Hochschule für Telekommunikation Leipzig (HfTL)

Profilierung Netzbasierte Anwendungen

Projektdokumentation

TaskY - Cache und Notifications in mobilen Webanwendungen

David Howon (147102)

Michael Müller (147105)

Wintersemester 2016/17

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Ziele

Diese Dokumentation entstand im Rahmen der Profilierung „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.

2.1 Serviceworker

2.2 Push API vs. Notification API

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 funktionale Anforderungen

Im Rahmen dieser Dokumentation werden unter funktionalen Anforderungen diejenigen verstanden, welche zur direkten Zielerfüllung beitragen (vgl. 1.1).

...weiter ausführen...

[FA-1] Single Page Application

[FA-2] Offlinefähigkeit Die Benutzung der Webanwendung soll nicht ausschließlich bei bestehender Internetverbindung, sondern ebenfalls Offline reibungslos möglich sein. Dazu bietet die hybride Webanwendung Mechanismen zum Vorhalten der persistenten Daten und des nutzerspezifischen Datenmodells im Offlinezustand. Benutzer werden über ggf. eingeschränkte Funktionalitäten informiert, während keine aktive Internetverbindung vorhanden ist.

[FA-3] Push-Benachrichtigungen Benutzer der Webanwendung werden unabhängig vom verwendeten Endgerät über bestimmte Ereignisse mit Hilfe von Push-Benachrichtigungen informiert. Diese Ereignisse werden vom Applicationserver verarbeitet und dieser initiiert Push-Benachrichtigungen beim Client.

[FA-4] Schnittstelle für Kommunikation mit Applicationserver 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

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

3.3 nicht-funktionale Anforderungen

Im Rahmen dieser Dokumentation werden unter nicht-funktionalen Anforderungen diejenigen verstanden, welche nicht zur direkten Zielerfüllung beitragen (vgl. 1.1).

...weiter ausführen...

[NFA-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.

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

[NFA-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.

[NFA-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.

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

[NFA-8] Gesicherter Zugriff auf API. Der Zugriff auf die API ist nur für authentifizierte Benutzer möglich. Für die Authentifizierung wird das Konzept Token verwendet.

[NFA-9] Ereignisse für 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.

- Freundschaftsanfrage wurde von einem anderen Benutzer gestellt
- Freundschaftsanfrage wurde durch einen anderen Benutzer bestätigt/abgelehnt
- ein anderer Benutzer hat die eigene Freundschaftsanfrage bestätigt/abgelehnt

4 Konzeption

4.1 Caching der statische Resourcen

4.2 Web Push

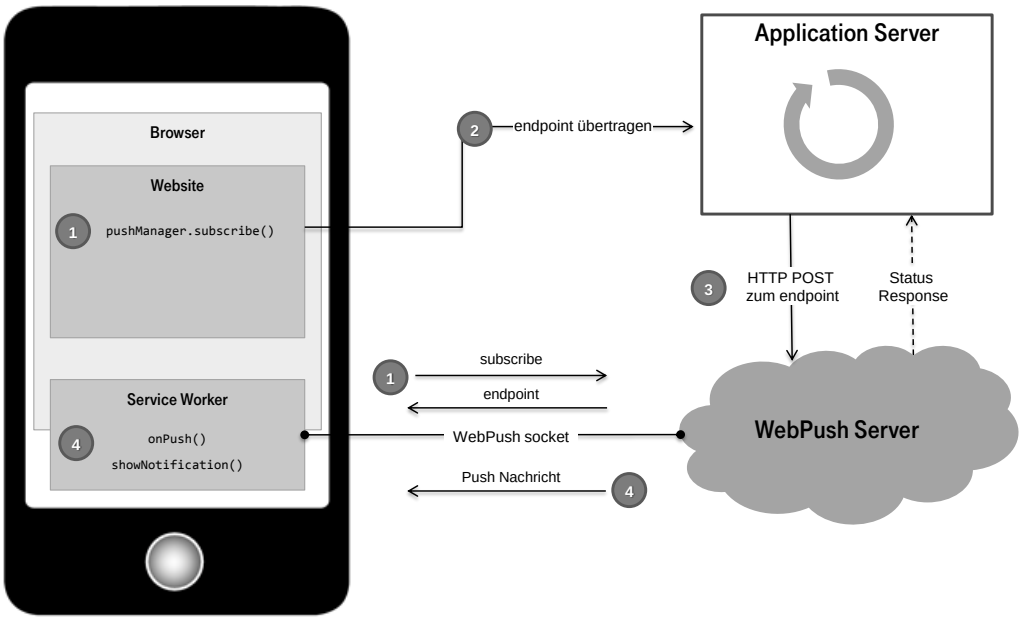

Abbildung 4.1: Push mittels Serviceworker (in Anlehnung an MozillaWiki [1])
Quelle: `https://wiki.mozilla.org/File:PushNotificationsHighLevel.png`

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

4.3 Architekturbeschreibung

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- und Serviceworker-Logik ist die gesamte Geschäftslogik auf einen Business-Server (Applicationserver) ausgelagert. Die zentrale Datenbank sowie die statischen Ressourcen zur

Darstellung des Client werden ebenfalls vom Applicationserver bereitgestellt. Für die Kommunikation steht eine RESTful-Schnittstelle zur Verfügung.

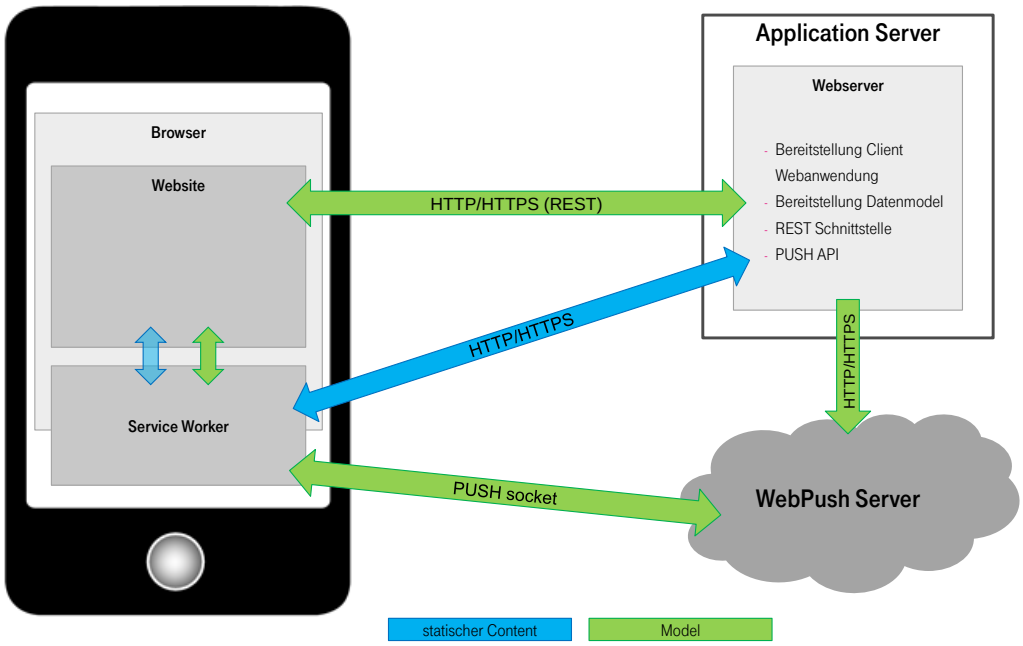

Abbildung 4.2: Archtikturbeschreibung - Umsetzung mit Serviceworker

4.4 Applicationserver

4.4.1 Datenbank

4.4.2 REST-API

Zur Bereitstellung von CRUD-Funktionalitäten über standardisierte HTTP-Methoden (vgl. **??** Anforderungen an Serverkomponente) wird dem Applicationserver eine RESTful-Schnittstelle hinzugefügt. Eine Übersicht über mögliche API-Routen mit entsprechender HTTP-Methode ist in Tabelle 4.1 dargestellt.

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
/push/devices	GET	Gibt alle registrierten Geräte zurück
/push/devices	POST	Registriert ein neues Gerät

Tabelle 4.1: Übersicht API Routen

Authentifizierung und Autorisierung

Für den Zugriff auf die CRUD-Methoden ist eine Benutzerauthentifizierung und Autorisierung notwendig. Dazu wird das Token-Verfahren verwendet.

4.5 Client-Oberfläche

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

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

4.6 Datenmodel

4.6.1 Datenbankschema

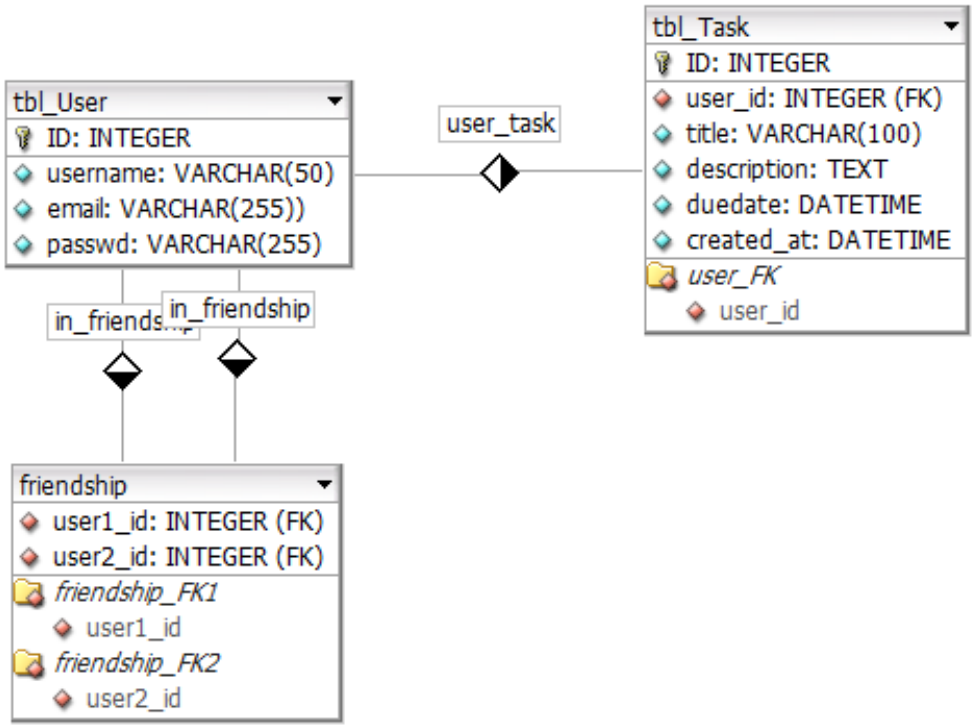

Abbildung 4.3: Datenmodell

4.6.2 Bereitstellung der Daten

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 ...

```
// Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
    public void paintComponent(Graphics g) {
        g.drawString("Hello, world!", 65, 95);
    }
}
```

6 Zusammenfassung und Ausblick

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

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

Literaturverzeichnis

[1] Mozilla: *Firefox/Push Notifications - MozillaWiki*. `https://wiki.mozilla.org/Firefox/Push_Notifications`. Version: 08.01.2017

Abbildungsverzeichnis

7 Anhang

7.1 API Beschreibung

Löscht eine einzelne Aufgabe

Tabelle 7.1: 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[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)

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