FERNUNIVERSITÄT HAGEN

Thesis Title

by

Roman Sachse

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty Name
Department or School Name

1. Dezember 2012

Declaration of Authorship

I, AUTHOR NAME, declare that this thesis titled, 'THESIS TITLE' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Write a funny quote here."

If the quote is taken from someone, their name goes here

FERNUNIVERSITÄT HAGEN

Abstract

Faculty Name
Department or School Name

Doctor of Philosophy

by Roman Sachse

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abbreviations

DOM **D**ocument **O**bject **M**odel

Physical Constants

Speed of Light $c = 2.997\ 924\ 58 \times 10^8\ \mathrm{ms}^{-\mathrm{S}}$ (exact)

Symbols

a	distance	m
P	power	W (Js^{-1})
ω	angular frequency	rads^{-1}

For/Dedicated to/To my...

Kapitel 1

Einleitung

1.1 fdfg

käölöjhjfdf

Kapitel 2

Hintergrund/Begriffsklärung

2.1 fdfg

Widgets PLE Klassifizierung nach Palmer Wilson Pattern

Kapitel 3

Anforderungsanalyse

Detaillierte Untersuchung des Problembereichs Theoretische oder praktische Herleitung der Anforderungen Zusammenfassung der Anforderungen (bspw. als Tabelle)

3.1 Use Case

Der folgende Use-Case soll am Ende der Masterarbeit idealerweise abgedeckt werden:

3.1.1 Ziel

Betreuung eines Studienkurses über das Internet

3.1.2 Voraussetzungen

Kursbetreuer/Mentor sitzt in Deutschland, die Studenten in Kamerun. Es steht dort nicht immer ein Internetzugang zur Verfügung. Des weiteren wird oft nach Zeit und nicht nach Volumen abgerechnet, so dass der Nutzer auch bei dem Vorhandensein eines Internetzugangs nicht zwingend online sein muss. Durch die Arbeit an unterschiedlichen Rechnern mit potentiell unterschiedlichen Betriebssystemen, ist die Installation einer komplexen Software nicht ohne Weiteres möglich. Workflow: Betreuer und Studenten stehen über unterschiedliche Kanäle in Kommunikation miteinander. Es müssen Termine geplant, Notizen und Nachrichten hin und hergeschickt werden. Dabei sind die Teilnehmer zu unterschiedlichen Zeiten online. Die Teilnehmer sollen das System offline Nutzen können, um einfache Arbeiten wie das Schreiben von Twitter-Nachrichten, Notizen und Instant- Messaging Nachrichten oder eine Terminabsprache über einen Kalender erledigen können. Bei dem Wechsel zwischen Online und Offline müssen die Daten

synchronisiert werden. Idealerweise haben die Nutzer alle Daten auf einem USB-Stick bei sich und können so von unterschiedlichsten Rechnern, wie beispielsweise in der Universität, im Internetcafe oder zu von Hause aus, arbeiten. Konzeption: Die Arbeit wird sich in die folgenden zwei Bereiche aufteilen:

1. Konzeption, Design und Implementierung eines leichtgewichtigen auf Html5 und Javascript basierenden Dashboards.

2. Konzeption und prototypische Implementierung eines oder mehrerer Workflows, die es erlauben mit den Widgets zu kommunizieren und zumindest Teile der Services zu nutzen, auch wenn das System (also der Browser) offline ist. Wird die Internetverbindung wiederhergestellt sollen vorgenommene Änderungen und Arbeiten mit dem Onlineservice synchronisiert werden.

Kapitel 4

Übersicht Stand der Forschung

Darstellung existierender Anstze (ggf. Klassifikation dieser Ansätze) Analyse der Anstze in Bezug auf die Anforderungen Zusammenfassung der Defizite des Stands der Forschung

Offline Apps, Widgets, unterschiedliche Formate, , Cors, Cross Origin Policy http://enable-cors.org/

neuerungen HTML5 Appcache HTML5

Opensocial vs W3C warum w3c widgets, opensocial wird nicht weiterentwickelt etc, Apache Shindig Problematishc W3C Widgets haben keine gesonderte Definition für HTML5 Appcache

4.0.3 REST

Rest

4.0.3.1 Widgetparameter

Die Parameter für Widgets werden in der Wookie DB hinterlegt. Die Widget Implementation wird über http://localhost:8080/wookie/shared/js/wookie-wrapper.js ausgeliefert. Hier ist setItem/getItem ist so implementiert, dass es einen Request an den Server sendet und die Einstellungen speichert.

Die Standardeinstellungen aus der config.xml werden nur beim ersten deploy ausgelesen, anschließend nicht! mehr

Jedes Widget bekommt eine eigene id! Diese kann dann für die Identifizierung genutzt werden.

Die Werte werden NICHT im local storage hinterlegt.

Kapitel 5

Lösungsansatz

Beschreibung, wie die eigene Lösung die Anforderungen erfüllt Konzeptioneller (holistischer) Überblick über die eigene Lösung

5.1 Überblick

Symfony2, Angular Beschreibung, Apache Wookie Beschreibung, Twitter Bootstrap

Kapitel 6

Details zur Lösung

Implementierungsdetails (nur die interessanten) Erfahrungen / Evaluation

6.1 Entwicklungsumgebung/Tools

Für die Entwicklung von Plesynd wurden unterschiedliche Tools und Frameworks genutzt. Entwickelt wurde das System auf Serverseite mit PHP[1]. Der Clientseitige Code wurde mit JavaScript[2] implementiert.

Die Frameworks im Folgenden kurz vorgestellt.

6.1.1 Symfony2

Symfony2[3] ist ein in PHP implementiertes komponenten-basiertes full-stack Framework zur Entwicklung von MVC(Model-View-Controller) Anwendungen. Zu den bereitgestellten Komponenten gehört beispielsweise ein Service Container [4]. Dieser erlaubt es die von den genutzten Klassen benötigten Abhängigkeiten zu definieren. Der Container sorgt dann für ein Instantiieren der Abhängigkeiten und ein Injizieren dieser zur Laufzeit (TODO: Dependendency-Injection erklären?). Desweiteren bietet Symfony2 unter anderem eine eigene Template-Engine (TODO: Twig Link), sowie ein Event-Listener System, welches ermöglicht auf bestimmte Events (TODO: Request Beispiele) zu reagieren. Das Model wird in Symfony2 standardmäßig über Doctrine2 (siehe 6.1.4) abgebildet. Die Komponenten werden in Symfony2 über sogenannte Bundles implementiert (TODO:

[1] http://php.net
[2] http://de.wikipedia.org/wiki/JavaScript
[3] http://symfony.com
[4] http://symfony.com/doc/current/book/service_container.html

Bundle Link). Bundles sind von dem System unabhängige Plugins, mit einer festen Struktur, welche einfach in eine Applikation eingebunden werden können. Das besondere an Symfony2 ist, dass das gesamte System aus Bundles aufgebaut ist. Hierzu gehören also auch das Kern-Framework, alle Komponenten, zusätzlicher Third-Party-Code, sowie die eigene Applikation selbst. Die für die Applikation entwickelten Bundles werden in 6.1.2, die Fremd-Bundles in 6.1.3 beschrieben

6.1.2 eigene Bundles

Wichtig: Bundles, was sind Bundles, System komplett aus Bundles aufgebaut. einfaches einbinden von Third Party Bundles. Plesynd besteht aus 3 Bundles: Plesynd, Wookie-Connector, User (welches wiederum vom FOSUserbundle ableitet. Benutzt noch weitere Bundles: FOSRest (mit Verweis zu Rest Erklärung), NelmioCorsBundle

6.1.3 Fremd-Bundles

6.1.4 Doctrine2

Orm, DataMapper Pattern, kein Active Record, Arbeit mit einfachen Objekten, Entity Manager Unit of Work

6.1.5 AngularJS

6.1.6 Jquery

Jquery[5] ist eine JavaScript-Bibliothek zur einfachen Manipulation des DOM. Desweiteren stellt sie erweiterte JavaScript-Funktionalitäten zur Verfügung und vereinfacht die Arbeit mit den browserbasierten Event-System(TODO: ref http://de.wikipedia.org/wiki/JQuery) AngularJS verwendet Jquery, wenn vorhanden, insbesondere zur Manipulation des DOM. In Plesynd basiert unter anderem noch das Postmessage(TODO: ref postmessage) auf Jquery.

6.1.7 Twitter Bootstrap

Twitter Bootstrap[6] ist ein von Twitter entwickeltes Framework zur schnellen und einfachen Entwicklung von Frontends. Es stellt CSS-Vorlagen und JavaScript-Komponenten

[5] http://jquery.com/
[6] http://twitter.github.com/bootstrap/

zur Verfügung, welche es dem Entwickler ermöglichen sollen in kurzer Zeit ein User-Interface zu entwerfen und umzusetzen. Bootstrap beinhaltet CSS-Vorlagen für Grids, Tabellen, Buttons etc. . In Plesynd wird von den von Bootstrap zur Verfügung gestellten Javascript-Komponenten momentan nur die Modal-Komponente[7] verwendet.

6.1.8 Apache Wookie

6.2 Implementierungsdetails

6.2.1 Ablauf holen der Daten

WookieConnectorBundle Der Loginname connection getUser() setLoginName(demo2); ist dafür verantwortlich, dass Wookie eine neue Widgetinstanz erstellt oder eine bestehende zurück liefert.

Man muss sich jetzt fragen, ob es möglich sein soll, dass jeder Workspace die selben Instanzen von Widgets hat oder nicht. Wenn ja, kann einfach die User Id aus Plesynd als identifier genommen werden, wenn nicht, sollte zum Beispiel eine Kombination aus UserId und WorkspaceId als Identifier benutzt werden.

==¿ eigener unique Identifier beim Widget jedes Widgets bekommt seinen eigenen localStorage über window.name prefix

6.2.2 Api der Offline Services

6.2.3 Preloads der Widgets

Es ist notwendig, dass alle widgets, aller workspaces bei dem ersten aufruf geladen werden. Gründe:

es müssen alle appcache dateien runtergeladen werden, ohne, dass man sich durch alle workspaces klickt, das Dashboard benötigt die Infos aller Widgets zur Ausgabe

Wie kann dies umgesetzt werden?

Ich habe keine Wahl als alle iframes immer zu laden und ihre sichtbarkeit zu ändern. Dies wird mir wahrscheinlich ziemlich Probleme mit dem drag and drop geben, aber es geht nicht anders.

[7] http://twitter.github.com/bootstrap/javascript.htmlmodals

Insbesondere beim online/offline Wechsel müssen *alle* iframes aktualisiert werden. Durch den Workspace Wechsel ist es allerdings so, dass angular bei der container arbeitsweise, die iframes immer komplett aus dem dom entfernt. dies bringt dann natürlich gar nichts, da man sich dann immer durch alle Workspaces bewegen müsste

6.2.4 Widgets auf Hauptebene

Worskpace =¿ Widget

Eigentlich ist es so gedacht, dass nur die Hauptentities im LocalStorage gespeichert werden. Die Widgets würden dann in ihren Workspaces liegen. Das Storage/Resource System kann nur mit Hauptdatensätzen arbeiten, es ist nicht möglich aus einem Workspace ein Widget zu löschen. Der Workspace muss geholt werden, der Eintrag gelöscht werden und der Workspace muss wieder geschrieben werden Probleme:

Wie kann man local mit Subentities arbeiten, wenn man keinen direkten Zugriff auf sie hat, man muss immer über die Workspaces gehen Als RestService wäre es kein Problem direkt Widgets zu löschen etc, aber dies geht nicht ohne weiteres im Local Storage Der Local Storage muss aber geupdatet werden, damit das System auch offline die richtigen Daten hat Bei angular Ressourcen kann es nur eine URL geben, dies macht den Rest Zugriff problematisch.

Widgets als eigenes Entity speichern? also quasi eine Ressource für Widgets und eine für Workspace?

Das System so umbauen, dass es freier entscheiden kann, was wie in welchen Storage geschrieben werden kann?

Nicht mehr mit Resourcen arbeiten?

IndexedDb?

Ich arbeite jetzt so, dass die Widgets in ihrem eigenen Local Storage gespeichert werden. Sie werden dann für die Ausgabe je nach Workspace gefiltert

6.2.5 Probleme bei multiple Instanzen des selben Widgets

Es ist möglich, dass das selbe Widget mehrfach in einer Plesynd-Instanz verwendet wird. Beispielsweise könnte das TodoList-Widget auf mehreren Workspaces verwendet werden, um je nach Kontext unterschiedliche Todo-Listen zu verwalten. Es können hierbei jedoch Synchronisationsprobleme auftreten, wenn beide Widget-Instanzen offline auf der

selben Datenbasis operieren. Der Grund dafür liegt darin, dass die POST-Methode nicht idempotent ist (siehe 4.0.3). Da die unterschiedlichen Widget-Instanzen nicht in Kommunikation miteinander stehen bedeutet dies im Falle einer Synchronisierung, dass beide Instanzen lokal hinzugefügte Datensätze per POST an den Server schicken. Somit entstehen nicht gewollte Dopplungen der Datensätze. DELETE und PUT Aufrufe stellen für diesem Fall keine Probleme dar, da diese Methoden idempotent sind und ein doppelter DELETE Request an eine Resource keine negativen Auswirken hat. Dieses Problem wurde gelöst, in dem jede Widget-Instanz seinen eigenen Local-Storage zur Speicherung seiner Daten erhält. Der Name des Local Storage ergibt sich aus dem internen Namen des Widgets (beispielsweise todo) konkateniert mit dem Namen des IFrames in dem das Widge aufgerufen wird.

Es ist natürlich möglich, dass unterschiedliche Widget-Instanzen zwar unterschiedlichen Local Storage benutzen, jedoch auf der selben Datenbasis arbeiten (beispielsweise die selben Todo-Listen verwenden). Da momentan noch keine Inter-Widget Kommunikation stattfindet kann es hierbei zu Problemen kommen, wenn im Offline Betrieb beispielsweise ein Datensatz in einer Instanz gelöscht wird, während er in einer anderen bearbeitet wurde. Dieses Problem wurde im Rahmen dieser Arbeit nicht bearbeitet und sollte eventuell in Folgearbeiten gelöst werden.

6.2.6 Cors

Beschreiben wie Cors umgesetzt, Das NelmioCorsBundle6.1.3 wird auf Serverseite eingesetzt um die Arbeit mit Cors zu erleichtern. Es ermöglicht die erlaubten Header über Config Einstellungen (TODO: beispiel) zu definieren und

6.2.7 Kommunikation der IFrames

Postmessage:

Kapitel 7

Zusammenfassung/Ausblick

Kurzfassung des Problems und wie es gelöst wurde Usb-Stick, mobile Version der Browser

7.0.8 Offene Fragen für weitere Forschungsarbeiten

Änderung von HttpAuth zu Token Basierter Authentifizierung Widget Authentifizierung Stabile Wookieversion benutzen Websockets Serverpush

Anhang A

Installation/Deployment

Write your Appendix content here.

A.1 Installation

A.1.1 Wookie

Die Datei local.widgetserver.properties muss direkt im Wookie Dir angelegt werden. Achtung, hier kann nichts hinterlegt werden, auf das schon der Apache hört (also port 80) Install etc

A.1.2 Plesynd

A.2 Deployment

A.2.1 Plesynd

Literaturverzeichnis