Systemarchitektur

FindLunch

Version	1.3
Datum	12.08.2016
Status	**Final**
Autoren	AJ
Verteiler	CCWI, JH, MEK, AK, AJ, CP, AS

Historie

Version	Datum	Autoren	Status	Kommentar/Änderung
1.1	23.06.2016	AJ	Final	Anpassung am Datenmodell zum Change Request Task 220 (Anzeige von Thumbnails)
1.2	05.08.2016	AJ	In Bearbeitung	Verteilung der App, Transaktionskonzept, Performance-Optimierung der Datenbank, Backup-Strategie, Push Dienst, Mehrsprachigkeit, Performancetests, Serverkonfiguration
1.2	05.08.2016	AS	Review	
1.2	08.08.2016	AK	Review	
1.2	09.08.2016	AJ	Final	Korrekturen
1.3	12.08.2016	AJ	Final	Passwörter und IP-Adressen für Veröffentlichung zensiert

Dokumentenbezug: Technische Architektur, Technische Dokumentation

Inhaltsverzeichnis

1 Einleitung

Die Architektur eines Softwaresystems beschreibt die einzelnen Komponenten des Systems, ihre Schnittstellen, sowie die zu ihrer Erstellung verwendeten Tools und Techniken. Ein klar definiertes Architekturdesign ist bei komplexer Software mit mehreren Entwicklern wichtig, um eine einheitliche und saubere Entwicklung zu gewährleisten.

In diesem Dokument wird die Architektur des FindLunch-Systems beschrieben und die wichtigsten Entscheidungen dokumentiert.

2 Übersicht

Abbildung 1 zeigt die Gesamtarchitektur des FindLunch-Systems. Die Daten werden vom Anbieter über einen Browser an den Sever gesendet und in einer Datenbank abgelegt. Von dort werden sie über einen Webservice (manueller Abruf) oder einen Push-Dienst (automatische Benachrichtigung) an eine Smartphone App übertragen und dem Kunden angezeigt.

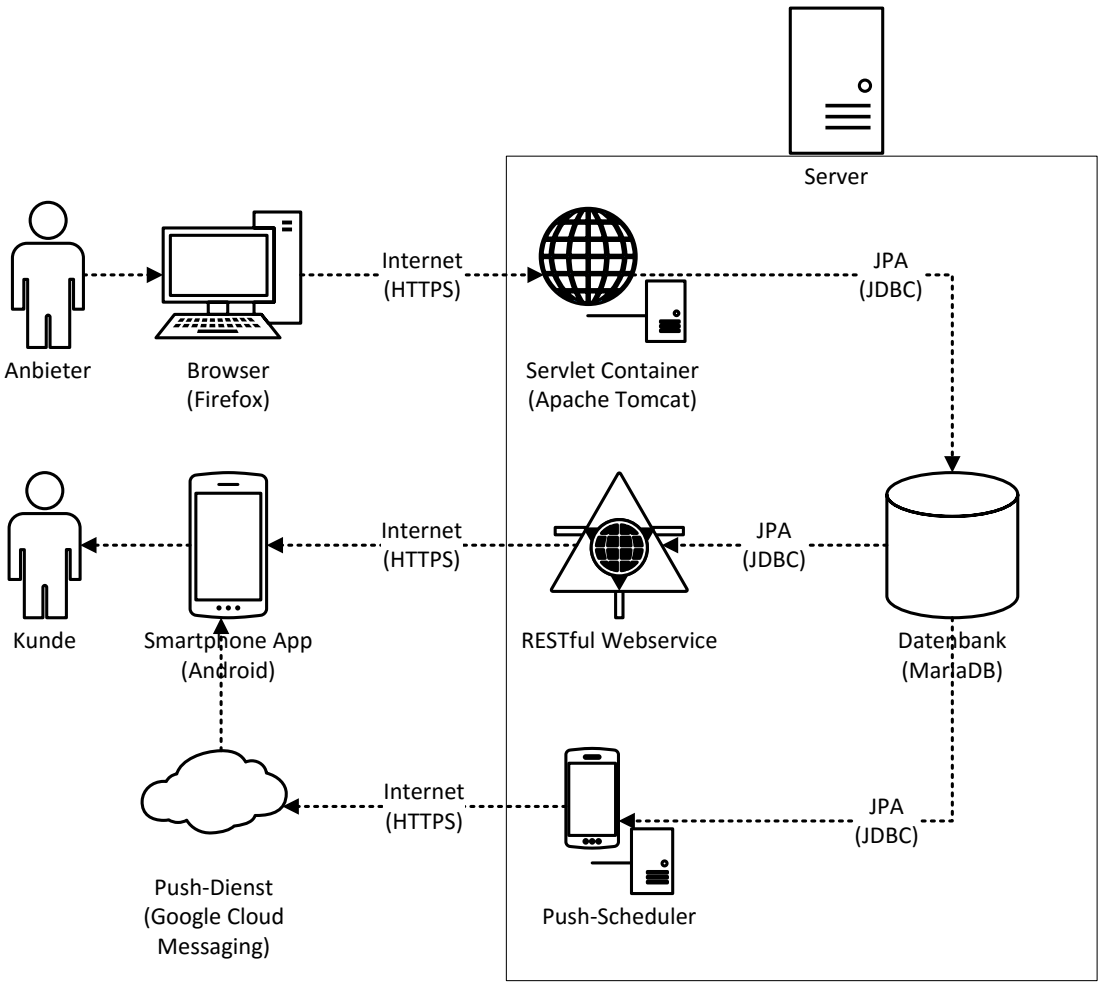

Abbildung 1: Architektur des FindLunch-Systems

3 Entwicklung

Für die Entwicklung des Web-Interfaces (für den Anbieter) und des Webservices (für den Kunden) wird das *Spring Framework* mit *Java 8* genutzt. Spring vereinfacht die Entwicklung der Java-Module durch *Dependency Injection* (Abhängigkeiten werden zentral vom Framework statt von den einzelnen Objekten verwaltet) und *Aspect Oriented Programming* (einzelne Aspekte werden zentral definiert und von den Objekten genutzt). Es folgt somit dem Designprinzip *Inversion of Control*[1].

Die Schnittstellen zu den Benutzern werden mit dem *Web MVC Framework* von Spring umgesetzt. Dies gilt sowohl für den Webservice, auf dessen Funktionen die Smartphone App zugreift, als auch für die Webseite, die der Anbieter nutzt. MVC steht dabei für das *Model View Controller (MVC)* Entwurfsmuster, das bei diesem Framework umgesetzt wird. Dabei werden die datentragenden Objekte (Model), die Oberfläche (View) und die Klassen zur Verarbeitung der Benutzereingaben (Controller) separat erstellt und kommunizieren über Schnittstellen miteinander[2].

3.1 Verwendete Spring-Module

Eine zentrale Rolle bei der Entwicklung spielt *Spring Boot*, das die Konfiguration des Spring Frameworks vereinfacht und das Erstellen des Programms in einer ausführbaren Version mit allen Abhängigkeiten (executable JARs) ermöglicht.

Der Zugriff auf die Datenbank erfolgt über Java Database Connectivity (JDBC) mittels der Java Persistence API (JPA). Diese vereinfacht die Programmierung, indem sie die Tabellen der Datenbank als Objekte in Java bereitstellt. Einzelne Zeilen in der Datenbank entsprechen Instanzen dieser Objekte in Java. Die JPA-Schnittstelle wird mit dem *Hibernate* Framework umgesetzt. Zur Vereinfachung dieser Umsetzung wird mit *Spring Data JPA* ein Repository-Interface erzeugt und für den Datenzugriff verwendet.

Für die Absicherung der Anwendung (Authentifizierung von Anbietern und Kunden) und der Kommunikation zwischen Server und Clients wird *Spring Security* verwendet.

Weitere Details zur Konfiguration sind in Anhang II: Erweiterte Konfiguration des Find-Lunch-Servers beschrieben.

---

[1] Martin Fowler, „Inversion of Control Containers and the Dependency Injection pattern“, *martinfowler.com*, 2004, http://martinfowler.com/articles/injection.html.

[2] Praveen Gupta und Prof. M.C. Govil, „Spring Web MVC Framework for rapid open source J2EE application development: a case study“, *International Journal of Engineering Science and Technology* 2, Nr. 6 (2010): 1684–89.

3.2 Weitere Software

Für die Entwicklung der Webanwendung werden die Entwicklungsumgebung Eclipse und das Build-Tool Apache Maven verwendet. Innerhalb von Eclipse werden außerdem die in Tabelle 1 aufgelisteten Plug-Ins genutzt.

Tabelle 1: Übersicht der verwendeten Eclipse-Plug-Ins

Plugin	Funktion
ObjectAid	Erzeugen von UML-Diagrammen
Metrics	Berechnen von Metriken und Anzeigen der Lines of Code
EclEmma	Berechnen der Code Coverage durch Integrationstests
JAutodoc	Erzeugen von JavaDoc

Über die bereits erwähnten Spring-Module wird die Funktionalität der Webanwendung bereitgestellt. Für deren Design werden die Template-Engine *Thymeleaf* und das CSS-Framework *Bootstrap* verwendet. Thymeleaf dient dem Erzeugen und Verwalten der Webseiten-Templates und Boostrap stellt Design-Elemente für diese zur Verfügung. Die Auslieferung der Webseite an den Browser erfolgt durch den Servlet Container *Apache Tomcat*. Dieser wird nicht auf dem Server installiert, sondern in einem executable JAR deployed (embedded Tomcat).

Als Entwicklungsumgebung für die Android-App wird Android Studio mit Gradle als Build-Tool verwendet. Die App greift dabei auf Webservices zu, die ebenfalls mit Spring entwickelt werden.

3.3 Versionsverwaltung

Zur Versionsverwaltung wird projektweit Git verwendet, dass über das Git-TF Plug-In in den Team Foundation Server (TFS) eingebunden ist. Für die Smartphone App und die Webanwendung werden zwei verschiedene Repositories angelegt.

Aufgrund von Problemen und Unzulänglichkeiten des Mergings mit Git (z.B. doppelte Implementierung von Funktionen, da bereits erstellter Code nicht sofort für alle verfügbar ist), wurde seitens der Entwickler entschieden die einzelnen Aufgaben sauber zu trennen, Konflikte durch zeitnahe Kommunikation zu vermeiden und direkt in den Master-Branch zu comitten.

Ursprünglich war vorgesehen für jedes Feature einen separaten Feature-Branch zu erstellen, in dem es entwickelt wird. Nach Fertigstellung des Features sollte über einen "Pull-Request" (TFS-Funktionalität) ein Review durch einen anderen Entwickler erfolgen und anschließend das Feature zurück in den Master-Branch überführt werden.

Die finale Version des Codes wird unter der Apache 2.0 Lizenz in dem Repository https://github.com/andju/findlunch veröffentlicht.

3.4 Deployment

Stabile Versionen der Software werden vom TFS heruntergeladen. Der Server wird dabei als executable JAR (siehe Kapitel 3.1) und die App als *Android Package* (APK-Datei) erzeugt. Die JAR-Datei wird auf dem Server abgelegt und dort gestartet. Das APK wird auf das Android-Smartphone kopiert und dort installiert. Hierzu muss die Installation aus unbekannten Quellen auf dem Gerät erlaubt sein. Im Produktivbetrieb würde das APK über den Google Play Store verteilt werden (siehe Kapitel 5.2.1).

4 Server

Für den (prototypischen) Betrieb des FindLunch-Systems wird ein Server benötigt, der die in Tabelle 2 spezifizierten Anforderungen erfüllt. Dieser Server dient der Verwaltung der Datenbank (Datenbankserver), der Bereitstellung des Webportals für die Anbieter (Servlet Container), des Webservice (zur Kommunikation mit der App) und des Push-Schedulers.

Tabelle 2: Anforderungen an den Server

Komponente	Anforderung
CPU	2 GHz, Dual-Core
RAM	4 GB
HD	100 GB
OS	Microsoft Windows Server 2012 R2

4.1 Setup

Für die Implementierung und den Betrieb des FindLunch-Systems wird einen Server der Hochschule München genutzt, der (ausschließlich innerhalb des Hochschulnetzwerks) über die IP-Adresse X.X.X.60 erreichbar ist. Auf dem Server wurden die in Tabelle 3 beschriebenen Benutzer eingerichtet.

Außerdem wurde die in Tabelle 4 und Tabelle 5 beschriebene Software installiert und eingerichtet. Das in Tabelle 6 beschriebene Packprogramm wird zur Umsetzung der in Kapitel 4.6 beschriebenen Backup-Strategie benötigt.

Tabelle 3: Auf dem Sever eingerichtete Systembenutzer

Name	Gruppe	Kennwort	Beschreibung
A	Administratoren		Entwickler
B	Administratoren		Architekt
C	Administratoren		Entwickler
D	Administratoren		Entwickler
sys_findlunch_serv	Benutzer	XXXXXXXXXX	Account, mit eingeschränkten Rechten, zum Ausführen des Tomcat-Servers
sys_findlunch_backup	Administratoren	XXXXXXXXXX	Account zum Ausführen der Backup-Jobs.

Tabelle 4: Auf dem Server installierte Java-Umgebung

Name	Version	Architektur
Java Runtime Environment 8	8u91	x64
http://download.oracle.com/otn-pub/java/jdk/8u91-b14/jre-8u91-windows-x64.exe		

Tabelle 5: Auf dem Server installierte Datenbank

Name	Version	Architektur
MariaDB	10.1.13 Stable	x64
https://downloads.mariadb.org/interstitial/mariadb-10.1.13/winx64-packages/mariadb-10.1.13-winx64.msi		

- Default server's character set: UTF8
- Service Name und Port: MySQL: 3306
- Root-passwort: $ThD8rkVp6
- Datenbank-Name: findlunch
- Service-User
  - Name: db_findlunch_service
  - Passwort: XXXXXXXXXX
  - Berechtigungen (Datenbank: findlunch): SELECT, INSERT, UPDATE, DELETE
  - CREATE-Statement:
    ```
    CREATE USER db_findlunch_service@'127.0.0.1'
    IDENTIFIED BY ' XXXXXXXXXX'
    GRANT SELECT, UPDATE, INSERT, DELETE ON `findlunch`.* TO 'db_findlunch_service'@'127.0.0.1';
    ```
- Admin-User
  - Name: db_findlunch_admin
  - Passwort: XXXXXXXXXX
  - Berechtigungen (Datenbank: findlunch): SELECT, INSERT, UPDATE, DELETE
  - CREATE-Statement:
    ```
    CREATE USER db_findlunch_admin@'%' IDENTIFIED
    BY 'XXXXXXXXXX'
    GRANT SELECT, UPDATE, INSERT, DELETE  ON `findlunch`.* TO 'db_findlunch_admin'@'%';
    ```
- Backup-User
  - Name: db_findlunch_backup
  - Passwort: XXXXXXXXXX
  - Berechtigungen (Datenbank: findlunch): SELECT, SHOW VIEW, LOCK TABLES
  - CREATE-Statement:
    ```
    CREATE USER db_findlunch_backup@'%' IDENTIFIED
    BY 'XXXXXXXXXX'
    GRANT SELECT, SHOW VIEW, LOCK TABLES ON `findlunch`.* TO ' db_findlunch_backup'@'%';
    ```

Tabelle 6: Auf dem Server installiertes Packprogramm

Name	Version	Architektur
7-Zip	16.02	x64
http://www.7-zip.org/a/7z1602-x64.exe		

4.2 Datenbank

Als Datenbankserver wird die freie Software MariaDB 10.1 eingesetzt. Die Entscheidung gegen MySQL als Datenbankserver wurde getroffen, da die Zukunft von MySQL als quelloffene Software ungewiss ist[3]. Hinsichtlich der für das FindLunch-System benötigten Funktionalität, gibt es keine signifikanten Unterschiede zwischen den beiden Produkten.

4.2.1 Datenmodell

Abbildung 2 zeigt das Datenmodell der Datenbank des FindLunch-Systems (erzeugt mit MySQL Workbench 6.3 CE). Das daraus generierte Skript zur Erzeugung der FindLunch-Datenbank befindet sich in Anhang III: Erzeugung der FindLunch-Datenbank.

[3] Mark Callaghan, „(less) open source“, 17. August 2012, http://mysqlha.blogspot.de/2012/08/less-open-source.html.

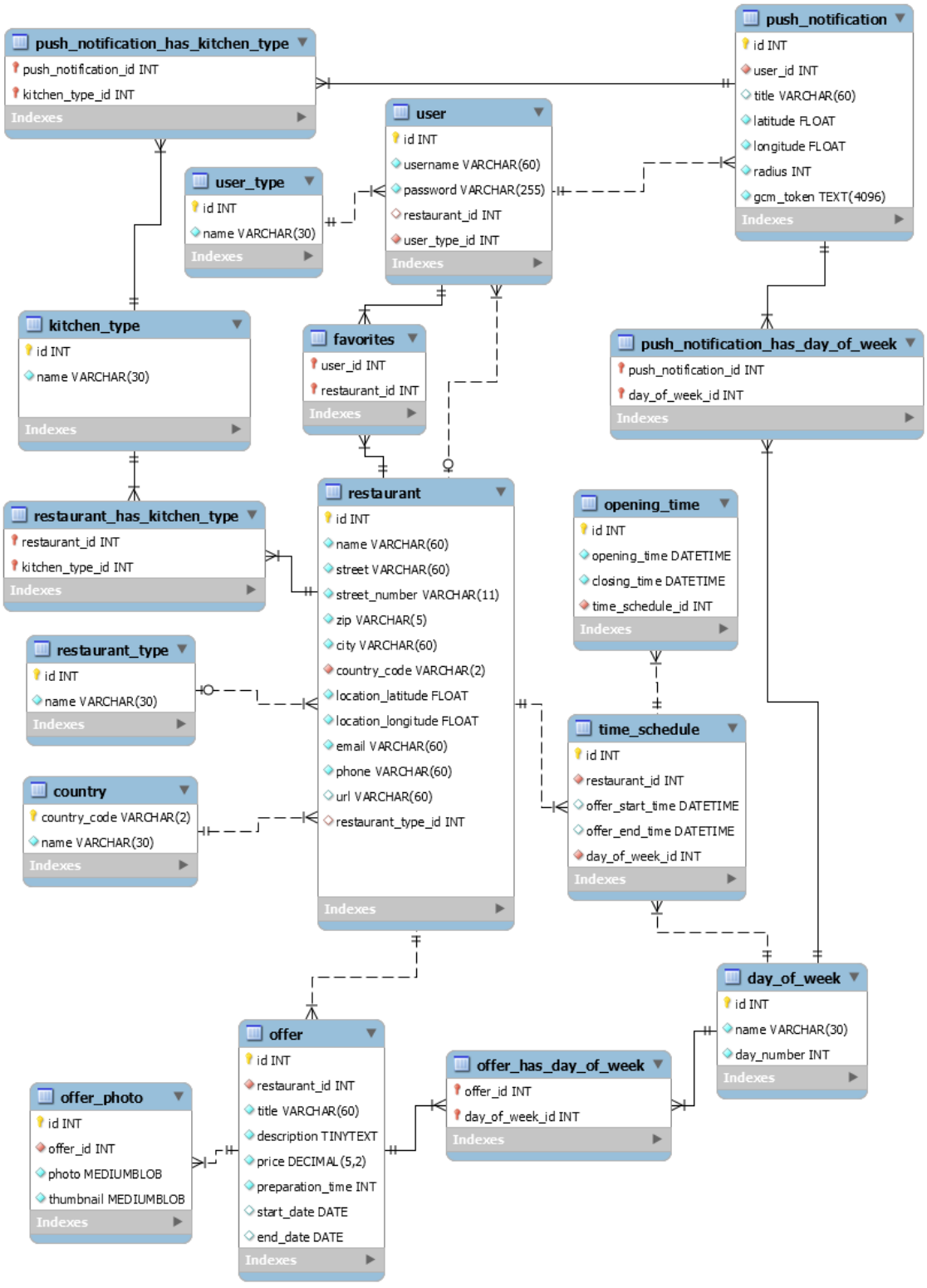

Abbildung 2: Datenmodell der FindLunch-Datenbank

4.2.1.1 Kunden und Anbieter

In der Tabelle *user* befinden sich die Zugangsdaten der Benutzer. Die beiden Nutzertypen (Kunde und Anbieter) werden anhand ihrer user_type_id unterschieden, die auf den entsprechenden Typ in der Tabelle *user_type* verweist.

Die Tabelle *restaurant* enthält die Daten der Restaurants (von Anbietern). Neben dem Namen, der Adresse (die Länder der Adresse werden in der Tabelle *country* gepflegt und über den ISO 3166 Country Code verbunden) und den Kontaktinformationen ist dies die Angabe zur Lage des Restaurants in Längen- (location_longitude) und Breitenangabe (location_longitude). Die Informationen zur Lage werden für die Umkreissuche in der App verwendet. Jedem Restaurant wird über einen Fremdschlüssel (restaurant_type_id) eine Kategorie zugeordnet, die in der Tabelle *restaurant_type* gepflegt ist.

Über die Tabelle *favorites* wird die Zuordnung der Restaurants als Favoriten der Kunden, und über die Tabelle *restaurant_has_kitchen_type* die Zuordnung des Restaurants zur Art der angebotenen Küche (deutsch, asiatisch, etc.) realisiert. Die angebotenen Küchen befinden sich in der Tabelle *kitchen_type*.

4.2.1.2 Angebote

Die Tabelle *offer* enthält die Angebote der Restaurants und ihre Details. Jedes Angebot ist über das Feld restaurant_id einem Restaurant eindeutig zuordenbar und hat ein- oder mehrere Fotos (mit Thumbnails), die in der Tabelle *offer_photo* als Binary Large Object (MEDIUMBLOB, bis zu 16 MB) abgespeichert werden.

Der Zeitraum (in Tagen), in dem ein gegebenes Angebot gültig ist, wird durch start_date und end_date definiert. Da ein Angebot nicht nur nach Zeiträumen, sondern auch auf bestimmte Wochentage (in dem jeweiligen Zeitraum) eingeschränkt sein kann, wird mit der Tabelle *offer_has_day_of_week* eine Beziehung zwischen dem definierten Zeitraum und den Wochentagen in der Tabelle *day_of_week* hergestellt. Diese enthält die Namen der Wochentage und ihre Reihenfolge.

4.2.1.3 Angebots- und Öffnungszeiten

In der Tabelle *time_schedule* werden für jeden Wochentag und jedes Restaurant die Uhrzeiten hinterlegt, zwischen denen die eingetragenen Angebote gültig sind. Die ID dieser Tabelle wird genutzt, um die Öffnungszeiten (die in der Tabelle *opening_time* gepflegt werden) zu verwalten. Dieses Konstrukt ermöglicht es, für jedes Restaurant pro Tag unterschiedliche Öffnungszeiten zu definieren. So können z. B. Mittagspausen (durch zwei Öffnungszeiten an einem Tag) abgebildet werden.

4.2.1.4 Push-Benachrichtigungen

Die Push-Benachrichtigungen eines Kunden für neue Angebote werden in der Tabelle *push_notification* verwaltet. Darin werden der Titel der Benachrichtigung und die örtlichen Bedingungen (latitude, longitude und radius) verwaltet, die zum Auslösen einer Push-Benachrichtigung erfüllt sein müssen. Optional können Küchenarten definiert werden, von denen das Restaurant (welches das Angebot herausgibt) mindestens einer zugeordnet sein muss. Diese werden in der Tabelle *push_notification_has_kitchen_type* gepflegt.

Im Feld gcm_token wird der Google Cloud Messaging Registration Token, der für die Adressierung des Smartphones benötigt wird, gespeichert. Da ein Kunde auf unterschiedlichen Geräten unterschiedliche Push-Benachrichtigungen anlegen kann, wird dieser Token nicht zentral in der Tabelle user gespeichert.

Über die Tabelle *push_notification_has_day_of_week* wird angegeben, an welchen Wochentagen die Push-Benachrichtigung ausgeführt werden soll.

4.2.1.5 Einschränkungen der Feldwerte

Neben den Einschränkungen durch die verwendeten Datentypen zeigt Tabelle 7 weitere Einschränkungen der gültigen Werte. Die Webseiten-URLs und die E-Mail-Adressen werden dabei künstlich beschränkt: Laut den Standards RFC 5322 und RFC 7230 existiert keine Limitation. Allerdings kann davon ausgegangen werden, dass in der Praxis URLs bzw. E-Mail-Adressen mit mehr als 60 Zeichen kaum auftreten.

Tabelle 7: Gültige Wertebereiche der Tabellenfelder

Tabelle	Feld	Datentyp	Gültiger Wertebereich
*	id	INT	Auto increment
day_of_week	day_number	INT	Unique, RegEx: `[1-7]`
offer	price	DECIMAL(5,2)	RegEx: `[0-9]{1,3}(.[0-9]{1,2})?`
offer	preparation_time	INT	RegEx: `[0-9]{3}`
restaurant	street_number	VARCHAR(11)	RegEx: `[\d]{1,4}[a-z]?(-[\d]{1,4}[a-z]?)?`
restaurant	zip	VARCHAR(5)	Für den Prototyp nur deutsche Postleitzahlen: 01000 – 99999.
country	country_code	VARCHAR(2)	Für den Prototyp nur „DE“. Generell alle ISO 3166-1 alpha-2 Codes.
restaurant	phone	VARCHAR(60)	RegEx: `[+0][0-9]{1,59}`
restaurant	url	VARCHAR(60)	RegEx: `(http)[s]?(://)[\w./~]{1,52}`
user	username	VARCHAR(60)	RegEx: `[a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}`
restaurant	email		

4.2.2 Performance-Optimierung

Die von der Anwendung verwendeten Bilder werden als Binärdaten in der Datenbank abgelegt (in den Spalten photo und thumbnail der Tabelle offer_photo). Dadurch ist die Anwendung plattformunabhängig und kann z.B. auch auf einem Linux-Server genutzt werden. Außerdem vereinfacht dies die Backup-Strategie, da die Bilder zusammen mit der Datenbank gesichert werden.

Ein Nachteil ist allerdings die geringere Performance beim Zugriff auf Binärdateien in der Datenbank[4]. Eine Möglichkeit zur Optimierung der Performance ist somit, die Bilder als Dateien im Dateisystem abzulegen und die Verweise darauf als Text (varchar) in der Datenbank zu speichern.

Eine weitere Möglichkeit zum Optimieren der Performance ist das Setzen von Indizes auf Spalten mit häufigen Lese- und seltenen Schreibzugriffen. Dies ist eine einfach umzusetzende Maßnahme, die keine Änderung am Datenmodell, der Webanwendung oder der App erfordert. Für folgenden Spalten in der FindLunch-Datenbank wird das Setzen von Indizes empfohlen:

1. Tabelle restaurant: location_latitude und location_longitude
    a. Diese Felder werden für die Umkreissuche in der App benötigt.
2. Tabelle user: username
    a. Dieses Feld wir für viele REST-Aufrufe, die benutzerspezifische Dienste bereitstellen, benötigt.

4.3 Servlet Container

Der Servlet Container nimmt die Anfragen des Browsers auf dem Client entgegen und leitet sie an Servlets (Java-Klassen) weiter. Deren Antwort gibt er in Form von Java Server Pages (JSPs) an den Client zurück. Abbildung 3 zeigt beispielhaft, an der Registrierung eines neuen Benutzers, die internen Abläufe beim Aufruf einer Funktion.

Intern werden verschiedene Pfade auf dem Webserver verlinkt, die unterschiedliche Zwecke erfüllen. Tabelle 8 zeigt eine Liste aller Webserver-Pfade und deren Beschreibungen. Abbildung 4 zeigt den Ablauf und die Navigation zwischen den wichtigsten Pfaden.

[4] Ein einfacher Test ist hier beschrieben: http://blog.lick-me.org/2013/01/repeat-after-me-mysql-is-not-a-filesystem/

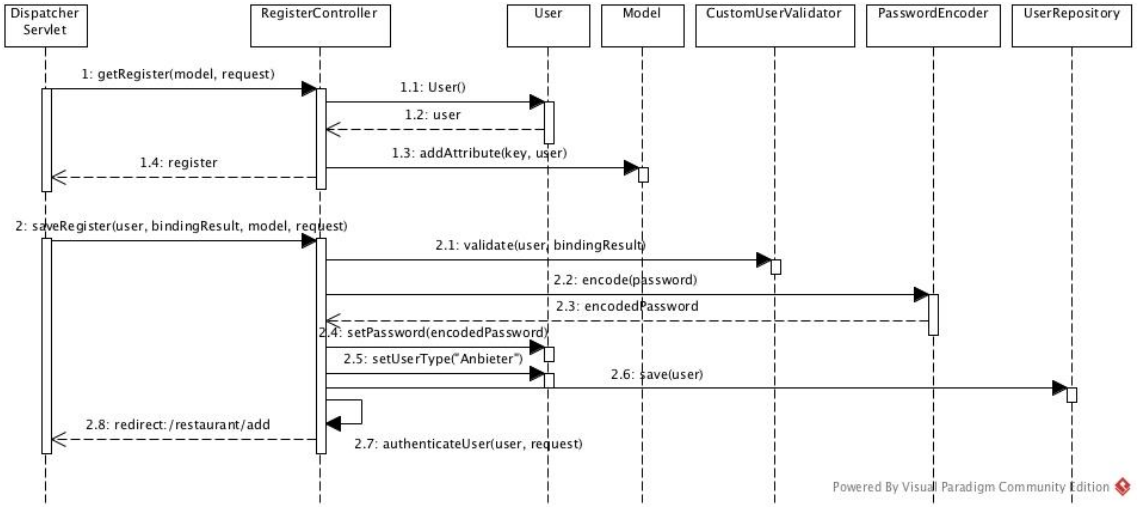

Abbildung 3: Ablauf bei der Registrierung eines neuen Benutzers

Tabelle 8: Beschreibung der Webserver-Pfade

Pfad	Beschreibung
/home, /	Startseite der Webanwendung
/login	Login der Anbieter
/logout	Logout der Anbieter
/register	Registrierung neuer Anbieter
/privacy	Datenschutzvereinbarung für die Website
/terms	Geschäftsbedingungen für die Website
/restaurant/add	Anlegen des Restaurants des Anbieters (nur einmal möglich)
/offer	Übersicht aller Angebote
/offer/edit/[offer_id]	Details des Angebots mit der ID [offer_id]
/offer/add	Neues Angebot erstellen
/offer/delete/[offer_id]	Löschen des Angebots mit der ID [offer_id]
/faq_customer	FAQ für die App
/faq_restaurant	FAQ für die Website
/about_findlunch	Über Findlunch: Wichtige Informationen über die Webseite

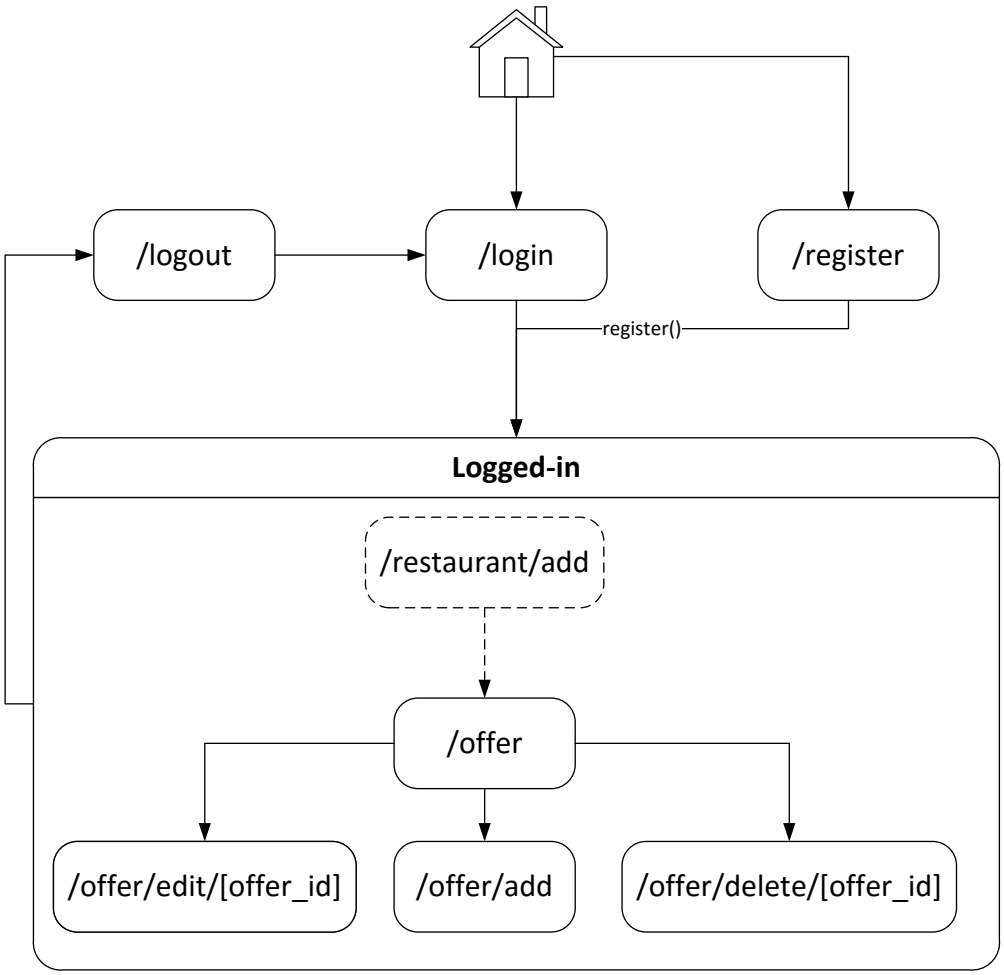

Abbildung 4: Ablaufdiagramm der Webserver-Pfade

4.3.1 Schnittstellen

Der Server nutzt die *Google Geocoding API*, um die eingegebenen Adressen von Restaurants in Koordinaten (Längen- und Breitengrad) umzuwandeln [5].

4.4 Push-Dienst

Das FindLunch-Sytem beinhaltet einen Push-Dienst über den der Kunde regelmäßig über anbietende Restaurants informiert werden kann.

4.4.1 Google Cloud Messaging

Die Push-Benachrichtigungen werden über den *Google Cloud Messaging (GCM)* Service[6] verteilt. Da die prototypische Implementierung der App auf Android durchgeführt wird, bietet sich dieser Dienst an. GCM ist Bestandteil der Google Play Services, die

[5] Details zur Google Geocoding API: https://developers.google.com/maps/documentation/geocoding/intro
[6] Details zum Google Cloud Messaging Service: https://developers.google.com/cloud-messaging/gcm

bereits in Android integriert sind[7]. Der Vorteil dieser Integration ist der sparsamere Umgang mit Ressourcen: Durch die tiefe Integration in das System können die Google Play Services in dessen Energiesparfunktion berücksichtigt werden. Ein eigener Listener müsste das Gerät (zusätzlich zu den Google Play Services) regelmäßig aus dem Standby aufwecken um auf neue Nachrichten zu prüfen. Darüber hinaus ist der GCM-Dienst auch unter iOS nutzbar, was eine zukünftige Implementierung auf diesem System vereinfacht.

Der GCM-Dienst kann Nachrichten über das HTTP- oder das XMPP-Protokoll austauschen. Da XMPP asynchron arbeitet hat es den Vorteil, mehrere Nachrichten zur selben Zeit übertragen zu können. Außerdem kann der Client eine Nachricht als „Antwort“ auf die Push-Benachrichtigung (Upstream) senden. Da dies für die prototypische Implementierung der App nicht benötigt wird, wird das verbreitetere HTTP -Protokoll verwendet.

Da über GCM nur Nachrichten mit 4KB Nutzlast versendet werden können, werden lediglich die für den Abruf der Restaurants und deren Angebote benötigten Parameter übertragen. Der Abruf selbst wird anschließend durch die App durchgeführt. Um das übertragene Datenvolumen gering zu halten, da dieses bei den meisten Mobilfunkverträge begrenzt ist, wird dieser Abruf erst nach dem Öffnen der Push-Benachrichtigung durchgeführt.

4.4.2 Ablauf des Push-Dienstes

Abbildung 5 zeigt den Ablauf des Push-Dienstes, dessen einzelne Schritte im Folgenden beschrieben werden:

1. Nach dem erfolgreichen Login des Benutzers startet die Funktion `MainActivity.onRestUserLoginFinished()` die Klasse `GcmRegistrationIntentService`. Diese ruft über die GCM-Funktion `getToken()` den GCM-Token vom Google Server ab. Dieser Token dient der eindeutigen Identifizierung des Geräts beim Versand der Push-Benachrichtigung. Er wird über einen Broadcast an die `MainActivity` zurückgesendet und zusammen mit den `userLoginCredentials` gespeichert.
2. Wenn der Benutzer eine Push-Benachrichtigung anlegt, wird der Webservice-Endpunkt `register_push` aufgerufen und der GCM-Token an den Server gesendet. Wie in Kapitel 4.2.1.4 beschrieben, wird der GCM-Token zusammen mit jeder einzelnen Push-Benachrichtigung gespeichert. Dies hat den Vorteil, dass ein Kunde unterschiedliche Push-Benachrichtigungen auf unterschiedlichen Geräten - die jeweils unterschiedliche GCM-Tokens haben - anlegen kann.
3. Der Scheduler auf dem Server überprüft, ob Push-Benachrichtigungen gesendet werden müssen. Für jede Push-Benachrichtigung wird eine JSON-Nachricht aufgebaut. Diese beinhaltet neben den Nutzdaten eine eindeutige Sender ID, anhand

[7] Details zu den Google Play Services: https://developers.google.com/android/guides/overview

derer GCM den FindLunch-Service authentifiziert, und den GCM-Token. Die Nachricht wird an den Webservice des GCM-Servers gesendet.

4. Der GCM-Server leitet die Nutzdaten der Nachricht an das Smartphone weiter, das er anhand des GCM-Tokens adressiert. Dort nimmt der Android-Dienst Google Play Services diese entgegen. Die App nutzt eine von Google erzeugte Konfigurationsdatei, um sich für den Empfang der Nachrichten bei diesem Dienst zu registrieren[8] (siehe Anhang I: JSON zur Konfiguration des Push-Handlings). Innerhalb der App nimmt die Methode `PushListenerService.onMessageReceived()` die Nachricht entgegen und verarbeitet den Inhalt. Die App zeigt in der Benachrichtigungsleiste von Android eine Nachricht an. Wenn der Anwender diese Nachricht öffnet, wird eine Liste von Restaurants in der App angezeigt.

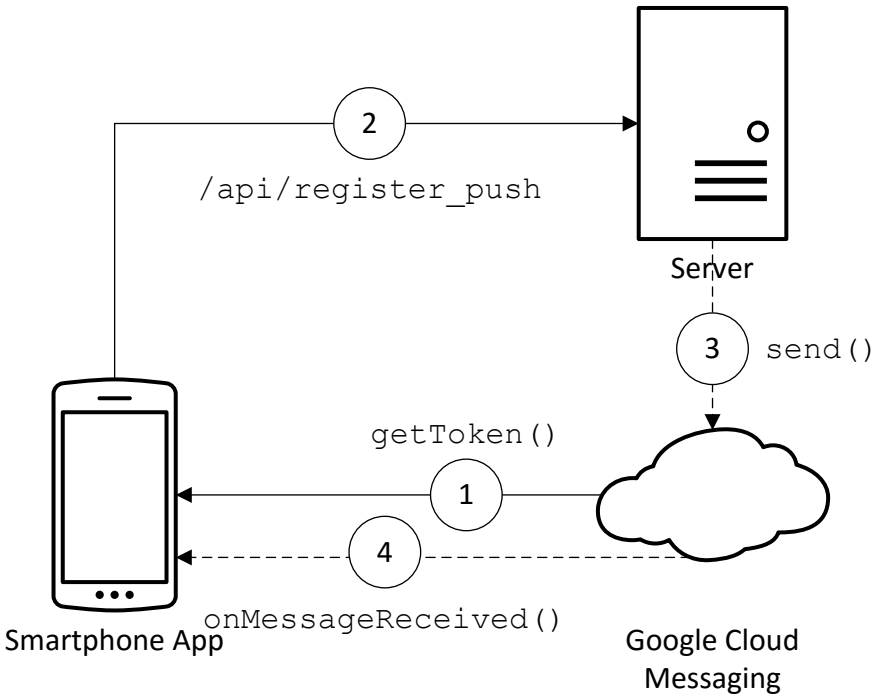

Abbildung 5: Ablauf des Push-Dienstes

4.4.3 Senden der Push-Benachrichtigung

Im Server wird ein *TaskScheduler*[9] implementiert, der täglich um 9:00 Uhr überprüft, ob Einträge in der Tabelle restaurant den Bedingungen in der Tabelle push_notification entsprechen. Für diese Einträge wird eine Nachricht erzeugt und an den GCM-Dienst (POST an https://gcm-http.googleapis.com/gcm/send) gesendet. Tabelle 9 beschreibt den Inhalt dieser JSON-Nachricht. Der GCM-Dienst kümmert sich anschließend um die Auslieferung der Nachricht an das Smartphone des Kunden.

[8] Details zu dieser Konfigurationsdatei: https://developers.google.com/cloud-messaging/android/client

[9] Details zum Task Scheduling des Spring Frameworks: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/scheduling.html

Tabelle 9: Inhalt der Nachricht an den GCM

Position	Titel	Inhalt
Header	Content-Type	application/x-www-form-urlen-coded;charset=UTF-8
Header	Authorization:key	AIzaSyDa7qzHVlNw5c5slr7 D_DqAih6eBb3qB0g
Header	to	push_notification.gcm_token
Body	data:title	push_notification.title
Body	data:numberOfRestaurants	Anzahl der Restaurants
Body	data:longitude	push_notification.longitude
Body	data:latitude	push_notification.latitude
Body	data:radius	push_notification.radius
Body	data:kitchenTypeIds	kitchen_type.id
Body	data:pushId	push_notification.id

4.5 Webservice

Die Smartphone-App greift auf die Funktionen eines RESTful Webservices zu. Dieser bietet verschiedene Endpunkte an, die von der App (mit den entsprechenden Parametern) aufgerufen werden. Die Rückgabewerte werden in eine JSON-Datei geschrieben, die an die App gesendet wird. Alle spezifizierten Parameter müssen beim Aufruf angegeben werden.

4.5.1 Endpunkte

Der Webservice bietet folgende Endpunkte an:

WE01	**Abrufen von Restaurants in der Nähe**
http-Methode	GET
Pfad	/api/restaurants?latitude=[lat]&longitude=[long]&radius=[rad]
Parameter	[lat]: Breitengrad der Position des Kunden (Float) [long]: Längengrad der Position des Kunden (Float) [rad]: Suchradius um die Position des Kunden in Metern (Integer)
Antwort	Liste von Restaurant-Objekten die sich im Radius [rad] um den Standort (definiert durch [lat]/[long]) befinden. Jedes Objekt beinhalten folgende Informationen: - Daten aus der Tabelle restaurant - Name des zu restaurant.country gehörenden Landes - Name des zu restaurant.restaurant_type_id gehörenden Restauranttyps - Liste der zugehörigen kitchenType-Objekte (aus der Tabelle kitchen_type) - Liste der zugehörigen timeSchedule-Objekte (aus der Tabelle time_schedule) - Ein Integer-Feld Distance, das die Distanz zwischen dem Standort des Restaurants und dem Standort des Kunden angibt - Ein Boolean-Feld isFavorite das immer als false zurückgibt

WE02	**Abrufen von Restaurants in der Nähe (für angemeldete Benutzer)**
http-Methode	GET
Pfad	/api/restaurants?latitude=[lat]&longitude=[long]&radius=[rad]
Parameter	[lat]: Breitengrad der Position des Kunden (Float) [long]: Längengrad der Position des Kunden (Float) [rad]: Suchradius um die Position des Kunden in Metern (Integer)
Header	Authorization (HTTP Basic Auth)
Antwort	Neben den in WE01 spezifizierten Inhalten zeigt das Feld isFavorite an, ob das Restaurant ein Favorit des authentifizierten Kunden ist.

WE03	**Abrufen aller momentanen Angebote des Restaurants mit der ID [restaurant_id]**
http-Methode	GET
Pfad	/api/offers?restaurant_id=[restaurant_id]
Parameter	[restaurant_id]: ID des Restaurants (Integer)
Antwort	Liste aller Offer-Objekte mit - $offer.restaurant\_id == [restaurant\_id]$ & - $offer.start\_date \leq today \leq offer.end\_date$ & - $offer\_has\_day\_of\_week.day\_of\_week\_id == today$ & - $time\_schedule.offer\_start\_time \leq now \leq time\_schedule.offer\_end\_time$ Jedes Objekt beinhalten folgende Informationen: - Daten aus der Tabelle offer - Ein BLOB-Feld defaultPhoto das den, sortiert nach ID, ersten Eintrag in offer_photo.thumbnail enthält, der zu dem jeweiligen Offer gehört.

WE04	**Abrufen der Fotos zum Angebot mit der ID [offer_id]**
http-Methode	GET
Pfad	/api/offer_photos?offer_id=[offer_id]
Parameter	[offer_id]: ID des Angebots (Integer)
Antwort	Liste aller OfferPhoto-Objekte aus der Tabelle offer_photo mit $offer\_photo.offer\_id == [offer\_id]$

<table>
<tr><td>WE05</td><td>Registrieren eines neuen Benutzers</td></tr>
<tr><td>http-Methode</td><td>POST</td></tr>
<tr><td>Pfad</td><td>/api/register_user</td></tr>
<tr><td>Anfrage (JSON)</td><td>Ein User-Objekt, das dem Kunden entspricht, der registriert wird. Es enthält die Informationen username und password.</td></tr>
<tr><td>Antwort</td><td>Status Code:
<table>
<tr><th>intern</th><th>HTTP</th><th>Beschreibung</th></tr>
<tr><td>0</td><td>200</td><td>Erfolgreich</td></tr>
<tr><td>1</td><td>409</td><td>Ungültiger Benutzername (entspricht nicht den Richtlinien)</td></tr>
<tr><td>2</td><td>409</td><td>Ungültiges Passwort (entspricht nicht den Richtlinien)</td></tr>
<tr><td>3</td><td>409</td><td>Benutzername bereits vorhanden</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>WE06</td><td>Login eines Benutzers</td></tr>
<tr><td>http-Methode</td><td>GET</td></tr>
<tr><td>Pfad</td><td>/api/login_user</td></tr>
<tr><td>Header</td><td>Authorization (HTTP Basic Auth)</td></tr>
<tr><td>Antwort</td><td>Status Code:
<table>
<tr><th>intern</th><th>HTTP</th><th>Beschreibung</th></tr>
<tr><td>0</td><td>200</td><td>Erfolgreich</td></tr>
<tr><td></td><td>401</td><td>Login fehlgeschlagen</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>WE07</td><td>Anmelden einer Push-Benachrichtigung</td></tr>
<tr><td>http-Methode</td><td>POST</td></tr>
<tr><td>Pfad</td><td>/api/register_push</td></tr>
<tr><td>Header</td><td>Authorization (HTTP Basic Auth)</td></tr>
<tr><td>Anfrage (JSON)</td><td>Das PushNotification-Objekt, das in der Datenbank abgespeichert wird. Dieses beinhaltet folgende Informationen:
Die in der Tabelle push_notification definierten Informationen, außer der ID.
Ein User-Objekt, das dem User entspricht, für den die Push-Benachrichtigung angelegt wird. Es enthält die Informationen username und password.
Eine Liste von DayOfWeek-Objekten, die den Wochentagen entsprechen, an denen die Push-Benachrichtigung versendet wird. Nur das Feld name ist in diesen Objekten befüllt. Die restlichen Felder haben den Wert NULL.
Eine Liste von KitchenType-Objekten, die den Küchentypen entsprechen, für die die Push-Benachrichtigung versendet wird.</td></tr>
<tr><td>Antwort</td><td>Status Code:<table><tr><th>intern</th><th>HTTP</th><th>Beschreibung</th></tr><tr><td>0</td><td>200</td><td>Erfolgreich</td></tr><tr><td></td><td>401</td><td>Login fehlgeschlagen</td></tr><tr><td>4</td><td>409</td><td>Kein Wochentag ausgewählt</td></tr></table></td></tr>
</table>

WE08	**Abrufen der aktiven Push-Benachrichtigungen**
http-Methode	GET
Pfad	/api/get_push
Header	Authorization (HTTP Basic Auth)
Antwort	Liste von PushNotification-Objekten die zum authentifizierten Benutzer gehören. Jedes Objekt beinhaltet folgende Informationen: - Die Informationen ID und Titel aus der Tabelle push_notification - Eine Liste von DayOfWeek-Objekten, die den Wochentagen entsprechen, an denen die Push- Benachrichtigung versendet wird. - Eine Liste von KitchenType-Objekten, die den Küchentypen entsprechen, für die die Push- Benachrichtigung versendet wird. Falls der Kunde nicht angemeldet ist, wird eine HTTP-Code 401 zurückgegeben.

WE09	**Abmelden von Push-Benachrichtigung**
http-Methode	DELETE
Pfad	/api/unregister_push/[push_id]
Parameter	[push_id]: ID der zu löschenden PushNotification (Integer)
Header	Authorization (HTTP Basic Auth)
Antwort	Status Code:

intern	**HTTP**	**Beschreibung**
0	200	Erfolgreich
	401	Login fehlgeschlagen
3	409	PushNotification ID ungültig (nicht vorhanden oder gehört nicht dem authentifizierten Benutzer)

<table>
<tr><td>WE10</td><td>Anlegen eines Favoriten</td></tr>
<tr><td>http-Methode</td><td>PUT</td></tr>
<tr><td>Pfad</td><td>/api/register_favorite/[restaurantId]</td></tr>
<tr><td>Parameter</td><td>[restaurantId]: ID des Restaurants, das als Favorit angelegt werden soll (Integer)</td></tr>
<tr><td>Header</td><td>Authorization (HTTP Basic Auth)</td></tr>
<tr><td>Antwort</td><td>Status Code:
<table>
<tr><th>intern</th><th>HTTP</th><th>Beschreibung</th></tr>
<tr><td>0</td><td>200</td><td>Erfolgreich angelegt</td></tr>
<tr><td></td><td>401</td><td>Login fehlgeschlagen</td></tr>
<tr><td>3</td><td>409</td><td>Restaurant ID ungültig</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>WE11</td><td>Löschen eines Favoriten</td></tr>
<tr><td>http-Methode</td><td>DELETE</td></tr>
<tr><td>Pfad</td><td>/api/unregister_favorite/[restaurantId]</td></tr>
<tr><td>Parameter</td><td>[restaurantId]: ID des Restaurants, das als Favorit gelöscht werden soll (Integer)</td></tr>
<tr><td>Header</td><td>Authorization (HTTP Basic Auth)</td></tr>
<tr><td>Antwort</td><td>Status Code:
<table>
<tr><th>intern</th><th>HTTP</th><th>Beschreibung</th></tr>
<tr><td>0</td><td>200</td><td>Erfolgreich</td></tr>
<tr><td></td><td>401</td><td>Login fehlgeschlagen</td></tr>
<tr><td>3</td><td>409</td><td>Restaurant ID ungültig</td></tr>
</table></td></tr>
</table>

WE12	**Abrufen der Küchentypen**
http-Methode	GET
Pfad	/api/kitchen_types
Antwort	Liste von KitchenType-Objekten aus der Tabelle kitchen_type.

WE13	**Abrufen der Restauranttypen**
http-Methode	GET
Pfad	/api/restaurant_types
Antwort	Liste von RestaurantType-Objekten aus der Tabelle restaurant_type.

WE14	**Abrufen der Wochentage**
http-Methode	GET
Pfad	/api/ days_of_week
Antwort	Liste von DayOfWeek-Objekten aus der Tabelle day_of_week.

4.5.2 Transaktionskonzept

Da es sich bei REST um ein zustandsloses Protokoll handelt, ist es nicht sinnvoll eine Transaktion am Client zu beginnen. Daher werden alle Transaktionen innerhalb des Servers, basierend auf Informationen die der Client zur Verfügung stellt, ausgeführt.

4.6 Backup-Strategie

Zum Backup der in der Datenbank gespeicherten Daten wird ein Prozess auf Basis des mit MariaDB ausgelieferten Tools *mysqldump* genutzt. Dieses sperrt für einen kurzen Zeitraum den Zugriff auf die Tabellen, um sowohl ihre Struktur (Data Definition Language), als auch ihren Inhalt (Data Manipulation Language) als SQL-Skript zu exportieren. Durch ein Ausführen des Skripts kann die Datenbank komplett wiederhergestellt werden.

Es handelt sich hierbei um ein *Warm Backup* bei dem, im Gegensatz zu einem Cold Backup, der Server nicht heruntergefahren werden muss. Anders als bei einem Hot Backup muss allerdings eine kurze Serviceunterbrechung in Kauf genommen werden.

Zum Speichern der Backup-Dateien wird ein zweiter Server benötigt. Dieser sollte auf Hardware laufen, die von der des Hauptservers unabhängig ist – beispielsweise auf einem physikalisch eigenen Server. Im folgenden Setup wird hierzu der in Tabelle 10 beschriebene Server genutzt. Die Verbindung mit der Netzwerkfreigabe dieses Servers wird nur hergestellt, wenn sie benötigt wird. Hierdurch ist es für Schadsoftware schwieriger, auf die Backups zuzugreifen.

Tabelle 10: Parameter des Servers für Backups

IP-Adresse	X.X.X.61
Samba-Freigabe (zum Ablegen der Backups)	\\X.X.X.61\Backup
Benutzer auf dem Server (mit Zugriff auf Freigabe)	sys_findlunch_backup
Passwort des Benutzers	XXXXXXXXXX

Auf dem Datenbankserver werden zwei Batchdateien angelegt:

1. Die Datei Create_Backup.bat (siehe Abbildung 6) führt das Backup der Datenbank durch und nutzt das Pipe-Konzept, um dieses Backup zu komprimieren. Die komprimierte Datei, deren Namen aus dem Backup-Zeitpunkt besteht, wird anschließend auf dem Server abgelegt.
2. Die Datei Clean_Backups.bat (siehe Abbildung 7) löscht alle Backups die älter als 30 Tage sind, um Speicherplatz zu sparen.

In der Windows Aufgabenplanung (%windir%\system32\taskschd.msc) werden zwei neue Aufgaben angelegt:

1. „FindLunch Backup erstellen“ startet „Täglich um 23:00 Uhr“ die Datei „Create_Backup.bat“.
2. „FindLunch Backups bereinigen“ startet „Alle 2 Wochen um 04:00 am Sonntag“ die Datei „Clean_Backups.bat“

Bei beiden Jobs ist wichtig, dass sie „unabhängig von der Benutzeranmeldung“ ausgeführt und „mit höchsten Berechtigungen“ gestartet werden.

```
REM Sicherstellen das S: nicht verwendet wird und Netzlaufwerk
verbinden
net use /delete s:
net use s: \\X.X.X.61\Backup XXXXXXXXXX /user:sys_findlunch_backup

REM mysqldump ausführen
cd "C:\Program Files\MariaDB 10.1\bin\"
mysqldump --single-transaction -u db_findlunch_backup -pXXXXXXXXXX
findlunch|"C:\Program Files\7-Zip\7z" a -sibackup.sql
S:\backup_%date:~-4,4%%date:~-7,2%%date:~-
10,2%_%time:~0,2%%time:~3,2%%time:~6,2%.7z

REM Netzlaufwerk entfernen
net use /delete s:
```

Abbildung 6: Inhalt der Datei Create_Backup.bat

```
REM Sicherstellen das S: nicht verwendet wird und Netzlaufwerk
verbinden
net use /delete s:
net use s: \\X.X.X.61\Backup XXXXXXXXXX /user:sys_findlunch_backup

REM Alte Backups nach _old verschieben und loeschen.
robocopy s:\ s:\_old /mov /minage:30
del s:\_old /q

REM Netzlaufwerk entfernen
net use /delete s:
```

Abbildung 7: Inhalt der Datei Clean_Backups.bat

4.6.1 Ausblick

Der Flaschenhals der beschriebenen Backup-Lösung ist die Netzwerkverbindung zwischen den beiden Servern. Da es sich um ein Warm Backup handelt, bestimmt diese die Dauer während der die Datenbank nur eingeschränkt zur Verfügung steht.

Unter der Annahme einer Gigabitverbindung zwischen den Servern (1.000 Mbit/s) und 4% Overhead im Netzwerkverkehr (40 Mbit/s) beträgt die Übertragungsrate 960 Mbit/s oder 120 MB/s. Während das Backup einer ein Gigabyte großen Datenbank mit 8 Sekunden noch akzeptabel ist, dauert es bei drei Gigabyte bereits 25 Sekunden. Bei Datenbanken dieser Größe sollte das Backup zunächst lokal durchgeführt, anschließend gepackt und auf den Backup-Server verschoben werden.

Bei noch größeren Datenbanken (ab 5-10 GB) sollte ein Hot Backup durchgeführt werden, um längere Unterbrechungen der Verfügbarkeit zu vermeiden. Neben dem Einsatz proprietärer Lösungen, wie z.B. MySQL Enterprise Backup von Oracle[10], ist es möglich den Volume Snapshot Service von Windows zu nutzen[11].

[10] Details siehe http://dev.mysql.com/doc/mysql-enterprise-backup/4.0/en/

[11] Wie auf http://karlssonondatabases.blogspot.de/2013/11/mariadb-database-disk-snapshot-backups.html beschrieben

5 Clients

Anbieter und Kunden nutzen jeweils eigene Plattformen (Rechner bzw. Smartphones) um auf die für sie relevanten Funktionalitäten des FindLunch-Systems zugreifen zu können. Die Clients werden daher für die Nutzung auf der jeweiligen Plattform optimiert.

5.1 Browser (Anbieter)

Das Webportal ist für die Nutzung mit Firefox (Version 45 oder höher) optimiert.

5.2 Smartphone-App (Kunde)

Die App ruft je nach Nutzereingabe die Funktionen des Webservice auf, der die benötigten Daten zurückliefert bzw. die serverseitigen Aktionen auslöst.

Der Kunde hat die Möglichkeit sich in der App anzumelden. Hierbei wird über den Webservice-Endpunkt *login_user* (siehe Kapitel 4.5) geprüft, ob Benutzername und Passwort gültig sind. Falls ja, wird der Status „eingeloggt“ in der App angezeigt und Benutzername und Passwort lokal gespeichert. Beim Abmelden werden der Status und die Login-Daten gelöscht.

5.2.1 Verteilung

Im Produktivbetrieb erfolgt die Verteilung der App über den Google Play Store. Dabei wird die App als APK-Datei auf die Server von Google geladen[12] und veröffentlicht[13]. Anschließend können Kunden die App über den in Android integrierten Google Play Store installieren. Auch die Verteilung von App-Aktualisierungen erfolgt über dieses Verteilungssystem[14].

5.2.2 Schnittstellen

Die App nutzt die Google Geocoding API um die eingegebene Adresse in Koordinaten (Längen- und Breitengrad) umzuwandeln. Dies ist für die serverseitige Überprüfung, welche Restaurants sich in der Umgebung des Kunden befinden, erforderlich. Eine clientseitige Umwandlung der Adresse benötigt mehr Ressourcen auf dem Smartphone und erhöht die Menge der übertragenen Daten, da das Smartphone mit dem Google Server

[12] Details zum Hochladen von Apps in den Google Play Store: https://support.google.com/googleplay/android-developer/answer/113469?hl=de&ref_topic=3450986

[13] Details zum Veröffentlichen von Apps im Google Play Store: https://support.google.com/googleplay/android-developer/answer/6334282?hl=de&ref_topic=3450986

[14] Details zum Aktualisieren von Apps im Google Play Store: https://support.google.com/googleplay/android-developer/answer/113476?hl=de&ref_topic=3450986

kommunizieren muss. Dennoch wurde diese Variante gewählt, da bei einer Positionsbestimmung durch GPS (beispielsweise in einer zukünftigen Version) die Position auch in Koordinaten vorliegt – und somit derselbe Webservice-Endpunkt genutzt werden kann.

Zur Darstellung der gefundenen Restaurants in einer Kartenansicht wird die *Google Maps API* genutzt[15].

Nach dem erfolgreichen Login des Benutzers erzeugt die App einen gerätespezifischen Registration Token für den GCM-Dienst und speichert ihn lokal. Dieser wird für die Anmeldung zum Push-Dienst benötigt.

5.2.3 Plattform

Obwohl nur die Umsetzung einer Android-App gefordert ist, stellt sich dennoch die Frage, ob diese *nativ* oder *cross-platform* umgesetzt werden soll. Im Folgenden werden beide Ansätze miteinander verglichen und die Entscheidungsfindung dokumentiert.

Eine *native App* wird mit dem vom jeweiligen Hersteller bereitgestellten SDK in der für die jeweilige Plattform vorgesehenen Programmiersprache entwickelt. Folglich ist sie auch nur auf diesen Geräten lauffähig. Die Vorteile dieses Ansatzes sind eine gute Benutzererfahrung (die Oberfläche entspricht weitgehend den Designvorgaben) und das keine Limitierungen bestehen (insbesondere bei Zugriff auf native Funktionen des Endgeräts). Da das SDK von einem großen Hersteller bereitgestellt wird, existieren eine ausführliche Dokumentation, sowie eine aktive Community (aufgrund des hohen Verbreitungsgrads der SDK) [16]. Des Weiteren ist man nicht auf die Funktionalität der Software von Drittanbietern und Plug-Ins für den Zugriff auf Funktionen des Endgeräts angewiesen.

Eine *cross-platform App* ähnelt einer „lokalen Webseite“, die einmalig erstellt wird. Deren Darstellung ist mittels der Browser-Komponente des jeweiligen Betriebssystems auf allen Betriebssysteme (z.B. Android, iOS und Windows Mobile) möglich. Um die App in den App-Stores zur Verfügung zu stellen, wird sie in eine mit dem jeweiligen SDK entwickelten native App „gepackt“. Diese App benötigt nur wenig Funktionalität (Darstellung des Inhalts), weshalb der Aufwand für Ihre Entwicklung gering ist[17].

Dieser Ansatz wird auch „hybrid“ genannt. Seine Vorteile sind geringere Kosten (Hauptarbeit bei der Entwicklung muss nur einmal für alle Plattformen durchgeführt werden),

[15] Details zur Google Maps API: https://developers.google.com/maps/documentation/android-api/intro

[16] Rosario Madaudo und Patrizia Scandurra, „Native versus Cross-platform frameworks for mobile application development“, 2013, http://2013.eclipse-it.org/proceedings/6_Madaudo-Scandurra.pdf.

[17] Henning Heitkötter, Hanschke Sebastian, und Tim A. Majchrzak, „Evaluating Cross-Platform Development Approaches for Mobile Applications“, 2014, http://www3.nd.edu/~cpoellab/teaching/cse40814/crossplatform.pdf.

die höhere Reichweite (einstellen in mehreren App-Stores) und die einfachere Pflege (die Entwickler arbeiten nur in einer Programmiersprache)[18].

Auf Empfehlung der Entwickler hat das Projektteam am 15.04.2016 entschieden, den Client als native App umzusetzen. Folgende Gründe haben zu dieser Entscheidung geführt:

1. Aufgrund des prototypischen Charakters des Projekts hat die Umsetzung einer grafisch ansprechenden Anwendung (für Demonstrationszwecke) eine höhere Priorität als ein möglichst hoher Verbreitungsgrad).
2. Da im Projektteam nur zwei iPhones und keine Windows Mobile Phones vorhanden sind, wäre es schwierig die Anwendung auf diesen Plattformen zu testen. Die Anschaffung weiterer Geräte wäre angesichts des Kosten/Nutzen-Aspekts nicht verhältnismäßig.
3. Das Entwicklerteam hat bereits Vorkenntnisse im Bereich Android-Entwicklung. Zusätzlich existiert, wie beschrieben, für die native Entwicklung eine umfangreiche Dokumentation und eine große Community – was bei der Suche nach Problemlösungen hilfreich sein kann. Somit ist das Risiko falscher Abschätzung und der Überschreitung des Zeitrahmens bei einer nativen Entwicklung geringer.

[18] Madaudo und Scandurra, „Native versus Cross-platform frameworks for mobile application development“.

6 Mehrsprachigkeit

Das FindLunch-System wurde so entworfen, dass es mit möglichst geringem Aufwand in andere Sprachen übersetzt werden kann. Hierzu müssen die verschiedenen Texte in der App und Web-Anwendung sowie die Typen-Namen in der Datenbank übersetzt werden. Dieses Kapitel beschreibt, wie das System übersetzt werden kann.

6.1 App

Alle Texte die in der Oberfläche angezeigt werden befinden sich in der Ressource values/strings.xml. Für die Unterstützung weiterer Sprachen können weitere Ordner mit dem entsprechenden Sprachsuffix angelegt werden, wie values-fr für Französisch, die eine strings.xml-Datei mit derselben Struktur, aber anderen Einträgen enthalten[19]. Entsprechend der in Android eingestellten Systemsprache wird die jeweilige strings.xml-Datei verwendet[20].

6.2 Webanwendung

Im Pfad /FindLunchServer/src/main/resources/messages befinden sich mehrere properties-Dateien, in denen die auf der Oberfläche angezeigten Texte hinterlegt sind. Für die Unterstützung weiterer Sprachen können weitere Ordner mit dem entsprechenden Sprachsuffix angelegt werden, wie messages_fr für Französisch, in denen die übersetzten properties-Dateien abgelegt werden. Entsprechend der im Browser eingestellten Sprache werden die Einträge aus dem jeweiligen Ordner verwendet.

6.3 Datenbank

Das in Kapitel 4.2.1 beschriebene Datenmodell wurde so gestaltet, dass es die Anforderungen an den Prototypen erfüllt, ohne unnötige Komplexitäten aufzubauen. Gleichzeitig ist es flexibel erweiterbar, um mögliche Anforderungen eines erweiterten Produktivbetriebs abzudecken. Für eine Mehrsprachigkeit der Anwendung müssen für statische Texte, wie Art der Küche, Einträge für alle unterstützen Sprachen gepflegt werden. Für dynamische Inhalte, wie die Namen der Restaurants oder Angebote, ist i.d.R. keine Übersetzung erforderlich bzw. sinnvoll.

[19] Details siehe: https://developer.android.com/training/basics/supporting-devices/languages.html

[20] Details siehe: https://developer.android.com/guide/topics/resources/localization.html#using-framework

6.3.1 Weitere Spalten

Zur Unterstützung weiterer Sprachen können Tabellen mit statischen Texten (country, day_of_week, kitchen_type, restaurant_type und user_type) weitere Spalten hinzugefügt werden, deren Werte dann in der Oberfläche des Systems angezeigt werden (z. B. name_EN_US bzw. day_number_EN_US für US-Englisch).

Dieser Ansatz ist einfach und schnell umzusetzen. Allerdings wächst mit jeder weiteren Sprache die Anzahl der Spalten in mehreren Tabellen. Daher ist er nur für eine geringe Anzahl zusätzlicher Sprachen geeignet.

6.3.2 Übersetzungstabelle

Eine gut skalierende Möglichkeit ist der Einsatz einer eigenen Tabelle in der pro Sprache, Objekt (z.B. kitchen_type) und Eintrag (id aus der Tabelle des Objekts) ein übersetzter Begriff gepflegt wird. Die Einträge in den ursprünglichen Tabellen entsprechen der „Standardsprache“ und werden verwendet, falls für einen Begriff keine Übersetzung gepflegt ist.

Tabelle 11 zeigt beispielhaft Einträge (unter der Annahme, dass der Küchentyp mit der ID 5 „Asiatisch“ heißt). Neben der guten Skalierbarkeit ist ein weiterer Vorteil, dass für die Einführung neuer Sprachen das Datenmodell nicht geändert werden muss.

Tabelle 11: Beispiel für Einträge in der Übersetzungstabelle

ID	Language	Object	Object_ID	Text
1	fr	kitchen_type	5	Asiatique
2	es	kitchen_type	5	Asiático

7 Lasttest: FindLunch-Server

Um Performance und Skalierbarkeit des FindLunch-Systems außerhalb des prototypischen Betriebs abzuschätzen, wird ein einfacher Performance-Test der REST-Schnittstelle zum Abruf der Restaurants durchgeführt und das Ergebnis ausgewertet.

7.1 Testaufbau

Für den Test wird der Einsatz der Anwendung in Berlin-Mitte angenommen. Berlin hat eine Fläche von 892 km^2 und ca. 4650 Restaurants[21]. Unter der Annahme, dass alle Restaurants Mittagsangebote anbieten und bei FindLunch teilnehmen, ergibt sich eine Dichte von 5,2 Restaurants pro km^2. Unter der weiteren Annahme, dass ein Kunde in einem Radius von 1.000m sucht, befinden sich durchschnittlich $\frac{\pi*1000}{1000} * 5{,}2 = 15{,}6 \triangleq 15$ Restaurants in seiner Umgebung. Berlin-Mitte hat etwa 363.000 Einwohner.

7.1.1 Testumgebung

Die Tests werden mit Apache JMeter[22] und folgender Konfiguration durchgeführt:

1. Die eingesetzte Thread-Gruppe hat eine Ramp-Up Periode von 5 Sekunden (Abstand, in dem die Kunden auf den Dienst zugreifen).
2. Ein HTTP-request-Sampler führt ein GET mit dem HttpClient4 auf die Adresse https://findlunch.biz.tm:8443/api/restaurants?latitude=48.1539&longitude=11.554&radius=1000 (was bei den vorhandenen Test-Daten 15 Restaurants zurückliefert) aus.
3. Ein Summary Report-Listener speichert die Ergebnisse. Dieser wird nach jedem Test gelöscht, da andernfalls vorherige Testläufe das Ergebnis verfälschen würden.

JMeter läuft auf einem Rechner mit der folgenden Konfiguration: CPU: Intel Core i5-6600 (3,3 GHz), RAM: 8 GB (2,1 GHz), OS: Windows 10 Professional. Der durchschnittliche Ping auf den Server beträgt 37,48ms.

Die Nutzung von CPU, Datenträger und RAM auf dem Server wird während der Tests mit dem Windows Ressourcenmonitor überwacht. Für die Dauer des Tests wird außerdem die Windows-Firewall deaktiviert, da es andernfalls zu Fehlern durch abgelehnte Verbindungen kommen würde.

[21] Quelle: http://www.visitberlin.de/de/artikel/fakten-und-zahlen

[22] Details siehe: https://jmeter.apache.org/

7.2 Testdurchführung

Für die einzelnen Tests wird die Anzahl der eingesetzten Threads variiert. Dies simuliert eine unterschiedliche Anzahl von Nutzern, die auf den Dienst zugreifen.

7.3 FindLunch-Test 1: Hardware für prototypischen Betrieb

Ablauf: Für den Test wird die in Kapitel 4 spezifizierten Server-Hardware verwendet:

- CPU: 2 GHz Dual-Core
- RAM: 4 GB

Auswertung: Wie in Tabelle 12 zu sehen ist, reicht die derzeit genutzte Hardware für einen produktiven Betrieb nicht aus: Spätestens bei 363 gleichzeitigen Zugriffen sind die Antwortzeiten nicht mehr praktikabel. Bei 1.452 wird weniger als die Hälfte der Anfragen noch beantwortet.

Die Beobachtung des Ressourcenmonitors auf dem Server zeigte, dass die CPU bei den Tests am stärksten beansprucht wird. Die Auslastung des RAM lag konstant bei 73-75% und auch beim Datenträgerzugriff zeigten sich nur einzelne Leistungsspitzen.

Tabelle 12: Ergebnis des Lasttests FindLunch-Test 1

Anteil der Bevölkerung	Anzahl Anfragen	Antwortzeit (ms)					
		Min	Max	Mean	Std. Dev.	% Fehler	KB/sec
-	10	267	377	310	38,99	0,00%	27,53
0,05%	181	755	7.839	4.153	1.563,84	0,00%	279,56
0,10%	363	1.236	19.259	12.829	2.397,65	0,00%	239,14
0,20%	726	15.807	49.565	36.679	5.358,53	0,00%	192,77
0,30%	1089	3.508	56.063	38.087	11.624,34	8,72%	219,62
0,40%	1452	2.997	32.761	13.319	10.298,14	56,96%	311,93

7.4 FindLunch-Test 2: Verbesserte Hardware

Ablauf: Für diesen Test wird die Hardware der Virtual Machine auf folgende Komponenten verbessert:

- CPU: 3 GHz Quad-Core (Takt: +50%, Kerne: +100%)
- RAM: 8 GB (+100%)

Auswertung: Tabelle 13 zeigt, dass sich die Leistung des Servers deutlich verbessert hat. Bei 726 gleichzeitigen Zugriffen ist die durchschnittliche Antwortzeit noch akzeptabel und selbst bei 1.452 werden fast alle Anfragen beantwortet.

Auch bei diesem Test war die CPU am stärksten ausgelastet (am Anfang zu 100%). Bei größeren Mengen von Benutzern bzw. höheren Anforderungen an die Performance könnte man sie noch weiter upgraden.

Tabelle 13: Ergebnis des Lasttests FindLunch-Test 2

Anteil der Bevölkerung	Anzahl Anfragen	Antwortzeit (ms)					
		Min	Max	Mean	Std. Dev.	% Fehler	KB/sec
-	10	353	814	456	138,19	0,00%	24,25
0,05%	181	476	1847	1.085	273,43	0,00%	356,77
0,10%	363	503	9.589	4.752	1.888,8	0,00%	365,49
0,20%	726	4.283	31.740	14.580	3.696,71	0,00%	247,78
0,30%	1089	8.175	68.130	26.024	6.430,72	0,55%	183,52
0,40%	1452	6.303	79.957	30.392	8.545,52	0,28%	206,49

7.5 Bewertung des Ergebnisses

Wie der Vergleich der beiden Tests in Abbildung 8 zeigt, skaliert der FindLunch-Server sehr gut zusammen mit der Hardware. Durch die Verdopplung der Hardwareleistung konnte die Antwortzeit bei großen Anfragemengen mehr als halbiert werden (von 36.679ms auf 14.580ms). Außerdem treten bei mehr als 1.000 Anfragen kaum noch Fehler auf.

Somit kann das FindLunch-System (die passende Hardware vorausgesetzt) auch in größerem Maßstab eingesetzt werden. Neben verbesserter Hardware gibt es noch andere Möglichkeiten, um die Performance des Systems zu verbessern (siehe hierzu auch Kapitel 4.2.2). Diese sollten für einen produktiven Einsatz gezielt untersucht werden.

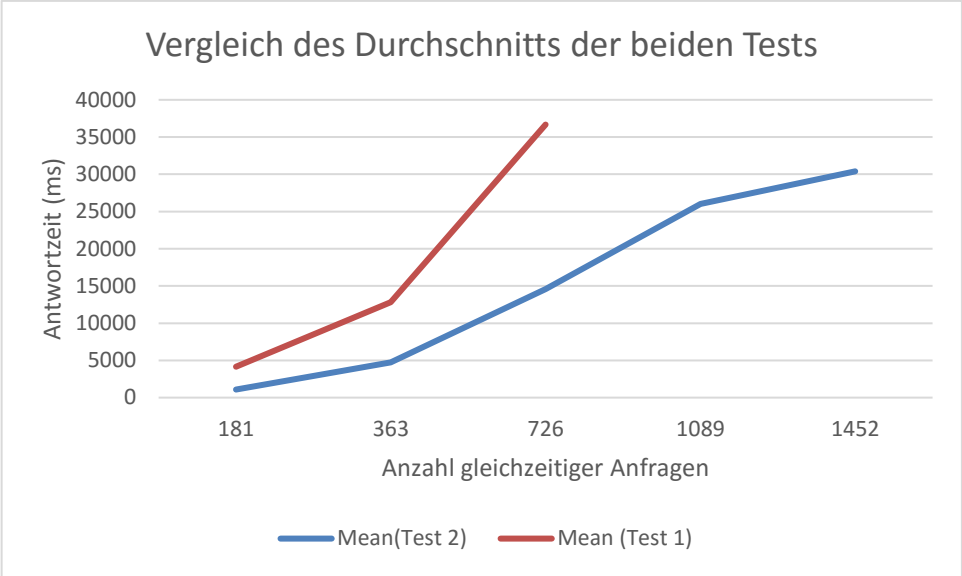

Abbildung 8: Vergleich der durchschnittlichen Antwortzeit in beiden FindLunch-Tests

8 Performance-Test: GCM

Bei dem für die Implementierung des Push-Dienstes (siehe Kapitel 4.4) verwendeten GCM handelt es sich um einen externen Dienst, der nicht unter der Kontrolle des Find-Lunch-Servers steht. Daher soll im Folgenden dessen Leistungsfähigkeit unter erhöhter Last überprüft werden. Das Ziel ist herauszufinden, ob auch bei einem erhöhten Datenvolumen eine konstante Leistung gewährleistet wird.

8.1 Testaufbau

Die folgenden Rechner befinden sich in einem LAN, das ans Internet angebunden ist (Bruttoraten: 25 Mbit/s Downstream & 1 Mbit/s Upstream):

- Rechner 1: CPU: Intel Core i5-6600 (3,3 GHz), RAM: 8 GB (2,1 GHz), OS: Windows 10 Professional
    - Android Virtual Device (AVD) 1: RAM: 2048 MB, VM Heap: 512 MB, OS: Android 6.0
    - FindLunch-Server
- Rechner 2: CPU: Intel Core i5-2520M (2,5 GHz), RAM: 4 GB (1,3 GHz), OS: Windows 10 Professional
    - Android Virtual Device (AVD) 2: RAM: 768 MB, VM Heap: 64 MB, OS: Android 6.0

Der durchschnittliche Ping auf googleapis.l.google.com beträgt 28ms.

Server und App wurden folgendermaßen modifiziert: Der Server speichert den Zeitstempel beim senden T_S in der GCM-Nachricht. Die App liest diese Information aus und gibt

ihn zusammen mit dem Zeitstempel des Empfangs T_A in ein Log aus. Die Durchlaufzeit R (durch die Netzwerklatenz L und Verarbeitungszeit P) kann für jede Nachricht aus den Logs berechnet werden:

$$R = 2L + P = T_A - T_S$$

Des Weiteren wurde aus Performancegründen die Anzeige einer Benachrichtigung in der App deaktiviert.

Es werden folgende Daten in der Datenbank erzeugt:

- Ein Restaurant an folgender Position: 48.1541, 11.5532
- Für jede der beiden AVDs: 5.000 zufällige Push Notification-Standorte, gleichverteilt im Bereich zwischen (48.1000, 11.5000) und (48.2000 N, 11.6000 E) mit Radius 30000. Das Restaurant befindet sich somit in der Umgebung jeder Push Notification.
- Für jeden Test wird der benötigten Anzahl von Push Notifications über die Tabelle push_notification_has_day_of_week der heutige Wochentag zugewiesen

8.2 Problem der Zeitsynchronisation

Ein Problem im vorhandenen Testaufbau ist die fehlende Möglichkeit der Zeitsynchronisation zwischen dem FindLunch-Server und den AVDs:

1. Eine Android App kann nicht die Systemzeit ändern.
2. Das Android auf der AVD kann sich nur mit der Android Debug Bridge (ADB) synchronisieren.
3. Die ADB kann (mit dem Befehl `adb shell date -s`) nur Sekundengenau synchronisiert werden, was im vorliegen Fall zu ungenau ist. Auch das in Abbildung 9 gezeigte Batch-Skript (prüft die Zehntelsekunden der Systemzeit solange bis sie 00 sind und setzt dann die Zeit der ADB) brachte keine ausreichende Genauigkeit.

Somit ist eine Uhrensynchronisation weder extern (z.B. über das Network Time Protocoll und dem Server ptbtime1.ptb.de) noch intern (z.B. über den Berkeley Algorithmus) möglich. Eine Möglichkeit wäre die Zeitabweichung auf Seiten des FindLunch-Servers zu berechnen (Apps schicken Zeitstempel an zu definierende REST-Schnittstelle), wobei die Netzwerklatenz berücksichtigt werden müsste.

Zur Durchlaufzeit kommt somit die Uhrenabweichung C als weitere Variable hinzu:

$$R = 2L + P + C$$

```
@echo off
C:
cd "C:\Program Files (x86)\Android\android-sdk\platform-tools"

:Start
if %time:~9,2% equ 00 goto SetTime
echo %time:~9,2%
goto Start

:SetTime
adb shell date -s '%date:~-
4%%date:~3,2%%date:~0,2%.%time:~0,2%%time:~3,2%%time:~6,2%'
```

Abbildung 9: Batch-Skript als Versuch zur Uhrensynchronisation mit der ADB

8.3 Vereinfachte Annahmen

Da der Test nur das relative Verhalten des GCM-Dienstes unter Last, nicht aber die absolute Durchlaufzeit untersuchen soll, wird zur Vereinfachung folgende Annahme getroffen: Je AVD wird die geringste gemessene Durchlaufzeit einer Nachricht (in allen Tests) als „Echtzeit-Übertragung“ angenommen. Somit entspricht die angenommene Uhrenabweichung C^{est} bei einer Gesamtmenge von x Nachrichten:

$$C^{est} = \min(\{D_1 \dots D_x\})$$

Die angenommene Abweichung wird von jeder gemessenen Durchlaufzeit abgezogen, wodurch sich die bereinigte Durchlaufzeit R^{net} ergibt:

$$R^{net} = R - C^{est} = 2L + P$$

Es wird weiterhin angenommen, dass die Netzwerklatenz über die Dauer des Tests gleichbleibt. Somit können die Werte von R^{net} aus den unterschiedlichen Tests miteinander verglichen werden.

8.4 Testdurchführung

Für jeden Test wird eine definierte Menge von Nachrichten vom FindLunch-Server an die AVDs gesendet. Jede erzeugte Nachricht hat eine Nutzlast von 180 Byte. Dazu kommen ca. 240 Byte zur GCM-spezifischen Adressierung, für die vom FindLunch-Server gesendeten Nachrichten.

Die cool down-Phase nach jedem Test (Zeit die zwischen senden der letzten Nachricht und beenden des Tests abgewartet wird) beträgt 120 Sekunden.

8.4.1 GCM-Test 1: Steigende Menge von Nachrichten (200-1.000)

Ablauf: Es werden in drei Tests 2*100, 2*250 und 2*500 Nachrichten versendet. Am Server werden hierzu 1.000 Threads bereitgestellt.

Statistiken: In den in der Tabelle 14 dargestellten Statistiken zeigt sich, dass bei diesem Test die Leistung der Clients eine erhebliche Auswirkung auf das Ergebnis hat. Daher werden für die weiteren Tests nur noch die Daten der AVD 1 ausgewertet.

Auswertung: Die Durchlaufzeiten pro Nachricht sind in Abbildung 10 dargestellt. Dabei zeigt sich, dass mit zunehmender Anzahl an Nachrichten die bereinigte Durchlaufzeit linear zunimmt. Eine mögliche Erklärung wäre, dass Google mit steigendem Volumen die Nachrichtenzustellung verlangsamt.

Allerdings liegt der Grund für das beobachtete Verhalten wahrscheinlich im Testaufbau: Auf Serverseite stehen 1.000 Threads zur Verfügung um die Nachrichten zu senden, aber auf den Clients nimmt jeweils nur ein Thread die Nachrichten entgegen.

Tabelle 14: Statistiken des GCM-Test 1

AVD	Nachrichten	Dauer	Verloren
AVD 1	100	8s	0
AVD 2	100	27s	0
AVD 1	250	13s	0
AVD 2	250	103s	0
AVD 1	500	21s	0
AVD 2	500	Absturz der AVD	

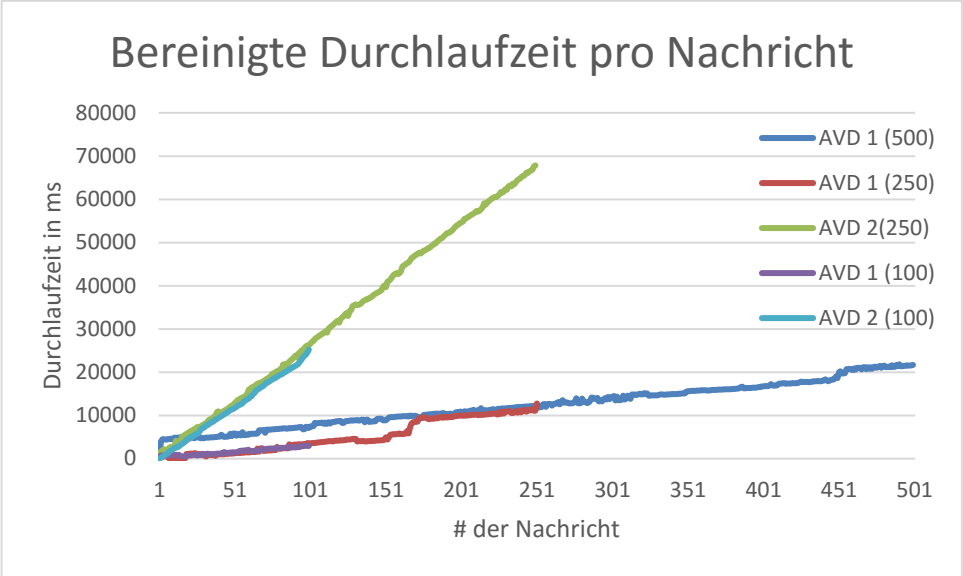

Abbildung 10: Bereinigte Durchlaufzeit pro Nachricht im GCM-Test 1

8.4.2 GCM-Test 2: Wiederholung mit 1.000 Nachrichten weniger Threads

Ablauf: Um das im GCM-Test 1 beschriebene Verhalten zu untersuchen wird ein erneuter Test mit 2*500 Nachrichten pro Client durchgeführt. Allerdings werden hierzu am Server nur zwei Threads zur Verfügung gestellt.

Statistiken: Obwohl die Anzahl der Threads um 96% reduziert wurde, hat sich die Dauer des Tests „nur“ auf 43 Sekunden erhöht – was etwa einer Verdopplung entspricht. Es gingen weiterhin keine Nachrichten verloren.

Auswertung: Wie in Abbildung 11 zu sehen, steigt mit der Anzahl der versendeten Nachrichten nun nicht mehr die bereinigte Durchlaufzeit: Mit Ausnahme einzelner Ausreißer ist sie relativ konstant. Dies zeigt sich auch in der in Abbildung 12 dargestellten Verteilung der bereinigten Durchlaufzeiten: 90% der Nachrichten haben eine bereinigte Durchlaufzeit von weniger als 90ms. Somit ist anzunehmen, dass es sich beim im GCM-Test 1 beobachteten Verhalten um kein Problem des GCM-Dienstes handelt.

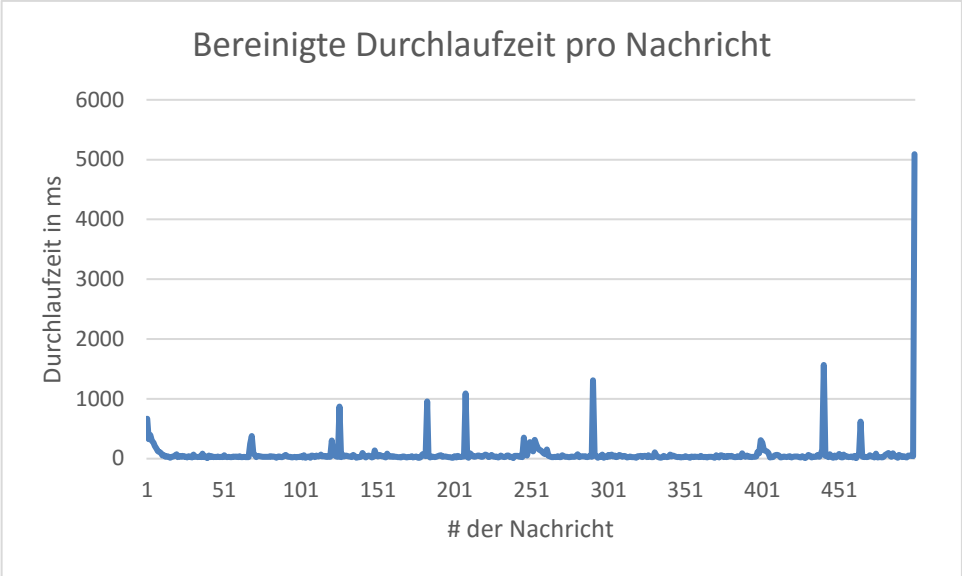

Abbildung 11: Bereinigte Durchlaufzeit pro Nachricht im GCM-Test 2

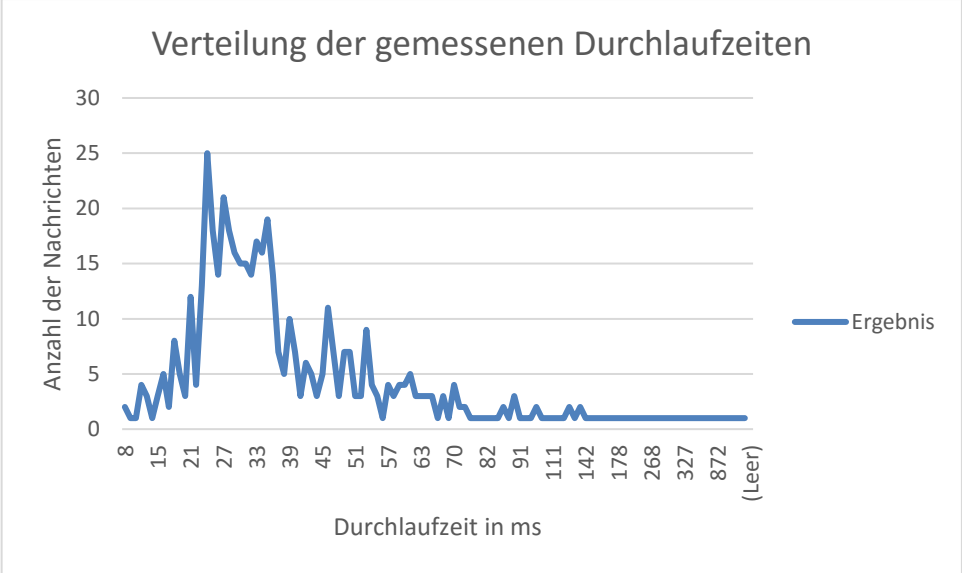

Abbildung 12: Verteilung der gemessenen (bereinigten) Durchlaufzeiten im GCM-Test 2

8.4.3 GCM-Test 3: Hohe Anzahl von Nachrichten (10.000)

Ablauf: Um eine realitätsnahe Last zu erzeugen werden 2*5.000 Nachrichten versendet. Die Anzahl der Threads bleibt bei zwei.

Statistiken: Zusammen mit der Anzahl der Nachrichten hat sich die Dauer des Tests gegenüber dem GCM-Test 2 etwa verzehnfacht (auf 494 Sekunden). Zwei Nachrichten gingen verloren. Hierbei handelt es sich möglicherweise um Ausreißer außerhalb der cool down-Phase.

Auswertung: Wie in Abbildung 13 dargestellt, haben sich zwar Größe und Anzahl der Ausreißer erhöht, aber der Großteil der Nachrichten wird sogar schneller zugestellt: 90% der Nachrichten haben eine bereinigte Durchlaufzeit von weniger als 83ms.

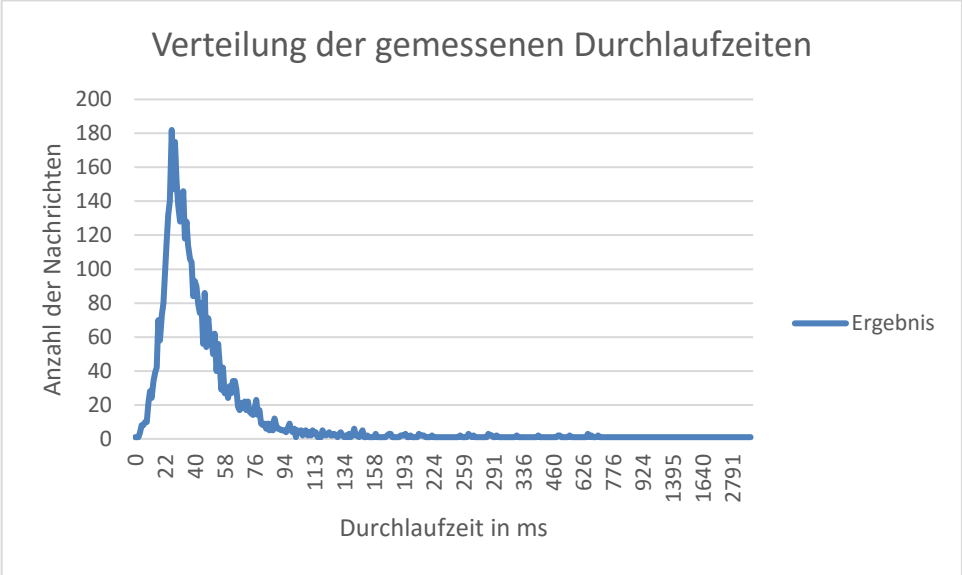

Abbildung 13: Verteilung der gemessenen (bereinigten) Durchlaufzeiten im GCM-Test 3

8.5 Bewertung des Ergebnisses

Die durchgeführten Performance-Tests haben gezeigt, dass der GCM-Dienst unabhängig von der Anzahl der versendeten Nachrichten zuverlässig arbeitet: Google führt keine Drosselung der Übertragungsgeschwindigkeit durch. Auch konnte keine Limitierung der erlaubten Nachrichten, unterhalb von 10.000 Stück, festgestellt werden.

Der GCM-Dienst ist somit für den Einsatz im Rahmen eines kleineren FindLunch-Betriebs mit etwa 7.500 bis 12.500 Benutzer, abhängig von der Nutzung der Push-Funktionalität, problemlos einsetzbar. Für größere Nutzerzahlen sollte ein erneuter Test mit einer deutlich höheren Anzahl von Endgeräten durchgeführt werden.

Tabellenverzeichnis

Abbildungsverzeichnis

Anhang I: JSON zur Konfiguration des Push-Handlings


```
{
  "project_info": {
    "project_number": "343682752512",
    "project_id": "findlunch-1309"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:343682752512:android:e2f9a2460fa10600",
        "android_client_info": {
          "package_name": "edu.hm.cs.projektstudium.findlunch.androidapp"
        }
      },
      "oauth_client": [],
      "api_key": [
        {
          "current_key": "AIzaSyDa7qzHVlNw5c5slr7D_DqAih6eBb3qB0g"
        }
      ],
      "services": {
        "analytics_service": {
          "status": 1
        },
        "appinvite_service": {
          "status": 1,
          "other_platform_oauth_client": []
        },
        "ads_service": {
          "status": 1
        }
      }
    }
  ],
  "configuration_version": "1"
```

Anhang II: Erweiterte Konfiguration des FindLunch-Servers

Die hier beschriebenen Einstellungen erfolgen über die Konfigurationsdatei application.properties

Datenbankzugriff

Während der Entwicklung kam es zu Verbindungsabbrüchen mit der Datenbank. Abgelaufene Verbindungen innerhalb des Connection-Pools konnten als Ursache identifiziert werden. Um dies zu beheben werden Verbindungen nun vor der Entnahme aus dem Pool geprüft und erneuert, falls ungültig. Dies erfolgt über die folgenden Einstellungen:

```
# Maximum number of active connections that can be allocated from this
pool at the same time.
spring.datasource.max-active=50
# Validate the connection before borrowing it from the pool.
spring.datasource.test-on-borrow=true
# Validation query to use
spring.datasource.validation-query=SELECT 1
```

Embedded Tomcat: SSL

Bei Verwendung von 8443 in der Einstellung `server.port` wird SSL automatisch aktiviert. Die Erstellung des Java-Keystores (jks) mit Hilfe von *keytool* und *OpenSSL* wird wie folgt durchgeführt:

1. Erstellen des jks inkl. Schlüssel und Zertifikat:
   `keytool -genkey -alias mydomain -keyalg RSA -keystore keystore.jks -keysize 2048`
2. Löschen des erstellten Schlüssels aus dem jks:
   `keytool -delete -alias mydomain -keystore keystore.jks`
3. Bauen einer p12-Datei aus dem Let's Encrypt Schlüssel und Zertifikat (chain ist aktuell nicht im p12 enthalten):
   `openssl pkcs12 -export -in cert1.pem -inkey privkey1.pem -out server.p12 -name findlunch`
4. Importieren der p12-Datei in den jks:
   `keytool -v -importkeystore -srckeystore server.p12 -srcstoretype PKCS12 -destkeystore keystore.jks -deststoretype JKS`
5. Einbinden des jks in die Anwendung über folgende Einträge in der Datei application.properties:

   ```
   server.ssl.key-store=classpath:keystore.jks
   server.ssl.key-store-password=f1ndLunch_SSL
   ```

```
server.ssl.key-password=f1ndLunch_SSL
```

Zum Deaktivieren von SSL müssen die unter 5.) genannten Zeilen entfernt und folgende Zeile hinzugefügt werden:

```
server.ssl.enabled=false
```

Alternativ hierzu kann der Eintrag `server.port` geändert werden.

Standalone Tomcat

Um FindLunch auf einem Standalone Tomcat (anstatt als executable JAR) zu verwenden, muss die Datei pom.xml so verändert werden, dass ein Web Application Archive (WAR) erzeugt wird. Die hierfür notwendigen Schritte wurden bereits als Kommentare in der pom.xml hinterlegt. Für weitere Information diesbezüglich wird auf die ausführliche Dokumentation von Spring-Boot verwiesen: http://docs.spring.io/spring-boot/docs/current/reference/html/howto-traditional-deployment.html

Upload-Konfiguration

Um den Upload von Bildern mit einer Dateigröße über 1MB zu ermöglichen, wurden folgende Einstellungen angepasst.

```
# Upload configuration
# Attention: Tomcat has its own maximum post filesize. Needs to be
adjusted within Beans.java (in case of embedded tomcat)
multipart.maxFileSize=10MB
multipart.maxRequestSize=20MB
```

Durch die Generierung von Thumbnails nach dem Upload ergeben sich trotz potentiell großer Dateigrößen bei Bildern keine Performance-Problem auf Seiten der Android-App. Gleichzeitig haben Anbieter den Vorteil, Bilder vor dem Upload nicht auf eine vorgegebene Dateigröße skalieren zu müssen.

Logging

Zusätzlich zu den Standard-Logging-Mechanismen von Spring und Hibernate wurde ein eigener Logging-Mechanismus implementiert, der erweiterte Informationen zur Analyse bereitstellt.

Standard-Logging

Die Ausgabemenge der Spring und Hibernate Logeinträgen kann über das Festlegen des Log-Level in den application.properties Einfluss angepasst werden.

```
# Logging configuration
# ROOT level
# logging.level.=DEBUG
logging.file=findLunch.log
logging.level.org.springframework.web=INFO
logging.level.org.hibernate=ERROR
```

Für das erweiterte Logging der Security-Schicht kann folgende Zeile hinzugefügt werden:

```
logging.level.org.springframework.security=TRACE
```

Das Logging der Datenbankzugriffe (inklusive Parameter), kann durch hinzufügen der folgenden Zeilen aktiviert werden:

```
logging.level.org.hibernate.type=TRACE
logging.level.org.hibernate.SQL=DEBUG
```

Erweitertes Logging

Der eigene Logging-Mechanismus dient der Protokollierung weiterer Ereignisse. Hierfür werden folgende Informationen geloggt:

- Datum, Uhrzeit
- Loglevel-Typ (z.B. Info, Error)
- Thread der Ausführung
- Aufgerufene Klasse
- Aufgerufene Methode

Einträge werden bei den nachfolgenden Ereignissen mit den dort angegebenen Informationen geloggt:

1. Aufruf von Webcontrollern
    a. User, URL, HTTP Methode (GET/POST…), Session, Parameter
2. Aufruf von REST-Controllern
    a. User, URL, HTTP Methode (GET/POST…), Session, Parameter
3. Auftretende Exceptions
    a. URL, Exception-Meldung
4. Auftretende Fehler (z.B. bei definierten Fehler-Status-Codes)
    a. URL, Fehlermeldung
5. Fehler in der Validierung beim Aufruf von Webcontrollern
    a. URL, Fehlerhafte Felder mit Meldung zum aufgetretenen Validierungsfehler
6. Ausgaben zum Scheduler (Versand von PushNotifications)
    a. Start / Stop vom Scheduler
    b. Versand von Messages

Sämtliche Log-Einträge werden in die Datei findLunch.log geschrieben.

Anhang III: Erzeugung der FindLunch-Datenbank

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema findlunch
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `findlunch` ;

-- -----------------------------------------------------
-- Schema findlunch
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `findlunch` DEFAULT CHARACTER SET utf8 ;
USE `findlunch` ;

-- -----------------------------------------------------
-- Table `findlunch`.`country`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`country` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`country` (
  `country_code` VARCHAR(2) NOT NULL,
  `name` VARCHAR(30) NOT NULL,
  PRIMARY KEY (`country_code`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`restaurant_type`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`restaurant_type` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`restaurant_type` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(30) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`restaurant`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`restaurant` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`restaurant` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(60) NOT NULL,
  `street` VARCHAR(60) NOT NULL,
  `street_number` VARCHAR(11) NOT NULL,
  `zip` VARCHAR(5) NOT NULL,
```

```
  `city` VARCHAR(60) NOT NULL,
  `country_code` VARCHAR(2) NOT NULL,
  `location_latitude` FLOAT NOT NULL,
  `location_longitude` FLOAT NOT NULL,
  `email` VARCHAR(60) NOT NULL,
  `phone` VARCHAR(60) NOT NULL,
  `url` VARCHAR(60) NULL,
  `restaurant_type_id` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_restaurant_countries1_idx` (`country_code` ASC),
  INDEX `fk_restaurant_restaurant_type1_idx` (`restaurant_type_id` ASC),
  CONSTRAINT `fk_restaurant_countries1`
    FOREIGN KEY (`country_code`)
    REFERENCES `findlunch`.`country` (`country_code`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_restaurant_restaurant_type1`
    FOREIGN KEY (`restaurant_type_id`)
    REFERENCES `findlunch`.`restaurant_type` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`day_of_week`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`day_of_week` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`day_of_week` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(30) NOT NULL,
  `day_number` INT NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `day_number_UNIQUE` (`day_number` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`time_schedule`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`time_schedule` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`time_schedule` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `restaurant_id` INT NOT NULL,
  `offer_start_time` DATETIME NULL,
  `offer_end_time` DATETIME NULL,
  `day_of_week_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_time_schedule_restaurant1_idx` (`restaurant_id` ASC),
  INDEX `fk_time_schedule_day_of_week1_idx` (`day_of_week_id` ASC),
```

```
  CONSTRAINT `fk_time_schedule_restaurant1`
    FOREIGN KEY (`restaurant_id`)
    REFERENCES `findlunch`.`restaurant` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_time_schedule_day_of_week1`
    FOREIGN KEY (`day_of_week_id`)
    REFERENCES `findlunch`.`day_of_week` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`opening_time`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`opening_time` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`opening_time` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `opening_time` DATETIME NOT NULL,
  `closing_time` DATETIME NOT NULL,
  `time_schedule_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_opening_time_time_schedule1_idx` (`time_schedule_id` ASC),
  CONSTRAINT `fk_opening_time_time_schedule1`
    FOREIGN KEY (`time_schedule_id`)
    REFERENCES `findlunch`.`time_schedule` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`offer`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`offer` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`offer` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `restaurant_id` INT NOT NULL,
  `title` VARCHAR(60) NOT NULL,
  `description` TINYTEXT NOT NULL,
  `price` DECIMAL(5,2) NOT NULL,
  `preparation_time` INT NOT NULL,
  `start_date` DATE NULL,
  `end_date` DATE NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_product_restaurant1_idx` (`restaurant_id` ASC),
  CONSTRAINT `fk_product_restaurant1`
    FOREIGN KEY (`restaurant_id`)
    REFERENCES `findlunch`.`restaurant` (`id`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`kitchen_type`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`kitchen_type` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`kitchen_type` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(30) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`offer_photo`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`offer_photo` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`offer_photo` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `offer_id` INT NOT NULL,
  `photo` MEDIUMBLOB NOT NULL,
  `thumbnail` MEDIUMBLOB NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_offer_photo_offer1_idx` (`offer_id` ASC),
  CONSTRAINT `fk_offer_photo_offer1`
    FOREIGN KEY (`offer_id`)
    REFERENCES `findlunch`.`offer` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`user_type`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`user_type` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`user_type` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(30) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`user`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `findlunch`.`user` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`user` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `username` VARCHAR(60) NOT NULL,
  `password` VARCHAR(255) NOT NULL,
  `restaurant_id` INT NULL,
  `user_type_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_user_restaurant1_idx` (`restaurant_id` ASC),
  UNIQUE INDEX `username_UNIQUE` (`username` ASC),
  INDEX `fk_user_user_type1_idx` (`user_type_id` ASC),
  CONSTRAINT `fk_user_restaurant1`
    FOREIGN KEY (`restaurant_id`)
    REFERENCES `findlunch`.`restaurant` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_user_user_type1`
    FOREIGN KEY (`user_type_id`)
    REFERENCES `findlunch`.`user_type` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`restaurant_has_kitchen_type`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`restaurant_has_kitchen_type` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`restaurant_has_kitchen_type` (
  `restaurant_id` INT NOT NULL,
  `kitchen_type_id` INT NOT NULL,
  PRIMARY KEY (`restaurant_id`, `kitchen_type_id`),
  INDEX `fk_restaurant_has_kitchen_type_kitchen_type1_idx` (`kitchen_type_id`
ASC),
  INDEX `fk_restaurant_has_kitchen_type_restaurant1_idx` (`restaurant_id`
ASC),
  CONSTRAINT `fk_restaurant_has_kitchen_type_restaurant1`
    FOREIGN KEY (`restaurant_id`)
    REFERENCES `findlunch`.`restaurant` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_restaurant_has_kitchen_type_kitchen_type1`
    FOREIGN KEY (`kitchen_type_id`)
    REFERENCES `findlunch`.`kitchen_type` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
```

```
-- Table `findlunch`.`favorites`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`favorites` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`favorites` (
  `user_id` INT NOT NULL,
  `restaurant_id` INT NOT NULL,
  PRIMARY KEY (`user_id`, `restaurant_id`),
  INDEX `fk_user_has_restaurant_restaurant1_idx` (`restaurant_id` ASC),
  INDEX `fk_user_has_restaurant_user1_idx` (`user_id` ASC),
  CONSTRAINT `fk_user_has_restaurant_user1`
    FOREIGN KEY (`user_id`)
    REFERENCES `findlunch`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_user_has_restaurant_restaurant1`
    FOREIGN KEY (`restaurant_id`)
    REFERENCES `findlunch`.`restaurant` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`offer_has_day_of_week`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`offer_has_day_of_week` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`offer_has_day_of_week` (
  `offer_id` INT NOT NULL,
  `day_of_week_id` INT NOT NULL,
  PRIMARY KEY (`offer_id`, `day_of_week_id`),
  INDEX `fk_offer_has_day_of_week_day_of_week1_idx` (`day_of_week_id` ASC),
  INDEX `fk_offer_has_day_of_week_offer1_idx` (`offer_id` ASC),
  CONSTRAINT `fk_offer_has_day_of_week_offer1`
    FOREIGN KEY (`offer_id`)
    REFERENCES `findlunch`.`offer` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_offer_has_day_of_week_day_of_week1`
    FOREIGN KEY (`day_of_week_id`)
    REFERENCES `findlunch`.`day_of_week` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`push_notification`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`push_notification` ;
```

```
CREATE TABLE IF NOT EXISTS `findlunch`.`push_notification` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `user_id` INT NOT NULL,
  `title` VARCHAR(60) NULL,
  `latitude` FLOAT NOT NULL,
  `longitude` FLOAT NOT NULL,
  `radius` INT NOT NULL,
  `gcm_token` TEXT(4096) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_push_notification_user1_idx` (`user_id` ASC),
  CONSTRAINT `fk_push_notification_user1`
    FOREIGN KEY (`user_id`)
    REFERENCES `findlunch`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`push_notification_has_day_of_week`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`push_notification_has_day_of_week` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`push_notification_has_day_of_week` (
  `push_notification_id` INT NOT NULL,
  `day_of_week_id` INT NOT NULL,
  PRIMARY KEY (`push_notification_id`, `day_of_week_id`),
  INDEX `fk_push_notification_has_day_of_week_day_of_week1_idx`
(`day_of_week_id` ASC),
  INDEX `fk_push_notification_has_day_of_week_push_notification1_idx`
(`push_notification_id` ASC),
  CONSTRAINT `fk_push_notification_has_day_of_week_push_notification1`
    FOREIGN KEY (`push_notification_id`)
    REFERENCES `findlunch`.`push_notification` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_push_notification_has_day_of_week_day_of_week1`
    FOREIGN KEY (`day_of_week_id`)
    REFERENCES `findlunch`.`day_of_week` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `findlunch`.`push_notification_has_kitchen_type`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `findlunch`.`push_notification_has_kitchen_type` ;

CREATE TABLE IF NOT EXISTS `findlunch`.`push_notification_has_kitchen_type` (
  `push_notification_id` INT NOT NULL,
  `kitchen_type_id` INT NOT NULL,
```

```
  PRIMARY KEY (`push_notification_id`, `kitchen_type_id`),
  INDEX `fk_push_notification_has_kitchen_type_kitchen_type1_idx`
(`kitchen_type_id` ASC),
  INDEX `fk_push_notification_has_kitchen_type_push_notification1_idx`
(`push_notification_id` ASC),
  CONSTRAINT `fk_push_notification_has_kitchen_type_push_notification1`
    FOREIGN KEY (`push_notification_id`)
    REFERENCES `findlunch`.`push_notification` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_push_notification_has_kitchen_type_kitchen_type1`
    FOREIGN KEY (`kitchen_type_id`)
    REFERENCES `findlunch`.`kitchen_type` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```