Hochschule für Telekommunikation Leipzig (HfTL)

Profilierung Netzbasierte Anwendungen

Profilierung Mobile Applikationen

Softwaredokumentation

TaskY - Cache und Notifications in mobilen Webanwendungen

David Howon (147102)

Michael Müller (147105)

Wintersemester 2016/17

Hochschule für Telekommunikation Leipzig
University of Applied Sciences

Inhaltsverzeichnis

1 Einleitung: Motivation und Ziele

Diese Dokumentation entstand im Rahmen der Profilierungen „Mobile Applikationen“ und „Netzbasierte Anwendungen“ im Wintersemester 2016/17 an der Hochschule für Telekommunikation Leipzig (HfTL).

Projektbericht - Bestandteile
- Motivation und Ziele
- Grundlagen
- Anforderungen
- Konzeption (MVC, Methodik, + Alternativen)
- Implementierung
- Zusammenfassung und Ausblick

... Einleitung moderne webtechnologien –> webapps statt nativen Apps ...
... Beschreibung der Aufgabe/des Problems ...

... Versuch der Lösungsfindung/Kurzbeschreibung Projekt ...

2 Grundlagen

- grobe Einführung
- eventuell Grundkonzept
- Welche Push Arten gibt es?

a) Websocket/ServerSent Events

b) PUSH-API

- Welche Datenänderungen auf Serverseite pusht man mit welcher Variante?

Viel müssen wir hier nicht schreiben...Grundlagen sollten eigentlich allen bekannt sein.

3 Anforderungen

3.1 allgemeine Beschreibung der Applikation

Nach erfolgreicher Registrierung und Anmeldung kann der Benutzer Aufgaben anlegen, bearbeiten, anzeigen und löschen. Weiterhin gibt es eine Kontaktliste, in welcher alle Kontakte angezeigt werden, die ebenfalls für die Anwendung registriert sind und zu persönlichen Kontakten hinzugefügt wurden. Aufgaben können mit persönlichen Kontakten geteilt werden. Ebenso ist es möglich Gruppen anzulegen, dieser Kontakte hinzuzufügen und Aufgabe mit der Gruppe zu teilen.

Über Änderungen an Gruppen oder Aufgaben wird der Benutzer über PUSH-Benachrichtigungen informiert. Wenn einer Aufgabe ein Benachrichtigungszeitpunkt angegeben wurde, wird ebenfalls eine PUSH-Notification angezeigt sobald die Aufgabe terminiert.

3.2 Lösung 1: Serviceworker

3.3 Lösung 2: nativer Android Dienst

3.4 Web Applikation

3.4.1 funktionale Anforderungen

[WFA-1] Benutzerauthentifizierung. Benutzer können sich für die Nutzung der Anwendung Registrieren und anschließend Anmelden. Für die Registrierung ist ein eindeutiger Benutzername mit Angabe einer E-Mail Adresse sowie ein Passwort notwendig.

[WFA-2] Kontaktliste. Benutzer können sich untereinander mittels Benutzername bzw. E-Mail Adresse zur persönlichen Kontaktliste hinzufügen.

[WFA-3] Gruppen verwalten. Benutzer können Gruppen anlegen und andere Benutzer hinzufügen. Ein Gruppenadministrator kann die Gruppe bearbeiten oder löschen. Benutzer können aus einer Gruppe austreten.

[WFA-4] Aufgaben anlegen, bearbeiten und löschen. Ein Benutzer soll Aufgaben anlegen und anschließend Bearbeiten oder Löschen können. Eine Aufgabe muss einen Titel besitzen. Optional können eine Beschreibung, ein Ort, Zeitraum sowie Fälligkeitsdatum hinterlegt werden.

[WFA-5] Aufgaben teilen. Aufgaben können mit mehreren Benutzer geteilt werden. Ebenfalls können Aufgaben einer Gruppe zugeordnet werden.

[WFA-6] Benachrichtigungen. Benutzer, die in einer Aufgabe involviert sind, erhalten Benachrichtigungen über Änderungen an Aufgaben. Wenn für eine Aufgabe eine Fälligkeit mit Benachrichtigung hinterlegt wurde, wird der Benutzer zum entsprechenden Zeitpunkt

informiert.
Wird ein Benutzer in eine Gruppe eingeladen bzw. wird einer Gruppe eine Aufgaben hinzugefügt bzw. bearbeitet werden alle Gruppenmitglieder entsprechend Benachrichtigt.

3.4.2 nicht-funktionale Anforderungen

[WNFA-1] Singlepage Applikation.

[WNFA-2] Offlinefähigkeit.

[WNFA-3] Look and Feel einer nativen Android App.

KRITERIUM	BESCHREIBUNG
1. Allgemeine technische Anforderungen	**Der native Dienst...**
1.1 Plattform: Android	...soll für die Android Plattform entwickelt werden. Als Mindestversion soll Android 5.0 (API Level 21) vorausgesetzt werden. Eine Unterstützung anderer Betriebssysteme ist nicht vorgesehen.
1.2 Hintergrunddienst	...soll als Hintergrunddienst ohne GUI implementiert werden.
1.3 automatischer Start	...soll automatisch nach Systemstart gestartet werden. Wird der Dienst aus irgendeinem Grund beendet, soll er automatisch neustarten.

Tabelle 1: Anforderungen an nativen Dienst

3.5 Serverkomponente

3.5.1 funktionale Anforderungen

[SFA-1] RESTful-Schnittstelle. Der API Server unterstützt folgende Anforderungen um die Funktionalitäten einer RESTful-Schnittstelle zu erfüllen:

- Bereitstellung von CRUD[1]-Funktionalität für Entities
- Aufruf von Ressourcen über eindeutige und einfache URLs (z.B. https://example.de/api/task/ und https://example.de/api/task/:taskId)
- Verwendung der standardisierten HTTP-Methoden (GET, POST, PUT und DELETE)
- Rückgabe im JSON-Format
- alle Requests werden auf der Konsole ausgegeben

[SFA-2] Gesicherter Zugriff auf API. Der Zugriff auf die API ist nur für authentifizierte Benutzer möglich. Für die Authentifizierung wird das Konzept Token verwendet.
KONZEPT TOKEN BESCHREIBEN

3.5.2 nicht-funktionale Anforderungen

ssdf

3.6 Lösung 2: nativer Android Dienst

Tabelle ?? stellt eine Auflistung von Anforderungen an die Lösung mittels nativem Android Dienst dar.

[1] *CRUD: create, read, update, delete*

4 Konzeption

4.1 Architekturbeschreibung - Serviceworker

Die Anwendung beruht auf dem Client-Server-Prinzip. Dabei stellt der Client lediglich die Oberfläche zur Interaktion mit dem Anwender dar. Außer der notwendigen UI Logik ist die gesamte Geschäftslogik auf einen dedizierten (Business-)Server ausgelagert. Dieser stellt ebenfalls die Datenbank bereit. Mittels AJAX und REST werden dynamische Daten vom Client beim Server angefragt.

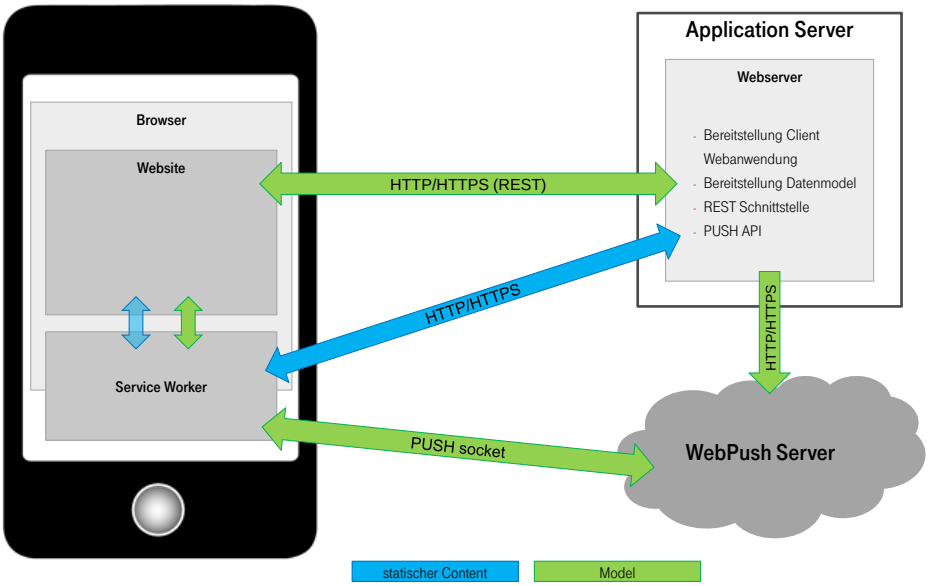

Abbildung 1: Archtikturbeschreibung - Umsetzung mit Serviceworker

4.1.1 Push API

... Beschreibung (mit Schema) der Softwarearchitektur ...

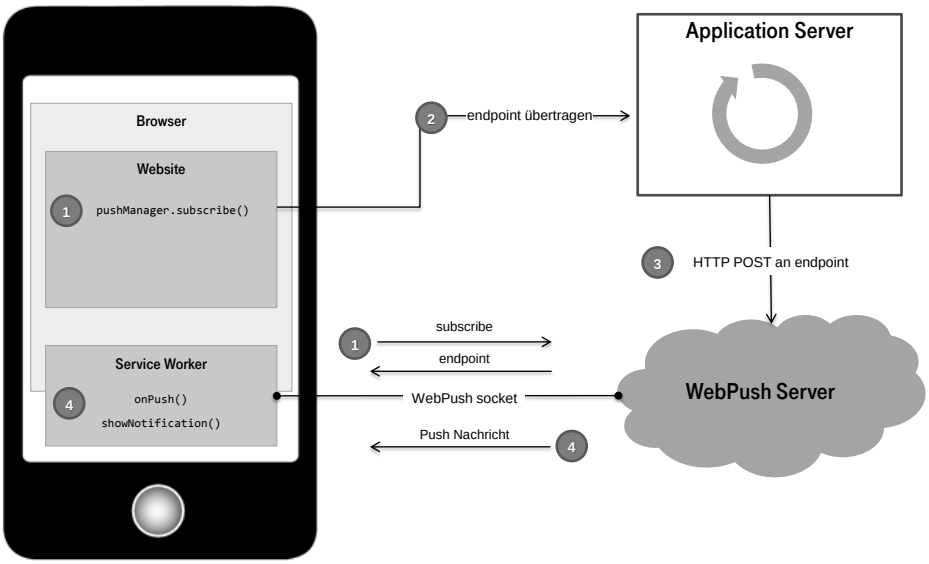

Abbildung 2: Push mittels Serviceworker

4.2 Architekturbeschreibung - nativer Android Dienst

... Beschreibung (mit Schema) der Softwarearchitektur ...

4.3 Push

4.4 Datenmodell

4.5 Serverkomponente

4.5.1 REST Schnittstelle

Route	HTTP-Methode	Beschreibung
/api/signup	POST	Registriert einen neuen Benutzer
/api/authenticate	POST	Authentifiziert einen Benutzer
/api/tasks	GET	Gibt alle Aufgaben zurück
/api/tasks	POST	Legt eine neue Aufgabe an
/api/tasks/:taskId	GET	Gibt eine einzelne Aufgabe zurück
/api/tasks/:taskId	PUT	Aktualisiert eine einzelne Aufgabe
/api/tasks/:taskId	DELETE	Löscht eine einzelne Aufgabe

Tabelle 2: Übersicht API Routen

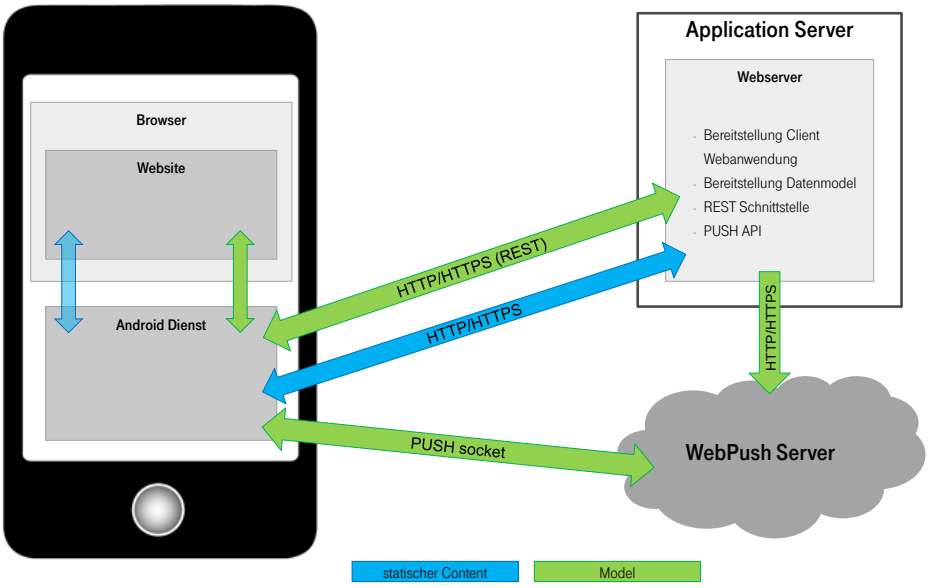

Abbildung 3: Architektur - Umsetzung mit nativem Android Dienst

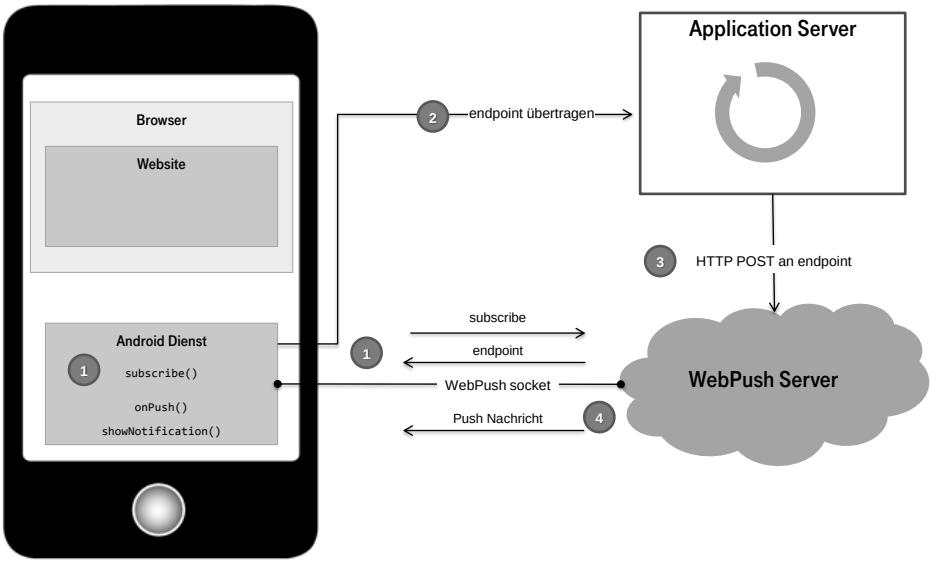

Abbildung 4: Push mittels nativem Android Dienst, in Anlehnung an (1,)

4.6 Benutzeroberfläche

... Mockup ...
... Beschreibung des UI ...

Um das „Look and Feel“ einer nativen App zu erreichen wird das UI-Framework **nativeDroid2** verwendet.

4.7 Serviceworker

4.8 nativer Android Dienst

5 Implementierung

... Wie wurden die beiden Ideen umgesetzt ...

5.1 Serverkomponente

5.2 Umsetzung ServiceWorker

... Umsetzung mittels ServiceWorker ...

5.3 Umsetzung mittels nativem Android Service

... Umsetzung mittels eigenem nativen Service ...

6 Zusammenfassung und Ausblick

... Was kann nicht geleistet werden? ...

... Was ist eventuell zukünftig möglich ? ...

Literatur

Mozilla: *Firefox/Push Notifications - MozillaWiki*. `https://wiki.mozilla.org/Firefox/Push_Notifications`. Version: 2016

7 API Beschreibung

Löscht eine einzelne Aufgabe

Tabelle 3: Übersicht API Routen

/api/signup

Request:	**Response:**
• `username`: (String) gewünschter Benutzername • `password`: (String) Passwort • `email`: (String) E-Mail Adresse	• `username`: (String) gewünschter Benutzername • `password`: (String) Passwort • `email`: (String) E-Mail Adresse

Benutzer anlegen	
URL	`/api/signup`
Methode	`GET`
Request-Parameter	Required: • `username`: (String) gewünschter Benutzername • `password`: (String) Passwort • `email`: (String) E-Mail Adresse
Success-Response	• Code: 200 • Content: `success: true, message: 'Successful created new user.'`
Legt einen neuen Benutzer an.	
Request `username`: (String) gewünschter Benutzername `password`: (String) Passwort `email`: (String) E-Mail Adresse	

- `username`: (String)
  gewünschter Benutzername
- `password`: (String)
  Passwort
- `email`: (String)
  E-Mail Adresse

Response

- Bereitstellung von CRUD[2]-Funktionalität für Entities
- Aufruf von Ressourcen über eindeutige und einfache URLs (z.B. https://example.de/api/task/ und https://example.de/api/task/:taskId)

[2] *CRUD: create, read, update, delete*