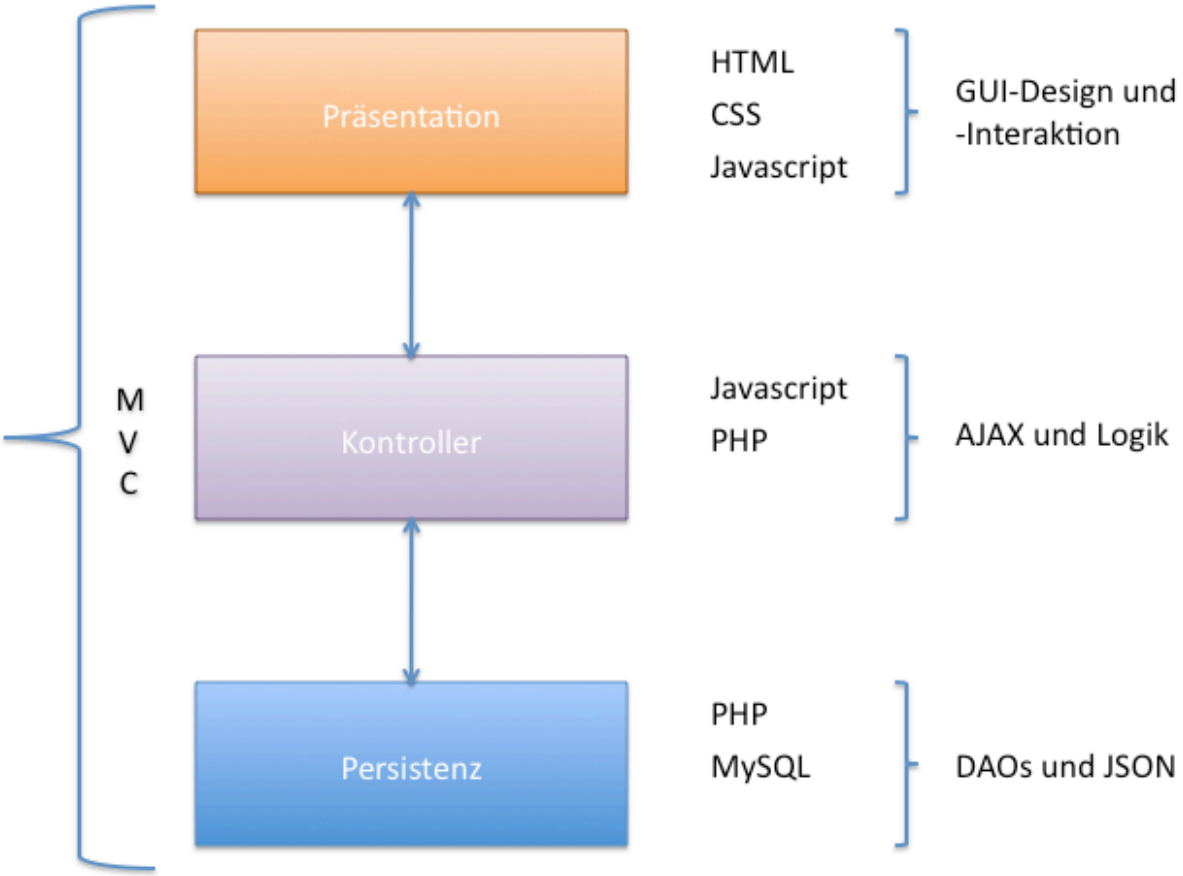
```
        'url' : config.page,
        'type' : 'get',
        'data' : config.data,
        'dataType' : 'html',
        'success' : function (data) {
            $('#page').html(data);
            config.afterLoad();
            location.href = '#page_' + config.page.split('.')[1];
            //log.info('loaded: ' + config.page);
        }
    });
};
```

Aufruf der Funktion:

```
Shoppinglist.load_page({
    'page' : page_url,
    'afterLoad' : function () {
         Shoppinglist[module_name].init();
    }
});
```

8.4 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. Durch die Nutzung von JSON wird weniger Datenverkehr 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:

```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:

```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[7] wird für den Javascript-Part eingesetzt. Die Ajax-Abfragen werden dadurch stark vereinfacht.

[7] Siehe http://api.jquery.com/jQuery.ajax/

9 Tests

9.1 Selenium IDE

Unit Tests wurden keine erstellt. Stattdessen wurden verschiedene Szenarien wie beispielsweise „Login" oder „Registration" mit Selenium IDE[8] aufgenommen. Selenium IDE kann die Aktivitäten des Benutzers simulieren und beispielsweise einen fehlgeschlagenen oder erfolgreichen Login mit einem Klick durchlaufen zu lassen. Da die Applikation nicht RESTful ist, wurde Selenium IDE auch eingesetzt um mit einem Klick zum Beispiel an die Einkaufsliste zu gelangen, die normalerweise mehrere Klicks via den Ansichten „Login", „Auswahl des Haushalts" und „Auswahl der Einkaufsliste" benötigt.
Selenium ist hilfreich beim ausfüllen von Formularen mit Testdaten. Das Formular kann durch Selenium mit einem Klick ausgefüllt und abgeschickt werden.

[8] Siehe http://seleniumhq.org/projects/ide/

10 Probleme

10.1 Flexibilität von PHPDao

Mit Hilfe von PHPDao ist die Implementation eines Persistenzlayers möglich. Nebst den vielen Vorteilen, welche sich dadurch ergeben gibt es auch einige Nachteile. So bietet PHPDao die Möglichkeit anhand des Datenbank-Schemas die Zugriffsklassen und dazugehörigen Datenobjekt-Klassen automatisch zu erstellen. Leider sind danach nur Zugriffe auf einzelne Tabellen möglich. Möchte man Datenbank-Abfragen mit Daten aus mehreren Tabellen tätigen, müssen die Zugriffsklassen manuell angepasst und erweitert werden. Ändert sich das Datenbank-Schema müssen die DAO Klassen neu erstellt werden, was sich als ziemlich mühsam herausstellt.

10.2 Polling

Die Daten der Itemauflistung auf dem Einkaufszettel werden alle 3 Sekunden aktualisiert. Eine AJAX-Abfrage liesst die aktuellen Daten aus der Datenbank. Wenn zwei oder mehrere Personen zur gleichen Zeit die selbe Einkaufsliste offen haben, sieht jeder automatisch die Änderungen des anderen. 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, da die Änderungen sonst überschrieben werden. 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 bearbeiten. Dies kann sonst zu Race Conditions[9]. 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, wenn mehrere Threads parallel laufen. Lösen kann man das Problem, in dem der `setTimeout`-Handler immer zurückgesetzt wird (`clearTimeout`) bevor der nächste Aufruf stattfindet.

[9] Siehe http://de.wikipedia.org/wiki/Race_Condition

10.3 Autorisierung

Das Thema Sicherheit spielt auch in unserer Arbeit eine grosse Rolle. So haben wir bei ShoppingList sichergestellt, dass Angriffe aus dem Internet abgewehrt werden und die Authentisierung sowie Autorisierung sauber implementiert werden. Die grösste Herausforderung dabei war, bei allen Funktionen zu prüfen, ob der Benutzer auch berechtigt ist, diese aufzurufen. Hätten wir diese Prüfung nicht implementiert wäre es möglich gewesen, in fremden Haushalten Einkaufslisten zu erstellen, Artikel von Listen oder ganze Haushalte zu löschen.
Um die Autorisierung sauber zu implementieren haben wir eine zentrale Autorisierungs-Klasse erstellt, welche die Prüfung übernimmt.

Ausschnitt aus einer Autorisierungsprüfung:

```
if(Authorization::auth_create_item($user_id, $shoppinglist_id)) {
        // do something...
}
```

Obenstehende Methode prüft ob der Benutzer XY auf der Shoppinglist XY einen Artikel hinzufügen darf. Dies darf der Benutzer natürlich nur, wenn er sich im selben Haushalt wie die Shoppinglist befindet oder ihm die Shoppingliste gehört.

10.4 Navigation

Da es sich um eine single-page-app handelt können einzelne Ansichten nicht direkt abgerufen oder als Bookmark gespeichert werden. Nach einem Refresh der Seite landet der Benutzer wieder 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

11 Learnings

11.1 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;
```

11.2 Datenbankabstraktion

Die Kommunikation zur Datenbank erfolgt ausschliesslich über den Persistenzlayer von PHPDao. Dieser übernimmt den Aufbau der Verbindung zur Datenbank sowie eine rein objektorientierte Handhabung der Daten. Alle nötigen Klassen für die Datenbankzugriffe werden zudem von PHPDao automatisch erstellt. Den Tipp PHPDao zu verwenden haben wir von einem Kollegen erhalten. Nach einer Analyse von verschiedenen Persistenzlayer Frameworks haben wir uns für PHPDao entschieden. Deswegen, weil PHPDao einfach zu implementieren ist und das Bedürfnis nach einfachen, objektorientierten Datenbankzugriffe erfüllt.

Beispiel: Laden eines „Shoppinglist" Objektes:

```
$shoppinglist = DAOFactory::getShoppinglistDAO()->load($shoppinglist_id);
```

Beispiel: Speichern eines „Item" Objektes:

```
$item = new Item();
$item->name = $item_name;
$item->status = 0;
$item->shoppinglistId = $shoppinglist_id;
$id = DAOFactory::getItemDAO()->insert($item);
```

Beispiel: Daten Beans „Shoppinglist":

```
<?php
	/**
	 * Object represents table 'shoppinglist'
	 * @author: http://phpdao.com
	 * @date: 2010-04-29 01:01
	 */
	class Shoppinglist {
		var $shoppinglistId;
		var $name;
		var $status;
		var $dateCreated;
		var $dateClosed;
		var $householdId;
		var $userId;
	}
?>
```