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

8. Januar 2013

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 G [1]

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, ,

4.0.3 CORS

Moderne Browser benutzen als Teil ihres Sicherheitskonzeptes die Same-Origin-Policy. Diese bewirkt, dass es Sprachen, die auf Clientseite ausgeführt werden (wie JavaScript) nicht erlaubt ist, Request an einen anderen Zielpunkt als ihrem Ursprung zu starten[2]. Diese Policy wird also lediglich bei Zugriff auf URLs mit der selben Domain und dem selben Port, wie die URL von der die Seite geladen wurde, erfüllt. Das bedeutet, dass ein Skript auf `http://sop.example.com/directory1` Requests an `http://sop.example.com/directory2` starten kann, nicht jedoch an `http://example.com/directory` (unterschiedliche Domain) oder an `http://sop.example.com:8080/directory2` (unterschiedlicher Port). Ausgenommen sind hierbei das in eine Seite eingebettete Laden von Resourcen. Hierzu gehören externe Inhalte, die über iFrames geladen werden aber auch externe JavaScript-Dateien(über ¡script¿...¡/script¿ Tags) und Medienresourcen wie Bilder und Videos. Des Weiteren ist es auch möglich Formulare an andere Zielpunkte als den Ursprung abzuschicken. Diese Einschränkung also primär Auswirkungen auf das Absenden von XMLHttpRequest, also auf normale Ajax-Requests.

Dies Einschränkung ist sehr sinnvoll um beispielsweise das Ausspähen privater Daten zu verhindern. Sie erschwert jedoch die Entwicklung moderner Ajax-Anwendungen und insbesondere die Entwicklung von Mashup-Applikationen wie PLEs, welche von prinzipiell schon so aufgebaut, dass sie ihre Inhalte und Resourcen aus unterschiedlichen Quellen beziehen. In einer PLE wie in dieser Arbeit beschrieben, ist es beispielsweise

so, dass das Widget (siehe **??**) selber von einem Widget-Container wie Wookie (siehe 6.1.8) ausgeliefert werden, also die Domain des Widget Containers als Origin besitzen. Arbeiten diese Widgets nun aber nicht nur lokal beim Client, sondern benötigen für ihre Funktionalität auch einen externen Server, so müssen sie in der Lage sein XMLHttpRequests an diesen zu senden. Aus diesem Grund wurde der Mechanismus des Cross-Origin Resource Sharing[3] eingeführt. Dieser erlaubt es unter bestimmten Bedingungen und Einschränkungen die Same-Origin-Policy zum umgehen.

Ein einfacher Cors-Request vom Client zum Server sieht wie folgt aus (Workflow analog zu [4]):

Der Client sendet eine Cross-Origin-Anfrage mit einem Origin Header an den Server:

```
GET /cors HTTP/1.1
Origin: http://api.bob.com
Host: api.alice.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

Anschließend antwortet der Server mit:

```
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: FooBar
Content-Type: text/html; charset=utf-8
```

Alle für den CORS-Request relevanten Header beginnen mit Access-Control. `Access-Control-Allow-Or` bedeutet, dass der Server eine Cross-Origin-Anfrage von dem angegebenen Origin erlaubt, `Access-Control-Allow-Credentials: true` besagt, dass in diesem Request auch Cookies erlaubt sind. Möchte der Client Zugriff auf Nicht-Standard-Header aus der Antwort des Servers, müssen diese in `Access-Control-Expose-Headers` angegeben werden.

Sollte der Client einen Request mit einer anderen Methode als `GET` oder `POST` (siehe **??** senden, reicht dieser einfache Workflow nicht aus. In diesem Fall muss vor der eigentlichen Anfrage ein so genannter „Preflight-Request“ ablaufen, welcher verifiziert, dass der Server diese Methode als CORS-Request erlaubt.

Zuerst wird vom Client eine Anfrage mit der `OPTIONS`-Methode durchgeführt, welche den folgenden Request authentifizieren soll:

```
OPTIONS /cors HTTP/1.1
Origin: http://api.bob.com
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: X-Custom-Header
Host: api.alice.com
Accept-Language: en-US
Connection: keep-alive
```

```
User-Agent: Mozilla/5.0...
```

`Access-Control-Request-Method` gibt hierbei an, welche Methode genutzt werden soll, `Access-Control-Request-Headers` informiert den Server über zusätzlich zu erwartende Header.

Der Server antwortet beispielsweise mit:

```
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: X-Custom-Header
Content-Type: text/html; charset=utf-8
```

Der Preflight-Request ist nur erfolgreich, wenn die Methode aus `Access-Control-Request-Method` in `Access-Control-Allow-Methods` und alle Header aus `Access-Control-Request-Headers` in `Access-Control-Allow-Headers` vorhanden sind.

4.0.4 HTML5

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

4.0.5 REST

Rest

4.0.5.1 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

5.1.1 Unabhängigkeit Offline/Online

Local Storage, Appcache, Minification, warum kein Indexed DB

Drop in Rest oder Local

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?). Des weiteren 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.

[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

Die Komponenten werden in Symfony2 über sogenannte Bundles implementiert (TODO: 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.

[5] http://jquery.com/

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 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 Umsetzung der Offline-Fähigkeiten

6.2.2.1 Local Storage

Drop In Rest Benutzen von Services, Controller bekommen das nicht mit

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

6.2.2.2 Einschränkungen im Offline-Betrieb

Trotz der Offline Fähigkeiten des System stehen im Offline-Betrieb nicht alle Funktionalitäten zur Verfügung. Hauptaugenmerk wurde auf die Weiterbenutzbarkeit der Widgets im Offline-Modus gelegt. Im Folgenden sind einige Funktionen aufgelistet, welche während des Offline-Betriebes deaktiviert werden.

- Hinzufügen, Bearbeiten und Löschen von Widgets
- Hinzufügen, Bearbeiten und Löschen von Workspaces
- Drag and Drop von Widgets
- Anlegen und Bearbeiten von Todo-Listen im Todo-Widget

Die Funktionalitäten werden meist einfach ausgeblendet, sobald das System feststellt, dass es sich im Offline-Modus befindet. (TODO: Codebeispiel)

6.2.2.3 Preloads der Widgets

Plesynd muss den User in die Lage versetzen nach einmaligem Laden des Systems offline weiterzuarbeiten. Dies ist nur möglich, wenn alle Daten beim erstmaligen Aufruf geladen werden. Insbesondere gilt dies für die IFrames der Widgets. Es müssen die Appcache-Dateien der einzelnen Widgets geladen werden und das Dashboard benötigt die Infos der Widgets zur Ausgabe der zusammenfassenden Informationen direkt bei Systemstart. Der Nutzer soll sich nicht erst durch alle vorhandenen Workspaces bewegen müssen, um die Daten aller Widgets zu laden. Aus diesem Grund müssen die IFrames aller Widgets aller Workspaces bei dem ersten Aufruf im DOM vorhanden sein. Des weiteren dürfen die IFrames bei Wechsel zwischen den Workspaces nicht wieder aus dem DOM entfernt. Der Grund hierfür ist, dass bei Wechsel des Online-Status alle IFrames aktualisiert werden müssen und nicht nur diejenigen des aktuellen Workspaces.

Aus diesen Gründen wurden wurde entschieden alle IFrames bei Systemstart direkt zu laden und im DOM vorzuhalten. Es findet lediglich eine Filterung auf Basis des anzuzeigenden Workspace statt, welche der IFrames ausgegeben werden und welche ausgeblendet werden (TODO: Codebeispiel)

6.2.3 Widgets auf Hauptebene

Die ursprüngliche Archtiektur von Plesynd sah vor, dass der Local Storage nur direkten Zugriff auf die Hauptdatensätze liefert. Dies dies hätte zur Folge, dass nur Workspaces

direkt referenziert werden könnten. Der Zugriff auf die Widgets würde dann über ihre Workspaces erfolgen. Das Storage/Resource System kann jedoch nur mit Hauptdatensätzen arbeiten. Dieses Vorgehen hat jedoch einige Probleme und Fragen nach sich gezogen:

- Wie kann man lokal mit Subdatensätzen arbeiten, wenn man keinen direkten Zugriff auf sie hat? Es ist nicht ohne weiteres möglich aus dem Local Storage eines Workspaces ein Widget zu löschen oder zu bearbeiten. Der Workspace müsste geholt, das Widget in dem Workspace gelöscht/bearbeitet und der Workspace wieder geschrieben werden.
- Als Rest Service wäre es kein Problem direkt Widgets zu löschen oder zu bearbeiten, aber dies geht nicht ohne weiteres im Local Storage. Der Local Storage muss aber geupdatet werden, damit das System auch offline mit den korrekten Daten arbeitet.

Es gibt mehrere Möglichkeiten dieses Problem zu lösen:

1. Umstellung auf eine andere Lösung zur lokalen Speicherung der Daten (z.B. IndexedDb)
2. Erweiterung der Implementierung zur Speicherung der Daten im Local Storage, so dass auch Subeinträge gefunden und bearbeitet werden können.
3. Speicherung der Subdatensätze auf der Hauptebene. Alle direkt über die REST-Schnittstelle ansprechbaren Resourcen werden auf der Hauptebene im Local Storage hinterlegt

Punkt **??** sollte vermieden werden, da er eine grundsätzliche Änderung der Systemarchitektur nach sich gezogen hätte (TODO: ref auf Entscheidung für Local Storage). Punkt 2 kam in die engere Betrachtung wurde jedoch verworfen, da er die Komplexität der Speicherung in den Local Storage beträchtlich erhöht hätte. Wie in 5.1.1 beschrieben, sollte es für die Implementierung der Logik unerheblich sein, ob das System zum Zeitpunkt einer Nutzeraktion online oder offline ist. Um dies mit Punkt 2 zu erreichen, hätte die in 6.2.2.1 vorgestellte Vorgehensweise grundlegend geändert werden müssen. Aus diesen Gründen wurde Punkt 3 umgesetzt. Jede REST-Resource wird direkt auf der Hauptebene abgelegt. Somit ergibt sich eine einfach 1:1 Abbildung der Resourcen auf den Local Storage.

6.2.4 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 4.0.5). 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. Der Name des IFrames ergibt sich aus der ID, welche Wookie für jedes Widgets erstellt. Diese ID ist einzigartig und verändert sich auch bei wiederholtem Aufrufen nicht. Somit ist sichergestellt, dass jedes Widget einen festen Local Storage erhält.

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 bearbeitet werden.

6.2.5 Drag'n Drop

Zur Umsetzung des Drag and Drop Funktionalität zum Sortieren der Widgets auf einem Workspace wurde das Sortable Widget[8] der JqueryUi-Bibliothek[9] verwendet. Diese wurde in einer eigenen AngularJS-Direktive gekapselt, so dass das JqueryUi-Widget in der bestehenden Architektur verwendet werden konnte. Die Direktive verwendet den Widget-Service(TODO: Ref), um die neue Position der Widgets dem Server mitzuteilen

[8] http://jqueryui.com/sortable/
[9] http://jqueryui.com

und den OnlineStatus-Service (TODO:ref), um die Drag and Drop Funktionalität im Offline-Modus zu deaktivieren

(TODO: Codebeispiel)

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 zwischen den IFrames

(TODO: Bib: https://developer.mozilla.org/en-US/docs/DOM/window.postMessage) 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 Änderung der Click Events auf Touch Events

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

[1] Michael C. Grant and David Carlisle. The PSFrag system, version 3. CTAN: tex-archive/macros/latex/contrib/%supported/psfrag/pfgguide.tex, November 1996.

[2] Jesse Ruderman. Same-Origin-Policy. https://developer.mozilla.org/en-US/docs/JavaScript/Same_origin_policy_for_JavaScript Abrufdatum: 08.01.2013, 2012.

[3] Anne van Kesteren. Cross-Origin Resource Sharing. CTAN: http://www.w3.org/TR/cors/ Datum: 08.01.2013, April 2012.

[4] Monsur Hossain. Using CORS. http://www.html5rocks.com/en/tutorials/cors/ Abrufdatum: 08.01.2013, September 2012.