

# Konzeption und Prototyp eines Verwaltungstool für Buchungen von Wohnräumen

Analyse der verteilten Anwendung

Eine Arbeit von  
Stefan Baumann, Florian Gumbel  
entstanden im Sommersemester 2016 im Fach  
**„Entwicklung verteilter Anwendung“ bei**  
**Prof. Dr. Karim Kremer**  
(16.09.2016)



Technische Hochschule Mittelhessen  
Wilhelm-Leuschner-Straße 13  
61169 Friedberg



## **Zusammenfassung**

Ein Dienstleister welcher in einem Feriengebiet seine Inserate verwalten will, bietet einen Buchungsservice per Telefon an. In seinen Call-Centern sitzen Mitarbeiter, welcher mithilfe der Software Buchungen aufnehmen und verwalten können. Dabei soll bei allen Anwendern die Synchronität der Daten sichergestellt werden, ohne, dass diese kontinuierlich manuell aktualisieren müssen. Eine Kalender-Ansicht hilft bei der Verwaltung von Buchungsobjekten und den gebuchten Zeiträumen.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Technologie</b>	<b>3</b>
2.1	Verwendete Technologien	3
2.1.1	AngularJS	3
2.1.2	Grunt	3
2.1.3	NPM	3
2.1.4	Bower	4
2.1.5	Sass	4
2.1.6	Yeoman	4
2.2	Verwendete Pakete	4
2.2.1	Firebase	4
2.2.2	AngularJS Material Design	5
2.2.3	Compass	5
2.2.4	Fullcalendar	5
<b>3</b>	<b>Projektvorbereitung</b>	<b>7</b>
3.1	Zielsetzung	7
3.2	Umsetzung	8
3.2.1	Frontend	8
3.2.2	Funktionen	8
3.2.3	Backend	9
<b>4</b>	<b>Projektstruktur</b>	<b>11</b>
4.1	Projekt anlegen	11
4.2	Grunt Tasks	12
4.2.1	serve	13
4.2.2	build	13
<b>5</b>	<b>Frontend</b>	<b>15</b>
5.1	Login	15
5.2	Hauptseite	16
5.2.1	Header	16
5.2.2	Main	18
5.3	Kunde	18

5.4	Objekt .....	21
5.5	Buchung .....	22
5.6	Kalender .....	24
5.6.1	Drag'n Drop .....	24
5.6.2	Resizing .....	24
5.6.3	Selektion .....	25
<b>6</b>	<b>Backend .....</b>	<b>27</b>
6.1	Echtzeit-Datenbank .....	27
6.2	Organisation .....	27
6.3	Zugriffsverwaltung .....	28
6.4	Lesen, Schreiben, Echtzeit-Synchronisation .....	29
6.5	Authentifizierung .....	30
6.6	Weitere Firebase-Funktionen .....	31
6.6.1	Technische Umsetzung der Datenbank-Anbindung .....	31
6.6.2	Collections .....	33
<b>7</b>	<b>Fazit .....</b>	<b>37</b>

# 1 Einleitung

Für die Vermietung seiner Ferienwohnungen und Häuser benötigt ein Unternehmen eine Software zur Verwaltung der jeweiligen Buchungen. Da das Unternehmen über mehrere Buchungsagenten verfügt, ist eine Anwendung nötig, die das parallele Arbeiten an verschiedenen Geräten ermöglicht. Dazu gehören nicht nur Computer und Laptops, sondern auch Tablets, um schnell von unterwegs eine Buchung einzutragen. Besonders wichtig ist dem Unternehmen die schnelle Aktualisierung der Daten in der Software, um eventuelle Doppelbuchungen zu vermeiden und das gleichzeitige Arbeiten zu ermöglichen.



## 2 Technologie

Dieses Kapitel bietet einen Überblick über die im Projekt verwendeten Technologien, Frameworks und Pakete.

### 2.1 Verwendete Technologien

#### 2.1.1 AngularJS

AngularJS ist ein von Google entwickeltes JavaScript-Framework für client-seitige Webanwendungen, welches nach dem Model-View-View-Model Prinzip funktioniert. Es eignet sich besonders gut für Single-Page-Applications, bei der die meisten Daten bereits beim ersten Aufruf geladen werden. Das führt dazu, dass bei einer Änderung der URL nicht mehr die komplette Seite aktualisiert wird sondern lediglich die benötigten Daten per Ajax nachgeladen werden. Dadurch, dass bei AngularJS alles mittels JavaScript gerendert wird, stellt die Suchmaschinenoptimierung einen zusätzlichen Aufwand da, da die Suchmaschinen aktuell noch ihre Probleme haben. Abgesehen davon verfügt AngularJS aber über viele Stärken. Dazu gehören unter anderem Two-Way Binding, sehr gute Testbarkeit, Abstraktion von Low-Level-Operationen so wie die Lesbarkeit und Erweiterung von HTML-Code.

#### 2.1.2 Grunt

Grunt ist ein sogenannter JavaScript-Taskrunner der dazu da ist um wiederkehrende Aufgaben bei Build-Prozessen in Frontend-Projekten zu automatisieren. Sobald er einmal konfiguriert ist, ist das Testen und Ausliefern selbst bei umfangreichen Projekten problemlos möglich. Es hilft viele Schritte wie zum Beispiel das minifizieren von JavaScript-Code oder das Umwandeln von SASS-Code zu CSS-Code von zentraler Stelle aus zu steuern.

#### 2.1.3 NPM

Node Package Manager ist ein Kommandozeilenprogramm für node.js. Es erleichtert JavaScript Entwicklern das Teilen von Code, welcher erstellt wurde um besondere Probleme zu lösen. Diese wiederverwendbaren Codeschnipsel werden Package oder manchmal auch Module genannt. Die Idee dahinter ist,



dass ein Package immer nur ein Problem richtig löst. Das ermöglicht es, mit Hilfe von vielen kleinen Package, zu einer Lösung zu gelangen.

### 2.1.4 Bower

Wie auch der node package manager ist Bower ein Kommandozeilen Paket-verwaltungstool für die clientseitige Webentwicklung. Er ist sogar in Node.js geschrieben und wird über NPM installiert. Er dient zur Installation und Aktualisierung von Programmbibliotheken und Frameworks. Als Ergänzung zu NPM und in Zusammenarbeit mit Grunt kann der Workflow erheblich beschleunigt und verbessert werden.

### 2.1.5 Sass

Sass ist ein ausgereifter, etablierter und leistungsfähiger CSS-Präprozessor. Mit der Dateiendung .scss kann die Erweiterung genutzt werden. Da die Browser aktuell keine .scss-Dateien unterstützen, muss das Sass-Kommandozeilentool den Code zunächst in .css-Dateien übersetzen. Andersherum ist es ohne Probleme möglich CSS-Code in SASS-Dateien einzubinden und zu nutzen, da dieser korrekt umgewandelt wird. Zu den Vorteilen von Sass gehört zum Beispiel Verschachtelungen, Variablen, Mixins, Vererbung und Importe.

### 2.1.6 Yeoman

Bei der modernen Frontendentwicklungen werden viele Bibliotheken und Werkzeuge benötigt. Yeoman bietet einen Workflow um Pakete zugänglich zu machen und zu installieren. Es ist ein Meta-Paketemanager, Entwicklungsserver, Code-Generator, der ein Grundgerüst mit den ausgewählten Bibliotheken und Frameworks bereitstellt. Es erstellt die Projektstruktur und lädt die notwendigen Ressourcen herunter. Je nach eingesetztem Generator stehen weitere verschiedene zusätzliche Funktionen zur Verfügung.

## 2.2 Verwendete Pakete

### 2.2.1 Firebase

Wie auch AngularJS stammt das Framework Firebase aus dem Hause Google. Allerdings wurde die Plattform nicht von Grund auf von Google entwickelt sondern erst im Jahr 2014 übernommen. Es stellt eine universelle App-Plattform für Entwickler und Marketer zur Verfügung, welches zur Entwicklung von hochqualifizierten Anwendungen dient. Das Herzstück des Framework ist die Analyse von Apps und mobilen Anwendungen. Des Weiteren bietet es Cloud-Speicher, Cloud-Messaging, Remote Config und Test Lab. Ebenfalls stellt es die Möglichkeit zur Authentifizierung bereit so wie eine Echtzeit NoSQL Cloud Datenbank.

### 2.2.2 AngularJS Material Design

Passend zu Angular gibt es das User Interface Component Framework AngularJS Material Design. Es ist ein Zusammenschluss der Material Design Guidelines und dem AngularJS Framework. Dies soll dabei helfen, zeitgemäße attraktive konsistente und funktionale Webseitendesigns zu Erstellen und auf allen Endgeräten zu gewährleisten.

### 2.2.3 Compass

Zusammen mit der Spracherweiterung Sass baut Compass ein Framework. Es ist quasi die Standardbibliothek für Sass und bietet vieles was auch gängige CSS-Frameworks enthalten. Compass erlaubt eine einheitliche Schreibweise die in verschiedene Eigenschaften übersetzt wird. Das hat zur Folge, dass sich die Entwickler keine Gedanken machen müssen, ob alle Browser abgedeckt werden. Ebenfalls ermöglicht es das automatisierte Erstellen von CSS-Sprites.

### 2.2.4 Fullcalendar

Fullcalendar ist ein jQuery-Plugin und stellt ein umfangreiches Kalender-User-Interface mit vielen Funktionen zur Verfügung. Zu dem Funktionsumfang gehören verschiedene Kalenderansichten wie beispielsweise Monats-, Wochen- und Tages-Views. Zwischen diesen lässt sich bei Bedarf dynamisch umschalten. Der Funktionsumfang wird ergänzt durch eine für Entwickler gedachte Schnittstelle, über welche man auf verschiedene User-Events innerhalb des Kalenders reagieren kann. Dadurch erhält der Entwickler die volle Kontrolle über die Steuerung der Events.

Zu den Kalender-Funktionen bietet Fullcalendar eine Erweiterung für den Kalender zur Verfügung, welche eine Timeline-Ansicht mit sich bringt die es ermöglicht eine Ansicht der Events auf horizontaler Achse.



## 3 Projektvorbereitung

Im folgenden Kapitel werden Ziele definiert, welche sich aus den Problemen in der Einleitung ergeben haben. Darauf folgt die Umsetzung der beschriebenen Ziele.

### 3.1 Zielsetzung

Das Ziel ist eine prototypische Entwicklung einer Software zur Verwaltung von Buchungen für Ferienhäuser und Wohnungen. Eine der Anforderungen an die Software ist die Nutzung auf verschiedenen Endgeräten. Das hat zur Folge, dass zu dem auch die Betriebssysteme variieren können. Daraus ergibt sich, dass die Anwendung plattformunabhängig nutzbar sein muss. Des Weiteren muss es möglich sein, dass mehrere Nutzer gleichzeitig die Software bedienen können ohne sich gegenseitig zu stören. Außerdem soll die Software sowohl im Büro als auch vor Ort nutzbar sein. Des Weiteren benötigt das Programm eine Anmeldung mit Authentifizierung. Eine Buchung kommt dann zustande wenn ein bestimmter Kunde ein bestimmtes Objekt für einen bestimmten Zeitraum mieten möchte. Daraus ergibt sich, dass Es eine Verwaltung für Kunden, Objekte und Buchungen selbst geben muss. Das Ganze soll mit einer einfachen und leicht bedienbaren Benutzeroberfläche ausgestattet sein.

## 3.2 Umsetzung

Für die oben genannten Ziele eignet sich eine Webanwendung besonders gut. Sie ist von jedem Gerät aus nutzbar und benötigt in der Regel nicht mehr als einen aktuellen Internetbrowser. Auch die Installation eines extra Programmes erübrigt sich damit. Der Nachteil an einer Webanwendung ist, dass eine stetige Internetverbindung Voraussetzung ist. In Anbetracht der weiteren Bedingungen an die Software wäre diese aber auch bei einer nativ installierten Anwendung Voraussetzung, da sie für den stetigen Abgleich der Daten zwingend vorhanden sein muss. Das Ziel ist es also eine Webanwendung mit einer Benutzeroberfläche zu erstellen, die auf jedem Gerät angezeigt werden kann sofern es über Internet und einen entsprechenden Browser verfügt. Des weiteren wird ein entsprechendes Backend benötigt, dass die Daten für die Authentifizierung, der Kunden, der Objekte und der Buchungen speichert. Weitere Anforderungen an die Anwendungen selbst ist die gleichzeitige Verwaltung von Kunden, der zu vermietbaren Ferienhäuser und Wohnungen sowie die eigentliche Buchung. Das ganze soll übersichtlich dargestellt werden und einfach, intuitiv und schnell benutzbar sein. Für die Umsetzung der eigentlichen Webanwendung kommen JavaScript, HTML und CSS zum Einsatz. Zu dem werden die in Kapitel 2 genannten Frameworks und Tools zur Unterstützung eingesetzt.

### 3.2.1 Frontend

Neben den oben genannten Script- und Auszeichnungssprachen soll für die Logik und Strukturierung der Anwendung das Framework AngularJS genutzt werden. Dazu passend soll das Layout mittels der Angular Version von Material gestaltet werden.

### 3.2.2 Funktionen

Die Funktionen sollen sinnvoll in die Webanwendung integriert werden. Dabei sollen sie schnell und ohne Umwege erreichbar sein. Zu den Hauptfunktionen zählt das Hinzufügen und Bearbeiten von Kundendaten, Ferienwohnungen und der eigentlichen Buchung selbst.

Das Formular zum Hinzufügen von Kunden und Wohnungen soll folgende Informationen beinhalten:

- Firma (Falls vorhanden)
- Vor- und Nachname
- Strasse und Hausnummer
- Postleitzahl und Stadt
- Telefonnummer
- Geburtstag

- Feld für Zusatzinformationen

Für das Bearbeiten oder Löschen eines Kunden soll es möglich sein, diesen über eine Suche zu ermitteln und die Daten anschließend im Formular anzeigen zu lassen.

Das Formular zum Hinzufügen von Ferienhäuser und Wohnungen soll folgende Informationen beinhalten:

- Name des Objektes
- Anzahl der Personen die es maximal beherbergen kann

Für das Formular zum Bearbeiten eines Objekts soll dieses in der Liste ausgewählt werden. Die Daten werden anschließend angezeigt und können bearbeitet oder gelöscht werden.

Das Formular zum Hinzufügen von Buchungen soll folgende Informationen beinhalten:

- Kunde, der das Objekt bucht
- Objekt, das gebucht wird
- Anzahl der Personen
- Startdatum
- Enddatum

Ein wichtiger Bestandteil dieser Software ist die Darstellung der aktuellen Buchungen. Sie soll sowohl übersichtlich sein als auch intuitiv zu bedienen sein. Eine Kalenderansicht mit der Auflistung aller Objekte und dessen Buchungen soll dem Nutzer einen schnellen Überblick verschaffen. Dazu soll das in 2 vorgestellte `fullcalendar`-Framework zum Einsatz kommen. Dieses bietet die Möglichkeit die Buchungen in einer Kalenderansicht darzustellen. Für eine gute Übersicht soll das ganze in der Monatsansicht und die Buchungen in Blöcken angezeigt werden. Wird auf ein Block geklickt soll sich automatisch die Buchung mit der Möglichkeit zur Änderung und Löschung öffnen.

### 3.2.3 Backend

Für das Speichern der Daten soll die in Firebase integrierte Objektbasierte Datenbank zum Einsatz kommen. Ebenfalls übernimmt ein weiterer Dienst die Authentifizierung bei der Anmeldung.



## 4 Projektstruktur

Im diesem Kapitel wird die grundlegende Projektstruktur und das Anlegen des Projektes beschrieben.

### 4.1 Projekt anlegen

Zu Beginn der Entwicklung war es nötig, eine Basis für die Anwendung zu schaffen. Zu dieser Basis gehörten

- **GitHub Repository** zur Versionsverwaltung des Codes
- **Konfigurations-Einstellungen in package.json** zur Verwaltung der von NodeJS benötigten Pakete
- **Konfigurations-Einstellungen in bower.json** zur Verwaltung der Pakete des FrontEnds
- **Grundfile.js** mit allen Tasks, die während der Entwicklung und zum Deployment zum Einsatz kommen
- **Ordner-Struktur** für Module und die eigentliche Anwendung

Dies per Hand zu machen, ist generell sehr fehleranfällig und benötigt einige Zeit. Aus diesen Gründen wurde hierbei die NodeJS-Kickstarter-Anwendung *yeoman.io* verwendet. Diese Anwendung stellt verschiedene Generatoren zur Verfügung, mit welchen sich unterschiedlichste Anwendungen kickstarten lassen. Aktuell umfasst die Generatoren-Bibliothek mehr als 4800 verschiedene Generator-Module.

Die vorliegende Anwendung wurde mithilfe des Generators “angular,, gekickstartet. “angular,, ist ein offizielles vom Yeoman-Team entwickeltes Modul. Über den Konsolenbefehl

```
yo generator:angular App
```

generiert Yeoman alle oben genannten Strukturen, Dateien der Boilerplate-Angular-Anwendung und Test-Methoden. Da es sich bei AngularJS-Anwendungen meistens um Single-Page-Anwendungen handelt, ist die Ausgangsdatei die *index.html*-Datei im App-Ordner. In dieser Datei werden alle JavaScript-Module und CSS-Bibliotheken zusammengeführt. Dabei zeichnet sich ein großer Vorteil von der Verwendung von Yeoman heraus: In der HTML-Datei befinden sich Marker, mit welchen die Bereiche markiert sind, in denen JavaScripte und



Stylesheets eingebaut werden. Installiert ein Benutzer über Bower neue Module, so erfolgt das Einbinden automatisch. Gesteuert wird dieser Prozess von dem `serve`-Task in *Gruntfile.js*.

Eine weitere sehr nützliche Funktion, ist die Möglichkeit, neue AngularJS-Module, wie z.B. Controller, Services oder Direktiven direkt über die Kommandozeile hinzuzufügen. Die Erstellung des Kontrollers erfolgt über den Befehl `yo angular:controller ControllerName`

Über diesen Befehl generiert Yeoman nun einen Angular-Controller, der in dem entsprechenden controller-Ordner abgelegt wird. Dieser Controller ist bereits mit der existierenden Anwendung verknüpft. Zudem wird das JS-File direkt in die *index.html*-Datei eingebunden.

Zusammengefasst erleichtert es die Verwendung eines Generators, wie in diesem Falle Yeoman, Grundlagen für eine Anwendung zu schaffen und diese in ihrer Entwicklung voranzutreiben.

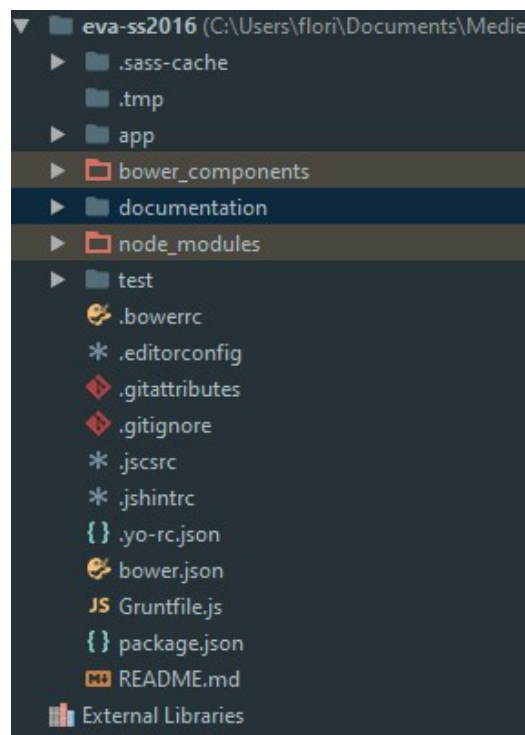


Abbildung 4.1  
Angelegte Projektstruktur

## 4.2 Grunt Tasks

Wie bereits im vorausgegangenen Abschnitt erwähnt, generiert Yeoman eine JavaScript-Datei namens *Gruntfile.js*. Diese beinhaltet vorgefertigte Grunt-Tasks zum Entwickeln, Test und Deployen/Builden der Anwendung. Nachfolgend wird auf die beiden wichtigsten tasks eingegangen.

### 4.2.1 serve

Der *serve*-Task ist der Task, der während der Entwicklung zum Einsatz kommt. Über ihn wird auf *http://localhost* ein lokaler Server gestartet, auf welchem die zu entwickelnde Anwendung erreichbar ist. Im Hintergrund wird zudem zwischen Grunt und dem Browser eine Socket-Verbindung zum Livereload aufgebaut. Der Zweck dieser Verbindung ist, dass sobald der Entwickler in der Entwicklungsumgebung eine Änderung an Projekt-Dateien wie z.B. an JavaScript- oder Sass-Dateien macht, automatisch ein Refresh des Browserfensters angestoßen wird. Darüber erspart sich der Entwickler den zusätzlichen Tastendruck beim Ansehen/Testen der Seite.

Damit das Erkennen von Änderungen möglich ist, startet der *serve*-Task Datei-Watcher, welche auf beliebige Dateien angesetzt werden können. Diese reagieren auf Änderungen und führen mit dem Datei-Typ verknüpfte Subtasks (z.B. Kompilieren von Sass-Dateien zu CSS-Dateien) aus.

### 4.2.2 build

Bei diesem Task handelt es sich um den Build-Task der Anwendung. Seine Aufgabe ist es, aus dem entwickelten Code eine fertige Anwendung zu generieren. Dazu gehört das Kompilieren von Sass zu CSS, Zusammenführen von CSS- und JavaScript-Dateien, Minimieren von JavaScript (dazu kann noch ein „Uglyfier“ hinzugeschaltet werden, der neben der Minimierung des Javascript-Codes diesen auch unleserlich macht), Zusammenfassen der HTML-Templates, Kopieren aller benötigten Ressourcen wie Grafiken und Fonts und letztendlich Generieren eines *dist*-Ordners, der die gesamte Anwendung enthält.

Um die Anwendung nun zu publizieren, ist nur noch der Inhalt des *dist*-Ordners nötig.



## 5 Frontend

In diesem Kapitel wird die prototypische Entwicklung des Frontend beschrieben. Dabei werden anhand der verschiedenen Bereiche die Umsetzung der Projektstruktur und die Funktionsweise des Programmes erläutert.

### 5.1 Login

Da die Software, wie in den Projektzielen erwähnt, nicht nur intern im Büro sondern auch von extern aus erreichbar sein muss, ist es nötig eine Authentifizierung einzurichten damit unbefugte keine Möglichkeiten haben die Daten zu verändern oder einzusehen. Um die Software nutzen zu können, benötigt der Bearbeiter eine gültige Kombination aus E-Mail Adresse und Passwort. Schon bei der Eingabe der Daten wird überprüft, ob die Felder leer sind oder es sich überhaupt um eine E-Mail Adresse handelt. Dabei wird abgefragt, ob ein @-Zeichen vorhanden ist und ob darauf ein Punkt folgt. Sobald das Formular abgeschickt wurde, werden die Daten zunächst im `login.js` Controller entgegengenommen. Von da aus werden diese an den Authenticate-Service weitergegeben wo sie an das Backend weitergereicht werden. In Firebase werden sie dann mit den hinterlegten Daten verglichen. Sind E-Mail Adresse und Passwort nicht korrekt oder nicht vorhanden, wird eine Fehlermeldung zurückgegeben die der Controller im View anzeigt. Sofern alle Angaben korrekt sind, wird der Nutzer weiter zu der Hauptseite geleitet. Da es sich hierbei um eine Betriebsinterne Software handelt, wird eine Registrierung nicht benötigt. Die E-Mail Adresse und das Passwort werden Manuell im Backend hinzugefügt, bearbeitet oder entfernt. Die Folgende Abbildung 5.1 zeigt, wie der Anmeldebildschirm aussieht.

The login form consists of a blue header bar with the text 'Login'. Below it are two white input fields: 'E-Mail Adresse' and 'Passwort'. At the bottom is a blue button with the text 'ANMELDEN'.

Abbildung 5.1  
Anmeldeformular

## 5.2 Hauptseite

Aus den in den Projektzielen gesteckten Bedingungen ergibt sich die in Abbildung 5.2 gezeigte Hauptseite. Diese ist schlicht aufgebaut bietet jedoch gleichzeitig alle gewünschten Funktionen, die mit einem Klick erreicht werden können.

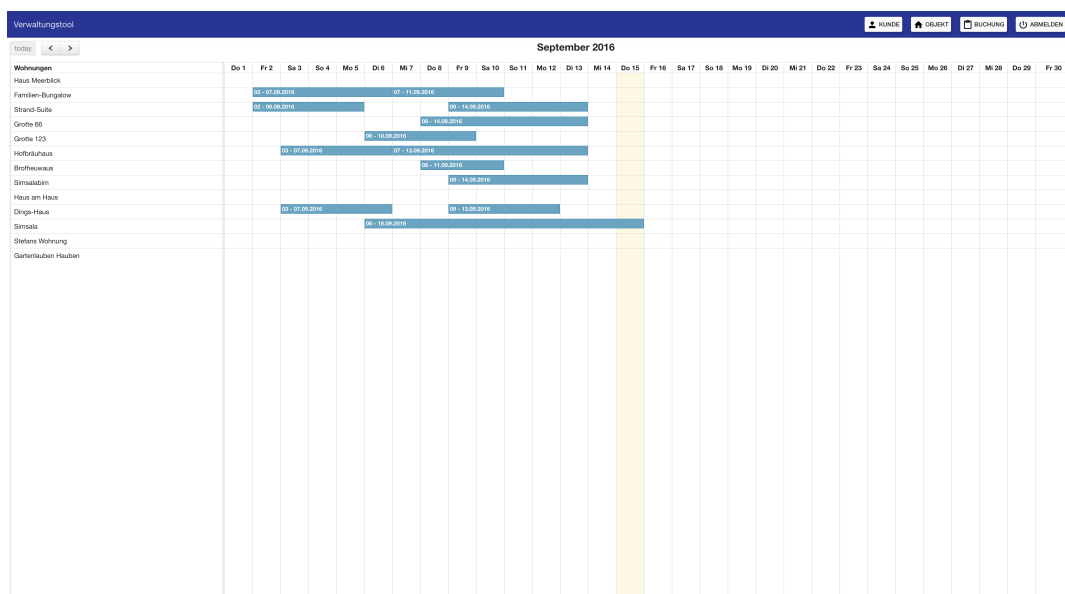


Abbildung 5.2  
Hauptseite

### 5.2.1 Header

Im Kopfbereich befinden sich neben dem Programmname auch folgende vier Button:

#### Kunde

Ein Dialog öffnet sich in dem Kunden hinzugefügt werden können. Nach

einer Suche ist das Bearbeiten und Löschen ebenfalls möglich.

### Objekt

Ein Dialog öffnet sich in dem eine neue Ferienwohnung / Haus hinzugefügt werden kann.

### Buchung

Ein Dialog öffnet sich in dem eine neue Buchung hinzugefügt werden kann.

### Logout

Nach der Bestätigung eines Dialogs wird der Nutzer abgemeldet und auf die Login-Seite verwiesen.

Sobald die Fenstergröße kleiner ist oder die Größe eines durchschnittlichen Tablets erreicht wurde, werden die Buttons im Kopfbereich ausgeblendet (Abbildung 5.3). Stattdessen erscheint ein Menüsymbol das bei Klick eine Seitennavigation einblendet in der sich alle Buttons befinden (Abbildung 5.4).

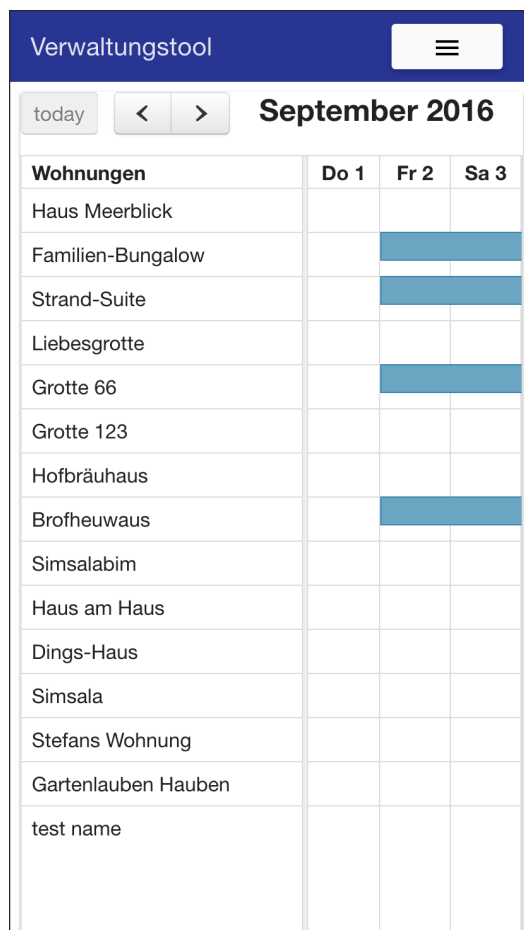


Abbildung 5.3  
Ansicht auf Mobilgeräten

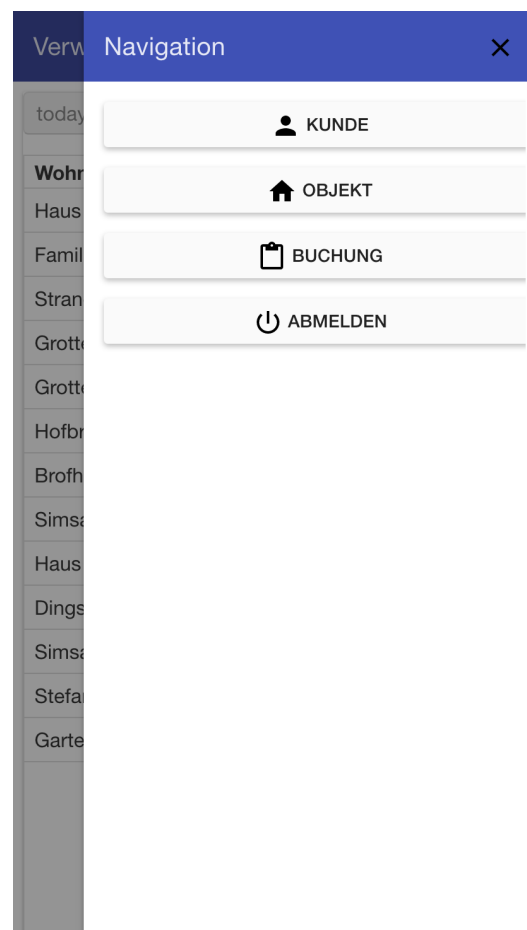


Abbildung 5.4  
Seitennavigation auf Mobilgeräten

### 5.2.2 Main

In der Main befindet sich der eigentliche Kalender, in dem in einer Monatsübersicht die Buchungen dargestellt werden. Dieser teilt sich in folgende zwei Bereiche:

#### Objektübersicht

Alle Objekte werden auf der linken Seite in form einer Liste dargestellt.

#### Buchungsübersicht

Alle Tage des aktuellen Monats werden als Spalten dargestellt. Die Buchungen werden als Blöcke angezeigt und verbinden die einzelnen Tage an denen das Objekt gebucht ist.

## 5.3 Kunde

Zusammen mit dem Objekt bildet der Kunde eine Buchung. Für die Kundeninformationen wurden alle Felder aus dem Projektzielen implementiert (Abbildung 5.6). Zudem sind alle Felder ausgenommen der Zusatzinformationen und der Firma pflicht. Diese werden beim Abschicken des Formulars geprüft, ob sie leer sind. Um die Eingabe des Geburtstages zu vereinfachen wurde jeweils für Tag, Monat und Jahr ein Dropdown-Menü bereitgestellt. Da das Mindestalter für Buchungen 18 Jahre ist, ist das höchste auszuwählende Jahr immer 18 Jahre vom aktuellen aus gerechnet. Alle Zeitangaben werden im UNIX TIMESTAMP gespeichert. Sind alle Felder korrekt ausgefüllt, kann das Formular abgeschickt werden. Sobald die `submit` Funktion im Controller aufgerufen wird, werden alle Felder aus dem View in einem `customer` JSON-Objekt gespeichert. Diese werden dem `Customer` Model übergeben und das Dialogfenster geschlossen.

#### Listing 5.1

submit Methode im customer Controller

```

1  $scope.submit = function () {
2      $scope.birthday = new Date($scope.yearOfBirth + '-'
3          + $scope.monthOfBirth + '-' + $scope.
4          dayOfBirth).getTime();
5      var customer = new Customer({
6          Company: $scope.company,
7          FirstName: $scope.firstname,
8          LastName: $scope.lastname,
9          BirthDate: $scope.birthday,
10         Street: $scope.street,
11         City: $scope.city,
12         ZipCode: $scope.zipcode,
13         Email: $scope.email,
14         Phone: $scope.phone,
15         Custom: $scope.custom,
16         Id: $scope.customerID || undefined
17     });

```

```
16         Customers.upsert(customer);  
17         $scope.hide();  
18         $rootScope.$broadcast('showToast');  
19     };
```

Soll ein bestehender Kunde bearbeitet werden, wie in Abbildung 5.7 zu sehen, muss dieser zunächst über das in Abbildung 5.5 gezeigte Suchfeld ermittelt werden. Mit einem Klick auf das Lupensymbol im Kopfbereich des Dialog wird die Suchleiste jeweils ein- oder ausgeblendet. Sobald das Suchfeld fokussiert wird, wird eine Funktion im Controller angestoßen, welche alle Kunden mit Vor und Nachname als Auswahlmöglichkeit auflistet. Dafür wird im Model die Liste aller Kunden aus der Datenbank angefordert. Das Model liefert dem Controller ein Array mit allen Kunden als JSON-Objekt. Bei einer hohen Anzahl an Kunden kann sich die Suche durchaus schwierig gestalten. Aus diesem Grund kann der Kunde durch eintippen von Buchstaben gefiltert werden. Jeder weitere Buchstabe schränkt die Suche nach dem Kunden ein und es werden nur Kunden angezeigt dessen Nachname die eingegebene Zeichenkette beinhalten.

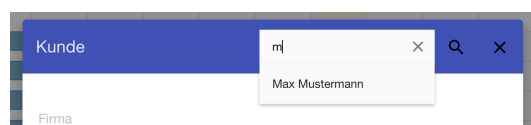


Abbildung 5.5  
Kundensuche



Kunde

Firma

Vorname Nachname

Tag ▼ Monat ▼ Jahr ▼

Strasse und Hausnummer

Postleitzahl Stadt

Email Telefonnummer

Zusatzinformation 0/150

ABBRECHEN SPEICHERN

Abbildung 5.6  
Neuen Kunden erstellen

Kunde Musterfirma

Firma

Vorname Nachname  
Max Mustermann

Tag Monat Jahr  
1 1 1998

Strasse  
Musterstrasse 8

Postleitzahl Stadt  
123456 Musterstadt

Email Telefonnummer  
max@musterfirma.de 012 / 345678

Zusatzinformation  
Nichtraucher 12/150

LÖSCHEN ABBRECHEN SPEICHERN

Abbildung 5.7  
Kunden bearbeiten

Wurde ein Kunde ausgewählt, werden alle Daten aus dem JSON-Objekt ausgelesen und in die jeweiligen Input Felder eingefügt. Gleichzeitig wird ein Button zum löschen des Kunden angezeigt. Um ein Fehler beim Löschen vorzubeugen muss zusätzlich noch ein Dialog bestätigt werden. Ist das der Fall, übergibt der Controller dem Model die UUID des Kunden in einer `delete` Funktion.

Wenn Daten verändert wurden können diese wie auch beim Hinzufügen eines neuen Kunden ganz normal bestätigt werden. Im Controller wird die `upsert` Methode des Model aufgerufen. Je nachdem, ob es sich dabei um einen bestehenden Eintrag in der Datenbank handelt oder einen neuen Kunden, wird ein `Update` oder `Insert` durchgeführt.

Soll weder ein neuer Kunde hinzugefügt noch ein bestehender bearbeitet werden kann das Dialog über den "Abbrechen," Button, das "X," oder einen Klick ausserhalb des Dialogs geschlossen werden. Der Controller ruft die `hide` Methode des Dialogs auf und das Dialogfenster wird ausgeblendet. Dabei werden auch alle Daten aus den Feldern entfernt.

## 5.4 Objekt

Als Objekt wird eine Ferienwohnung oder Haus bezeichnet. Wie auch beim Kunden können bei einem Objekt verschiedene Informationen gespeichert werden. Die beiden Felder aus den Projektzielen wurden implementiert (Abbildung 5.8). Dazu gehört die Eingabe des Namens und die Personenanzahl in ein Textfeld. Wird das Feld fokussiert, und ohne Eingabe verlassen, erscheint eine Fehlermeldung mit dem Hinweis, dass das Feld auszufüllen ist (Abbildung 5.9). Wird dieser Hinweis ignoriert und der Nutzer versucht dennoch das Formular abzuschicken überprüft der View, ob alle als Pflicht gekennzeichneten Felder ausgefüllt wurden und der **submit** Button gedrückt wurde. Ist eines dieser beiden **false**, wird das Formular nicht abgeschickt.

Abbildung 5.8  
Objekt erstellen

Abbildung 5.9  
Objekt bearbeiten

Auch bei der Angabe der maximalen Personenanzahl wurde darauf geachtet fehlerhafte Eingaben abzufangen. Standardmäßig ist eine Mindestpersonenanzahl von 1 eingestellt. Der Nutzer hat die Möglichkeit mit Hilfe von Pfeilen den Wert zu Verändern. Die Maximale Anzahl von Personen ist 10 pro Objekt. Da es auch möglich ist Manuell eine Zahl einzugeben wird ebenfalls überprüft, ob sich die Zahl zwischen 1 und 10 befindet. Ist dies nicht der Fall wird ein Hinweis ausgegeben und das Formular kann nicht abgeschickt werden. Sind alle Angaben korrekt und der **submit** Button wird betätigt, wird das Formular abgeschickt. Im Controller wird eine Methode aufgerufen in der die Daten aus den Feldern in ein JSON-Objekt gespeichert werden. Dieses wird der **upsert** Methode des Models übergeben. Die Überprüfung der Personenanzahl findet im View statt.

### Listing 5.2

#### Validierung der Personenanzahl im View

```

1      <input required type="number" step="any" name="rate" ng-
      model="size" min="1" max="10" />
2      <div ng-messages="resourceForm.size.$error">
3          <div ng-message="required">Personenanzahl von
              1 bis 10</div>
4      </div>

```

Um ein bereits existierendes Objekt bearbeiten zu können muss dies anders als bei den Kunden in der Auflistung im Kalender ausgewählt werden (Abbildung 5.10). Auch hier werden die Daten anhand der UUID aus der Datenbank geladen und in die Felder eingefügt. Zusätzlich wird der Button zum Löschen des Objektes angezeigt. Die Besonderheit bei der Änderung von Daten ist eine Eigenschaft im JSON-Objekt in dem die Informationen der Felder gespeichert werden. Neben dem Namen und der Personenanzahl wird auch die ID des Objektes gespeichert. Ist, wie bei einer Änderung, bereits eine UUID vorhanden, wird diese als Wert gespeichert. Wird ein Objekt neu hinzugefügt, ist die ID `undefined`. Diese Information wird in der `upsert` Methode des Models abgefragt und dementsprechend ein neues Objekt hinzugefügt oder ein bestehendes aktualisiert.

Abbildung 5.10  
Eingabefehler bei Objekt

## 5.5 Buchung

Eine Buchung ist ein Zusammenschluss aus Kunde, Objekt, Personenanzahl, Startdatum und Enddatum. Wie bei der Bearbeitung von Kunden kann der Nutzer den Kunden und das Objekt über ein Suchfeld filtern und Auswählen. Wenn eine neue Buchung hinzugefügt wird, wird vorerst die Auswahl der Personenanzahl ausgeblendet (Abbildung 5.11). Erst wenn ein Objekt im Suchfeld ausgewählt wurde, wird die Auswahl in form eines Dropdown-Menüs im View angezeigt. Im Controller wird die maximale Personenanzahl des ausgewählten Objektes aus dem JSON-Objekt entnommen und somit die zur Auswahl stehenden Zahlen ermittelt. So zeigt das Dropdown-Menü nur genau die Anzahl von Personen an, die dieses Objekt zur Verfügung stellt. Für das Startdatum und Enddatum wurden, anders als beim Kunden, ein Datepicker bereit gestellt, der es dem Nutzer erlaubt das Datum durch einfache Auswahl in einem Kalender festzulegen (Abbildung 5.12). Für diesen ist Standardmäßig das aktuelle Datum eingestellt.

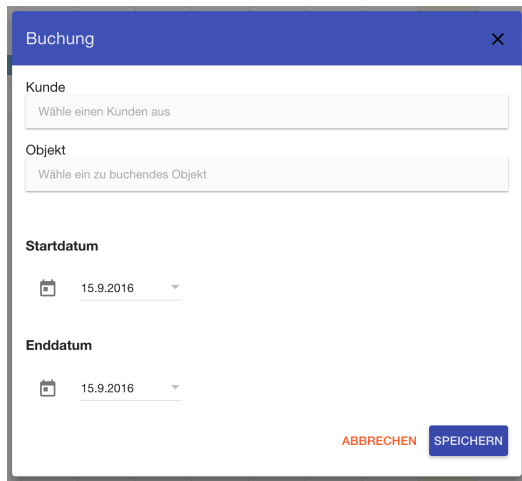


Abbildung 5.11  
Objekt erstellen

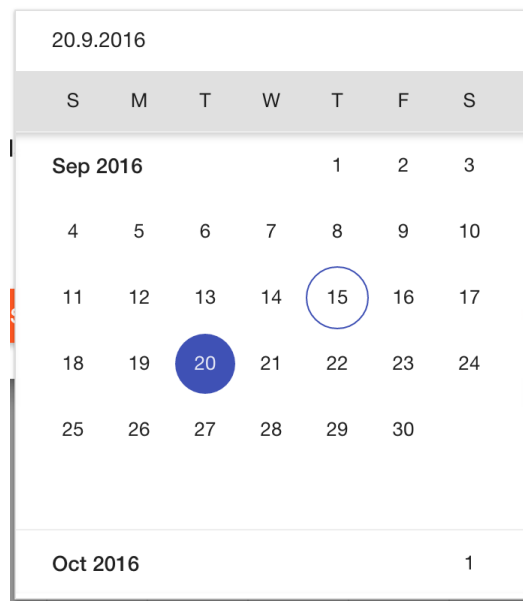


Abbildung 5.12  
Auswahl des Datum

Um eine Buchung bearbeiten zu können muss diese, anders als beim Kunden oder Objekt, zunächst in der Buchungsübersicht ausgewählt werden. Hier wird die Personenauswahl von Beginn an angezeigt und ist mit dem eingetragenen Wert vordefiniert. Auch hier besteht die Möglichkeit, durch bestätigen des Löschen-Dialogs die Buchung aus der Datenbank zu entfernen (Abbildung 5.13).

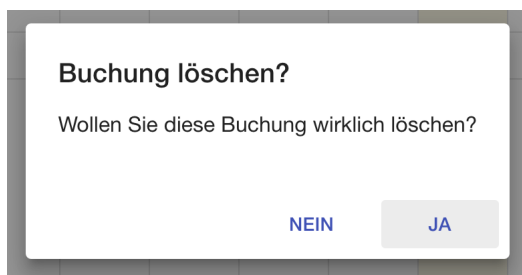


Abbildung 5.13  
Objekt erstellen



Abbildung 5.14  
Objekt Fehlereingabe

Wird ein Dialog bestätigt, erscheint für drei Sekunden am oberen rechten Fensterrand eine Nachricht (Abbildung 5.14). Ein sogenannter Toast fährt herunter und zeigt die Erfolgsmeldung und einen Button zum schließen an. In der mobilen Ansicht fährt der Dialog, in voller Bildschirmbreite, von unten herein.

## 5.6 Kalender

Der Kalender ist der Hautbestandteil der Anwendung (Abbildung 5.2). Auf ihm kann der Nutzer alle Buchungen einsehen und editieren. Hinzu kommt die Verwaltung der Mietobjekte. Diese sind auf der linken Seite aufgelistet. Auf der horizontalen Achse des Kalenders befinden sich die einzelnen Tage eines Monats sowie die vorhandenen Buchungen.

Dank der umfangreichen Konfigurations-Möglichkeiten kann der Kalender in seinem Verhalten sehr gut angepasst werden. Einen Teil dieser Anpassungen wir im nachfolgenden erläutert.

### 5.6.1 Drag'n Drop

Der Kalender bietet die Möglichkeit, Events mittels Drag'n Drop dieses innerhalb zu verschieben. Dieses Verhalten wird von der Option `draggable = true` freigeschaltet. Die Events, welche im Hintergrund liegen werden automatisch nach dem Verschieben aktualisiert. Eine Aktualisierung mit der Datenbank erfolgt jedoch nicht. Hierfür steht die Callback-Methode `eventDrop` bereit, welche durch die Kalender-Konfiguration überschrieben werden kann. Im Falle der vorliegenden Buchungs-Anwendung wird dabei der folgende Code ausgeführt:

Listing 5.3

Aktualisierungscode nach einem Drag n Drop-Event

```

1 function updateBooking(event){
2     var newEvent = {
3         Id : event.id || event.Id,
4         Resource : event.resourceId,
5         StartDate : event.start.toDate(),
6         EndDate : event.end.toDate(),
7         Customer : event.customerId,
8         Size : event.size
9     };
10    Bookings.upsert(new Booking(newEvent));
11 }
```

Diese Funktion erstellt aus dem Kalender-Event ein neues Model und ruft die `upsert`-Methode der Buchungs-Collection auf. Hierbei wird die Entscheidung getroffen, ob eine neue Buchung hinzukommt oder eine bestehende geupdated wird.

### 5.6.2 Resizing

Fullcalendar bietet die Möglichkeit, Buchungen mit einem einfachen Resizing zu verändern. Dabei kann der Nutzer die Buchung einfach an beiden Enden anfassen und auf eine beliebige Länge ziehen oder drücken. Das Updaten der Buchung erfolgt analog zum Updaten bei der Drag'n Drop-Funktion.

---

### 5.6.3 Selektion

Für Anwender ist dieses Feature beim Anlegen von Buchungen sehr nützlich. Hierüber lassen sich aufeinander folgende Tage markieren und die Selektion mit einer Aktion verknüpfen. In diesem Fall wird der Dialog zum Erstellen einer neuen Buchung geöffnet. Dabei sind die Felder Startdatum, Enddatum und das Objekt bereits vordefiniert. Anschließend muss lediglich der Kunde sowie die Anzahl der Personen nachgetragen werden.



## 6 Backend

Für die entwickelte Anwendung kommt die Plattform „Firebase“ von Google zum Einsatz.

Diese stellt neben einer Echtzeit-Datenbank, Authentifizierungs-Funktionen und Cloud-Messaging, auch Speicherplatz sowie ein Hosting für Web-Apps zur Verfügung. In der Basis-Version, welche auch für die Entwicklung der vorliegenden Anwendung verwendet wurde, ist die Nutzung kostenlos. Für höhere Leistungsansprüche stehen kostenpflichtige Pakete zur Verfügung. Im nachfolgenden erfolgt eine Erläuterung der einzelnen Komponenten, die von der Anwendung benutzt werden.

### 6.1 Echtzeit-Datenbank

Bei der von Firebase verwendeten Datenbank handelt es sich um eine sogenannte NoSQL-Datenbank. Im Gegensatz zu anderen Datenbank-Systemen wie beispielsweise MySQL, erfolgt die Abspeicherung der Daten dokumentenbasiert im JSON-Format. Dokumentenbasiert bedeutet, dass einzelne Einträge der Datenbank als Objekte abgelegt werden. Die Objekte sind von Grund auf nicht an Schemata gebunden und können beliebige Strukturen beinhalten. Dabei können auch Objekte, die von ihrer Gruppierung her gleich sind, unterschiedlich viele und anders typisierte Daten beinhalten.

### 6.2 Organisation

Die Benennung der Gruppen von Datensätzen ist von Firebase sehr flexibel gestaltet und ermöglicht beliebige Namen. Es existieren folgende drei Gruppen:

- bookings (Abbildung 6.1)
- customers (Abbildung 6.2)
- resources (Abbildung 6.3)



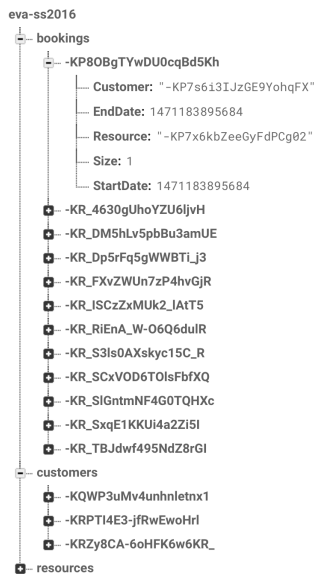


Abbildung 6.1  
Datenbank Buchungen

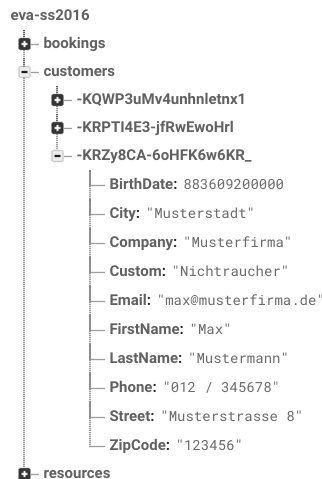


Abbildung 6.2  
Datenbank Kunde

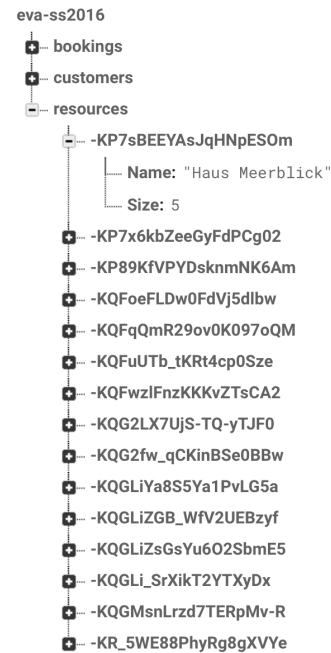


Abbildung 6.3  
Datenbank Objekt

Jede Gruppe kann beliebig viele Untergruppen oder Objekte besitzen. Dabei gibt es keine Einschränkungen.

Die Daten in den Objekten sind in der JSON-typischen Key:Value-Notation abgelegt. Identifiziert werden die Objekte über einzigartigen Schlüssel, die als ID fungieren. Über diese Schlüssel lassen sich später in der Anwendung einzelne Objekte direkt abfragen. Weiterhin können über diese Keys Beziehungen zu anderen JSON-Objekten aufgebaut werden. Behandelt werden diese Schlüssel wie Zeichenketten und erfahren somit keiner speziellen Behandlung sofern sie als Werte eines JSON-Objektes abgelegt sind.

## 6.3 Zugriffsverwaltung

Firebase bietet dem Administrator sehr flexible Zugriffsregeln, mit welchen die exakte Steuerung von Lese- und Schreibzugriffen möglich ist. Die Ausgangskonfiguration beschränkt das Lesen und Schreiben von Datensätzen auf angemeldete Benutzer. Auf die Möglichkeiten der Authentifizierung wird in dem Kapitel 6.5 eingegangen.

Wie die Organisation der Daten in der Datenbank im JSON-Format angelegt ist, werden auch die Regeln als JSON-Notation hinterlegt.

### Listing 6.1

Beispiel-Konfiguration für Datenbankzugriffe

1 {

```

2      "rules": {
3          ".read": "auth != null",
4          ".write": "auth != null"
5      }
6  }

```

Der hier dargestellte Konfigurationsabschnitt zeigt die Regeln für den Lese- und Schreibzugriff auf die gesamte Datenbank. Demnach dürfen nur angemeldete Benutzer lesen und schreiben. Firebase ermöglicht es, für jede Gruppe eine separate Konfiguration zu generieren. Entscheidend dafür, ob Daten gelesen oder geschrieben werden dürfen ist das Ergebnis des Ausdrucks bei `.read` und `.write`. Diese müssen jeweils bei ihrer Ausführung zu einem boolschen Wert evaluieren. Dabei können diese Regeln eine beliebige Komplexität aufweisen. In der vorliegenden Anwendung wurde auf eine komplexe und spezifische Konfiguration verzichtet, da der Anwendungsfall keine spezielle Behandlung von verschiedenen Benutzern fordert. Alle Datensätze können nur von authentifizierten Benutzern gelesen und geschrieben werden.

## 6.4 Lesen, Schreiben, Echtzeit-Synchronisation

Der entscheidende Punkt bei der Auswahl der Backend-Plattform war der Aspekt der Echtzeit-Synchronisation der Datenbank zwischen den angemeldeten Clients. Um eine Echtzeit-Anwendung zu realisieren, sollten Daten, die bei Client A generiert, verändert oder gelöscht wurden, bei Client B ohne ein Neuladen der gesamten Anwendung aktualisiert werden.

Diese Synchronisation wird von Firebase durch die Verwendung der Web-Socket-Technologie ermöglicht. Technisch gesehen basiert dies auf dem Austausch von Nachrichten. Dabei baut die Anwendung eine konstante Verbindung zu dem Firebase-Server auf. Sobald eine Daten-Transaktion abgeschlossen ist, sendet Firebase eine Nachricht an alle verbundenen Clients. Bei der Entwicklung kann der Entwickler entscheiden, ob er auf etwaige Nachrichten reagieren möchte. Bei den Nachrichten wird zwischen fünf verschiedenen Ereignissen unterschieden:

### **value**

Allgemeine Änderung an einer Gruppe

### **child\_added**

Einer Gruppe wird ein neues Element hinzugefügt

### **child\_changed**

In einer Gruppe ändert sich ein einzelnes Kind-Element

### **child\_removed**

Ein Element wird aus einer Gruppe entfernt

## `child_moved`

Die Reihenfolge eines Kind-Elements verändert sich

Wie dieses Feature in die entwickelte Anwendung integriert wurde, wird im Kapitel 6.6.2 Collections genauer erläutert.

## 6.5 Authentifizierung

Bei fast jeder zu entwickelnden Anwendung steht das Thema Nutzer-Authentifizierung auf dem Plan. Standardmäßig erfolgen Implementierungen, die Benutzer mit Hilfe von E-Mail Adresse / Benutzernamen und Passwort authentifizieren. Leider kann es hierbei immer wieder zu Fehlern in der Entwicklung oder im Design kommen, sodass es unter Umständen möglich ist, Zugang zu gesperrten Bereichen oder Daten zu erlangen. Um dem Entwickler einerseits dieses Risiko zu nehmen und andererseits erheblichen Implementierungs-Aufwand zu ersparen, bietet Firebase auch einen Authentifizierungs-Service an. Per Standard ist das E-Mail-Passwort-Verfahren aktiviert. Zudem bietet Firebase auch die Authentifizierung mit Hilfe von Accounts von Facebook, Google, Twitter und GitHub an. Dabei werden OAuth 2.0 und OpenID Connect verwendet. Weiterhin lassen sich beliebig viele weitere OAuth-Services hinzufügen.

In der vorliegenden Anwendung wurde das E-Mail-Passwort-Verfahren verwendet, da analog zum Zugriffsschutz auch hier nur ausgewählte Benutzer Zugriff auf die Anwendung erhalten und somit keine Authentifizierung über weitere Dienste nötig sind.



E-Mail-Adresse	Quelle	Erstellt	Angemeldet	Nutzer-UID
chrome@web.de	✉	28.08.2016	28.08.2016	E5p7usvUopUDM68dt5Warhb...
florian.guembel1@gmail.com	✉	13.08.2016	13.08.2016	HMi6kelYgvMZCJ9Q5Y10p5...
s@s.de	✉	08.08.2016	13.09.2016	UMJG9dxMV7NNfsgIRB2ta74...

**Abbildung 6.4**  
Tabelle mit Email und Password

Wie diese Anmeldung in der Anwendung abläuft, wird im entsprechenden Kapitel 5 im Bereich Login erläutert.

## 6.6 Weitere Firebase-Funktionen

Neben den bereits beschriebenen Funktionen stellt Firebase zusätzlich Hosting und Datenspeicher, sowie Cloud Messaging, Analytics und Monetarisierungsoptionen zur Verfügung. Zudem bietet Firebase sehr gute Voraussetzungen für schnell wachsende Anwendungen, da über die Web-Konsole schnell und einfach Kapazitäten angefordert werden können.

### 6.6.1 Technische Umsetzung der Datenbank-Anbindung

Um die Arbeit mit Datensätzen und Daten-Listen zu erleichtern, wurde ein System nach einem Model-Collection-Pattern entwickelt. Demnach sind einzelne Datensätze über Models organisiert. Diese Models werden in typisierten Collections zusammengefasst. Im nachfolgenden werden der Aufbau und die Funktionsweise beider Konstrukte erläutert.

### Konfiguration für Firebase

Für jede Anwendung, die auf den Firebase-Service zugreift, muss zunächst in der Firebase-Console ein API-Key generiert werden. Dieser identifiziert die Anwendung bei Firebase. Mit der Konfiguration kommen zudem die URL zur Authentifizierung sowie Adressen für Storage und Datenbank mit.

Listing 6.2

Konfiguration des Frontends

```

1  var config = {
2      apiKey: "AIzaSyD5n8G5I9feSfmW9gr59YwhDsG93Z5m4fM",
3      authDomain: "eva-ss2016.firebaseio.com",
4      databaseURL: "https://eva-ss2016.firebaseio.com",
5      storageBucket: "eva-ss2016.appspot.com"
6  };

8  firebase.initializeApp(config);

10 window.database = firebase.database();

```

Im weiteren Verlauf des Konfigurations-Scriptes wird ein Firebase-Datenbank-Objekt erstellt. Dieses bietet alle nötigen Funktionalitäten, um später mit der Datenbank arbeiten zu können. Damit jeder beliebige JavaScript-Code darauf zugreifen kann, erfolgt die Speicherung als Property des globalen `window`-Objektes.

### Models

Als Model bezeichnet man eine Klasse, die die Struktur eines Objektes beschreibt. Ein Model hat einen Constructor, über welchen sich ein neues Objekt dieses Types generieren lässt. Im Fall der vorliegenden Anwendung bilden drei

verschiedene Models die Datenstruktur des Systems ab.  
Dazu gehören:

### Booking

Hält eine Buchung

### Customer

Beschreibt den Kunden

### Resource

Beschreibt das zu vermietende Objekt

Über ein Model lässt sich ein Objekt eines spezifischen Types generieren. Innerhalb der AngularJS-Anwendung wird das Model als Factory-Service implementiert. Dies ermöglicht es, beliebig viele Instanzen dieses Models zu generieren. Im nachfolgenden der Aufbau des Resource-Models:

#### Listing 6.3

Hauptteil des Resource-Models

```

1  angular.module('bookingCalendarApp')
2    .factory('Resource', function ($log) {
3      function Resource(properties){
4
5        var self = this;
6
7        this.Id      = undefined;
8        this.Size    = undefined;
9        this.Name    = undefined;
10
11       function extend(properties){
12         [...]
13       }
14       extend(properties);
15     }
16     return Resource;
17   });

```

Wesentlicher Bestandteil des Models sind seine Properties, welche die beschreibbaren Felder darstellen. Bei jeder Instanziierung wird die Methode `extend()` aufgerufen, welche die übergebenen Werte auf das Model überträgt. Innerhalb der Anwendung können Models in allen möglichen Komponenten verwendet werden. Zu beachten ist, dass es anders als bei komplett objektorientierten Sprachen wie beispielsweise Java, keine `getter`- oder `setter`-Methoden gibt und die Properties des Models alle als Public angesehen werden können.

## Remote Objects

Der Service `RemoteObjects` agiert als Factory zum Erstellen einer Collection für spezifische Models. Sie ist über einen Angular-Service als Singleton-Klasse

implementiert und besitzt nur eine Methode zum Generieren einer neuen Collection.

#### Listing 6.4

Code des RemoteObjects-Service

```

1  angular.module('bookingCalendarApp')
2    .service('RemoteObject', function (Collection) {
3      var service = {};
4      service.createCollection = function(name, path, Model
5        , realtime){
6
7        if(!name || !path || !Model){
8          throw new Error('remoteObject:: Missing
9            parameter in Object')
10        }
11        return new Collection({
12          name : name,
13          path : path,
14          realtime : realtime,
15          Model : Model
16        });
17      };
18      return service;
19    });

```

Wie zu sehen, fordert die Factory bei der Instanziierung bis zu vier Parameter. Über den Parameter **name** wird der Name der Collection bestimmt, **path** ist der Pfad, unter welchem die Objekte dieses Types innerhalb von Firebase abgelegt sind, **Model** ist das Model-Objekt und **realtime** ein Indikator, ob die Collection auf Echtzeit-Updates der Datenbank hören soll. Rückgabewert der Methode **createCollection()** ist immer die Collection des jeweiligen Model-Types. Wie diese Collections funktionieren, wird im folgenden erläutert.

### 6.6.2 Collections

Eine Collection bezeichnet eine Art Sammlung für einen bestimmten Datentyp. Sie stellt Funktionen zur Verfügung, mit welchen der Entwickler einfache Datenbank-Abfragen generieren und Daten speichern kann. Ebenso wie der RemoteObjects-Service, ist auch eine Collection als Service implementiert, sodass nur jeweils einer für jedes Model existieren kann.

#### Instanziierung

Über den Service **RemoteObject** wird die neue Collection instanziiert. Wie bereits beschrieben werden Name, Pfad, Model und Realtime-Indikator an den Konstruktor der Collection übergeben. Zunächst werden alle übergebenen Parameter in interne Variablen geschrieben. Anschließend wird ein Referenzobjekt mit Hilfe der Firebase-Bibliothek zur Datenbank generiert

## Listing 6.5

Befehl zum Generieren einer Firebase-Referenz

```
1  var reference = $window.database.ref(this.path);
```

Diese Referenz wird in der gesamten Collection von Methoden verwendet, die Daten abfragen. Zudem greift der Mechanismus der Echtzeit-Funktionalitäten auf dieses Referenz-Objekt zu.

## Methoden

Nachdem nun die Vorbereitungen abgeschlossen wurden, folgt ein Einblick auf die von der Collection zur Verfügung gestellten Methoden. Im Umfang enthalten sind standardmäßig `insert`, `remove`, `upsert` und `find`. Zusätzlich existiert die Methode `list`.

Eventuell stellt sich die Frage, warum die Implementierung eines Collection-Services angestrebt wurde, da sich alle Datenbank-Operationen auch manuell ansteuern lassen. Hierzu lässt sich sagen, dass dieser Service viele Vorteile bietet:

- Code wird nur einmal geschrieben
- Code lässt sich beliebig oft wiederverwenden
- Wartbarkeit ist deutlich höher
- Asynchronität von Anfragen wird durch Promise-Chaining entfernt

Nachfolgend eine Erläuterung der einzelnen Methoden.

### insert

Die `insert`-Methode dient zum Einfügen neuer Datensätze. Dabei nimmt diese als Argument das einzufügende Model mit den Daten entgegen.

## Listing 6.6

Insert-Methode einer Collection

```
1  this.insert = function(model){
2      var deferred = $q.defer();
3      reference
4          .push(prepareModel(model))
5          .then(function(result){
6              deferred.resolve(result);
7          })
8          .catch(function(error){
9              deferred.reject(error);
10         })
11     ;
12     return deferred.promise;
13 };
```

Wie bereits zu sehen verwendet die `insert`-Methode ein `deferred`-Objekt. Im vorausgegangenen Abschnitt wird erwähnt, dass durch die Collection

die Asynchronität durch dieses Promise-Chain aufgebrochen wurde. Zunächst erfolgt ein Zugriff auf das Referenz-Objekt und der Aufruf seiner `push`-Methode. Die `push`-Methode erwartet ein beliebiges Objekt und legt dieses in der Datenbank ab. Anschließend wird das abgespeicherte Objekt von der Datenbank zurückgeliefert. Das empfangene Objekt wird über das Promise-Chain zurückgegeben. Vorteil dieses Promise-Chain ist, dass Funktionen, die von dem Ergebnis abhängig sind, in einfachen Funktionsaufrufen nacheinander aufgerufen werden können.

Alle Methoden mit Datenbank-Zugriff verwenden das Promise-Chain.

### **remove**

Bei der `remove`-Methode wird nur die `remove`-Funktion der Datenbank-Referenz aufgerufen. Wichtig ist hierbei, dass eine neue temporäre Referenz generiert wird, die nur auf dieses Objekt (über dessen Id) zeigt.

### **upsert**

Die `upsert`-Methode ist eine Kombination aus `insert` und `update`. Besitzt das Model, welches der Funktion übergeben wurde eine Id, so wird ein `set`-Aufruf auf die Referenz dieses Objektes gesendet. Über `set` können dem referenzierten Objekt beliebige Felder gesetzt und geändert werden. Für den Fall, dass keine Id vorhanden ist, wird die `insert`-Methode aufgerufen.

### **find, list**

Bei `find` und `list` handelt es sich um zwei fast identische Methoden. `list` bietet die Möglichkeit, das gesamte Datenpaket einer Referenz zu empfangen. Mit `find` lässt sich nach einem Datensatz suchen. Um einen solchen zu finden, muss ein Such-Objekt übergeben werden. Ein Such-Objekt besteht aus Key-Value-Paaren. Die Keys müssen dabei auf die Properties eines Objektes in der Datenbank passen. Die Ergebnisse werden wie gewohnt in einem Promise-Chain zurückgegeben.

## **Echtzeit-Anbindung**

Um den Anspruch zu erfüllen, dass eine automatische Synchronisierung der Daten erfolgt, musste über die Collections ein System implementiert werden, welches auf die Ereignisse reagiert, die von Firebase bei Daten-Änderungen gefeuert werden.

Jede Instanz der Web-Anwendung ist über eine Web-Socket-Verbindung mit der Firebase-Datenbank verbunden. Über diese Verbindung erfolgt sämtliche Kommunikation mit der Datenbank. Zum einen werden Daten gesendet, zum anderen Nachrichten über Ereignisse verschickt.

Bei den Ereignissen wird zwischen vier verschiedenen Ereignissen unterschieden:

### **child\_added**

Zu der Referenz wird ein neues Objekt hinzugefügt



**child\_changed**

Ein Objekt erfährt ein Update

**child\_removed**

Ein Objekt der Referenz wird entfernt

**value**

Indiziert eine allgemeine Änderung auf dem Referenz-Objekt

Auf diese Events registriert die Collection-Klasse jeweils einen Listener:

**Listing 6.7**

Registrierung der Event-Handler von Datenbank-Events

```

1  function bindRealTimeHandlers(){
2      reference.on('child_added', dataAdd);
3      reference.on('child_changed', dataChange);
4      reference.on('child_removed', dataRemove);
5      reference.on('value', datasetChange);
6  }

```

Die registrierten Event-Listener empfangen die Daten, die von der Datenbank mit der Nachricht mitgesendet werden. Anschließend wird in der gesamten Angular-Anwendung ein Event getriggert, welches wiederum die Daten enthält. Dadurch kann an jeder Stelle in der Anwendung - egal ob Controller, Service oder Direktive - auf die Ereignisse reagiert werden. In diesem Falle, liegen die Listener alle im **Calendar-Controller** und updaten die Daten im Kalender.

Nachfolgend ein Code-Beispiel, welches auf das Hinzufügen eines Buchungsobjektes reagiert:

**Listing 6.8**

Event-Handler für hinzukommende Ressourcen

```

1  $rootScope.$on('Resource::added', addRemoteResource);
2  function addRemoteResource(event, addedResource){
3      calendarInstance.fullCalendar('addResource',{
4          d : addedResource.Id,
5          title : addedResource.Name
6      });
7  }

```

Sobald ein neues Ressourcen-Objekt eintrifft, wird dem Kalender diese Ressource hinzugefügt.

Zusammengefasst stellt Firebase eine perfekte Plattform zur schnellen Datensynchronisation bereit.

## 7 Fazit

Die Entwicklung dieses Projektes hat einige sehr interessante Aspekte der interaktiven und innovativen Web-Entwicklung hervorgebracht. Angefangen bei der Verwendung der Firebase-Plattform von Google. Diese bot einen Funktionsumfang, der, im Falle einer Selbstentwicklung, einem Entwickler einen sehr hohen Aufwand erspart. Zu Beginn ist Firebase kostenlos und ermöglicht auch Studenten, die Plattform ohne Kosten zu nutzen.

In jedem Fall konnte sich die Plattform für die weitere Nutzung in Uni-Projekten und darüber hinaus empfehlen.

Der entstandene Prototyp funktioniert sehr gut. Vor allem die Synchronisation der Daten zwischen verschiedenen Clients läuft erstaunlich schnell. Eine Weiterentwicklung des Prototyps ist voraussichtlich nicht vorgesehen

Zusammengefasst lässt sich sagen, dass das Projekt uns persönlich in unseren Interessensgebieten durchaus weitergebracht hat und neue Technologien dadurch zu unserem Repertoire gehören.

Kommende Projekte würden wir wieder mit dieser Technologie-Kombination entwickeln.

# Abbildungsverzeichnis

<b>4</b>	<b>Projektstruktur</b>	
4.1	Angelegte Projektstruktur .....	12
<b>5</b>	<b>Frontend</b>	
5.1	Anmeldeformular .....	16
5.2	Hauptseite .....	16
5.3	Ansicht auf Mobilgeräten .....	17
5.4	Seitennavigation auf Mobilgeräten .....	17
5.5	Kundensuche .....	19
5.6	Neuen Kunden erstellen .....	20
5.7	Kunden bearbeiten .....	20
5.8	Objekt erstellen .....	21
5.9	Objekt bearbeiten .....	21
5.10	Eingabefehler bei Objekt .....	22
5.11	Objekt erstellen .....	23
5.12	Auswahl des Datum .....	23
5.13	Objekt erstellen .....	23
5.14	Objekt Fehlereingabe .....	23
<b>6</b>	<b>Backend</b>	
6.1	Datenbank Buchungen .....	28
6.2	Datenbank Kunde .....	28
6.3	Datenbank Objekt .....	28
6.4	Tabelle mit Email und Password .....	30

# Listings

## 5 Frontend

---

5.1 submit Methode im customer Controller .....	18
5.2 Validierung der Personenanzahl im View .....	21
5.3 Aktualisierungscode nach einem Drag n Drop-Event .....	24

## 6 Backend

---

6.1 Beispiel-Konfiguration für Datenbankzugriffe .....	28
6.2 Konfiguration des Frontends .....	31
6.3 Hauptteil des Resource-Models .....	32
6.4 Code des RemoteObjects-Service .....	33
6.5 Befehl zum Generieren einer Firebase-Referenz .....	34
6.6 Insert-Methode einer Collection .....	34
6.7 Registrierung der Event-Handler von Datenbank-Events .....	36
6.8 Event-Handler für hinzukommende Ressourcen .....	36



# Glossar

**AJAX** *siehe* Asynchronous JavaScript and XML.

**Asynchronous JavaScript and XML** Konzept der Asynchronen Datenübertragung.

**Cascading Style Sheet** Gestaltungssprache für elektronische Dokumente und Programme wie HTML-Webseiten oder JavaFX-Anwendungen.

**CSS** *siehe* Cascading Style Sheet.

**HTML** *siehe* Hypertext Markup Language.

**Hypertext Markup Language** Eine textbasierte Auszeichnungssprache zur Strukturierung digitaler Dokumente wie Texte mit Hyperlinks, Bildern und anderen Inhalten.

**JavaScript** Skriptsprache für das Web.

**JavaScript Object Notation** Kompaktes Datenformat in leicht verständlicher Textform, in etwa vergleichbar mit XML.

**JS** *siehe* JavaScript.

**JSON** *siehe* JavaScript Object Notation.

**No Structured Query Language** Datenbank mit nicht- relationalem Ansatz.

**NoSQL** *siehe* No Structured Query Language.

**Uniform Resource Locator** Identifizierung einer Netzwerkressource, beispielsweise ein Server.

**URL** *siehe* Uniform Resource Locator.