

# 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**

Eine Agentur, die Wohnräume vermietet, möchte ein Programm haben um die Buchungen zu Verwalten. Zu den Prämissen gehört, dass es Betriebssystem unabhängig nutzbar ist und Änderungen auf schnellstmöglichen weg an die Mitarbeiter kommuniziert wird um Doppebuchungen zu vermeiden. Es soll ebenfalls möglich sein zur gleichen Zeit Buchungen, Kunden und Wohnungen hinzufügen zu können.

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 aktualisier 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
2.2	Verwendete Pakete	4
2.2.1	Firebase	4
2.2.2	AngularJS Material Design	4
2.2.3	Compass	5
2.2.4	Yeoman	5
2.2.5	ngDialog?	5
2.2.6	fullcalender	5
<b>3</b>	<b>Projektvorbereitung</b>	<b>7</b>
3.1	Zielsetzung	7
3.2	Umsetzung	7
3.2.1	Backend	8
3.2.2	Frontend	8
3.2.3	Funktionen	8
<b>4</b>	<b>Entwicklung des Prototypen</b>	<b>11</b>
4.1	Projektstruktur	11
4.1.1	Projekt kickstarten	11
4.1.2	Grunt Tasks	12
4.2	Frontend	13
4.2.1	Login	13
4.2.2	Hauptseite	14
4.2.3	Kunde	15
4.2.4	Objekt	18
4.2.5	Buchung	19
4.2.6	Kalender	21

4.3	Backend .....	21
4.3.1	Echtzeit-Datenbank .....	21
4.3.2	Authentifizierung .....	23
4.3.3	Weitere Firebase-Funktionen .....	24
<b>5</b>	<b>Fazit</b> .....	<b>25</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 verwendeten Technologien, Frameworks und im Projekt verwendeten Pakete dar.

### 2.1 Verwendete Technologien

#### 2.1.1 AngularJS

AngularJS ist ein von Google entwickeltes JavaScript-Framework für client-seitige Webanwendungen. Es funktioniert nach dem Model-View-View-Model Prinzip. Es eignet sich besonders gut für Single-Page-Applications. Dabei werden die meisten Daten beim ersten Aufruf geladen. 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 damit noch ihre Probleme haben. Abgesehen davon verfügt AngularJS 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 in 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 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 Paketverwaltungstool für die clientseitige Webentwicklung. Er ist sogar in Node.js geschrieben und wird über NPM installiert. Es 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 Dateierweiterung `.scss` kann die Erweiterung genutzt werden. Da die Browser aktuell keine `.scss`-Dateien unterstützen, muss das SASS-Kommandozeilentool den Code in `.css`-Dateien übersetzen. Andersherum ist es ohne Probleme möglich CSS-Code in SASS-Dateien zu kopieren und zu nutzen. Dieser wird korrekt umgewandelt. Zu den Vorteilen von Sass gehört zum Beispiel Verschachtelungen, Variablen, Mixins, Vererbung und Importe.

## 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 von Google übernommen. Es stellt eine universelle App-Plattform für Entwickler und Marketer zur Verfügung. Es dient zur Entwicklung von hoch-qualifizierten Anwendungen. 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 das 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**

Compass baut mit der Spracherweiterung Sass 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 Yeoman**

Yeoman ist ein Kommandozeilenprogramm für das Erstellen von Grundgerüsten für Webanwendungen mit ausgewählten Bibliotheken und Frameworks. So wurde die Grundstruktur der Clientseite mittels Yeoman erstellt. Beim Anlegen konnten verschiedene installierbare Werkzeuge ausgewählt werden. Darunter Bootstrap mit SASS für die grafische Oberfläche und der responsiven Gestaltung.

### **2.2.5 ngDialog?**

### **2.2.6 fullcalender**



## 3 Projektvorbereitung

Durch die in der Einleitung erwähnte Problemstellung ergeben sich folgende Zielsetzung und Umsetzung.

### 3.1 Zielsetzung

Prototypische Entwicklung einer Software zur Verwaltung von Buchungen für Ferienhäuser und Wohnungen. Eine der Anforderungen an die Software war die Nutzung auf verschiedenen Endgeräten. Das hat zur Folge, dass auch die Betriebssysteme variieren können. Daraus ergibt sich, dass die Anwendung plattformunabhängig nutzbar sein muss.

Für diesen Fall eignet sich eine Webanwendung besonders gut. Sie ist von jedem Gerät aus nutzbar und benötigt in der Regel nicht mehr als einen Internetbrowser. Auch die Installation eines extra Programmes erübrigt sich. Allerdings ist eine stetige Internetverbindung Voraussetzung. In Anbetracht der weiteren Bedingungen an die Software ist 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, das die Daten speichert.

Weitere Anforderungen an die Anwendungen sind die Verwaltung von Kunden, der zu vermietbaren Ferienhäuser und Wohnungen sowie die eigentliche Buchung. Das Ganze soll übersichtlich dargestellt werden und einfach und schnell benutzbar sein.

### 3.2 Umsetzung

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 Backend

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

### 3.2.2 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 Materialize gestaltet werden.

### 3.2.3 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 Buchung selbst. Folgende Informationen sollten für jeden Kunden zur eindeutigen Identifizierung gespeichert werden:

- Firma (falls vorhanden)
- Vor- und Nachname
- Strasse und Hausnummer
- Postleitzahl und Stadt
- Telefonnummer
- Geburtstag
- Feld für Zusatzinformationen

Die zu vermietenden Ferienhäuser und Wohnungen sollten folgenden Attribute haben:

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

Um eine Buchung zu erstellen sollte die folgenden Informationen festgehalten werden:

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

---

Um eine bessere Übersicht über die Buchungen zu bekommen, sollen diese optisch in einem Kalender dargestellt werden.



## 4 Entwicklung des Prototypen

Im folgenden wird anhand der verschiedener Bereiche die Funktionsweise des Programmes erläutert.

### 4.1 Projektstruktur

#### 4.1.1 Projekt kickstarten

Zu Beginn der Entwicklung war es nötig, eine Basis für die Anwendung zu haben. 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. `angularist` 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 vorzutreiben.

### 4.1.2 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.

#### **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. *Kompillieren von Sass – Dateien zu CSS – Dateien* aus.

#### **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. Zu seinen Aufgaben gehört das Kompillieren von Sass zu CSS, Zusammenführen von CSS- und Javascript-Dateien, Minimieren von Javascript *dazukannnocheinÜglyfier" hinzugeschaltetwerden, dernebenderMinimierungdesJavascript-Codesdiesenauchunleserlichmacht*, 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.

## 4.2 Frontend

Der für den Nutzer sichtbare Bereich.

### 4.2.1 Login

Da die Software wie bereits 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. Um die Software nutzen zu können, benötigt der Bearbeiter eine gültige Kombination aus E-Mail und Passwort. Schon bei der Eingabe der Daten wird überprüft, ob die Felder leer sind oder es sich dabei um eine E-Mail handelt oder nicht. Dabei wird darauf hingewiesen sobald das AT -Zeichen oder der Punkt fehlt.

Sobald das Formular abgeschickt wurde, werden die Daten zunächst im `login.js` Controller entgegengenommen. Von da aus werden die Daten an Authenticate-Service weitergegeben. Dort werden diesen an das Backend geschickt. In Firebase werden diese dann mit den hinterlegten Daten verglichen. Sind die Angaben nicht korrekt wird eine Fehlermeldung zurückgegeben die der Controller im View anzeigt. Sofern alles richtig ist, wird der Nutzer weiter zu der Hauptseite geleitet.

Da es sich hierbei um eine Betriebsinterne Software handelt wird eine Registrierung nicht benötigt. Die Emailadresse und das Passwort werden Manuell im Backend hinzugefügt, bearbeitet oder entfernt.

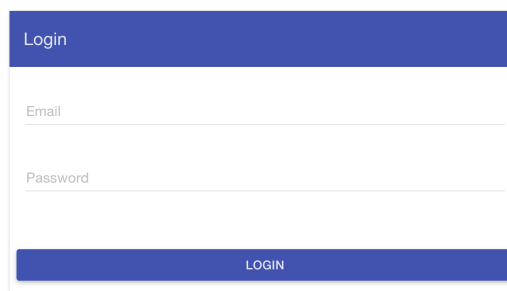


Abbildung 4.1  
Anmeldeformular

## 4.2.2 Hauptseite

Die Hauptseite ist sehr schlicht aufgebaut und teilt sich auf in zwei Teile.

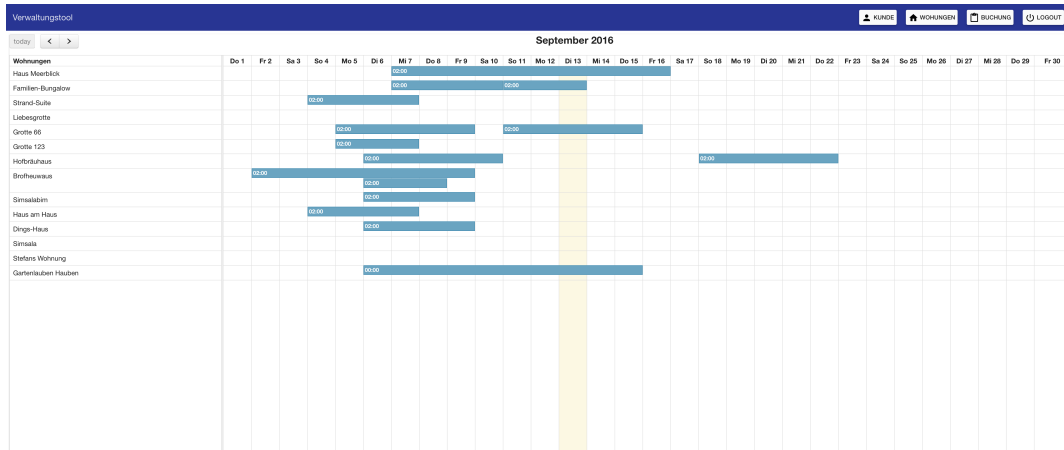


Abbildung 4.2  
Hauptseite

### Header

Im Header befinden sich neben dem Programmlogo auch folgende vier Button:

#### Kunde

Ein Dialog öffnet sich in dem Kunden hinzugefügt, bearbeitet oder gelöscht werden können.

#### 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 oder die Größe eines durchschnittlichen Tablets erreicht hat, werden die Buttons ausgeblendet. Stattdessen erscheint ein Symbol das bei Klick eine Seitennavigation einblendet in der sich alle Buttons befinden.

Wohnungen	Do 1	Fr 2	Sa 3
Haus Meerblick			
Familien-Bungalow			
Strand-Suite			
Liebesgrotte			
Grotte 66			
Grotte 123			
Hofbräuhaus			
Brofheuwäus			
Simsalabim			
Haus am Haus			
Dings-Haus			
Simsala			
Stefans Wohnung			
Gartenlauben Hauben			
test name			

Abbildung 4.3  
Ansicht auf Mobilgeräten

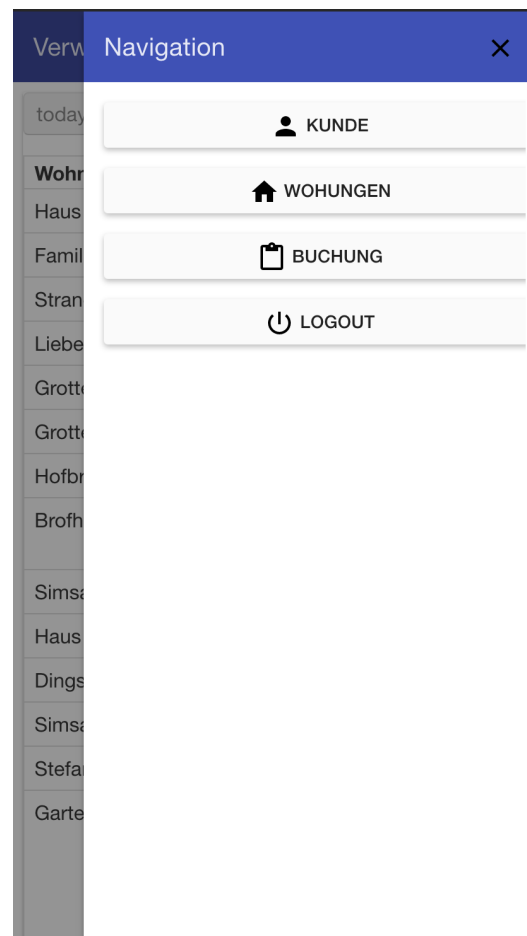


Abbildung 4.4  
Seitennavigation auf Mobil

## Main

In der Main befindet sich lediglich der Kalender, welcher auf voller Breite und Höhe abgebildet. Dieser ist aufgeteilt 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 angezeigt. Die Buchungen werden als Blöcke angezeigt und verbinden dessen Tage in Spalten.

## 4.2.3 Kunde

Zusammen mit dem Objekt bildet der Kunde eine Buchung. Für die Kundeninformationen wurden alle Felder aus dem Projektzielen implementiert. Zudem sind alle Felder ausgenommen der Zusatzinformationen und der Firma Pflicht

und werden beim Abschicken des Formulars überprüft, ob sie leer sind. Um die Eingabe des Geburtstages zu vereinfachen wurde ein für Tag, Monat und Jahr jeweils ein Dropdown-Menü bereitgestellt. Da das Mindestalter für Buchungen 18 Jahre ist, ist das höchste auszuwählende Jahr immer 18 Jahre vom aktuellen gerechnet. Das Datum wird 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 einem `customer` JSON-Objekt gespeichert. Diese werden dem `Customer` Model übergeben und das Dialog geschlossen.

#### Listing 4.1

Beispielhafte Antwort auf eine Registrierungsanfrage

```

1  $scope.submit = function () {
2      $scope.birthday = new Date($scope.
        yearOfBirth + '-' + $scope.
        monthOfBirth + '-' + $scope.dayOfBirth
        ).getTime();
3      var customer = new Customer({
4          Company: $scope.company,
5          FirstName: $scope.firstname,
6          LastName: $scope.lastname,
7          BirthDate: $scope.birthday,
8          Street: $scope.street,
9          City: $scope.city,
10         ZipCode: $scope.zipcode,
11         Email: $scope.email,
12         Phone: $scope.phone,
13         Custom: $scope.custom,
14         Id: $scope.customerID || undefined
15     });
16     Customers.upsert(customer);
17     $scope.hide();
18     $rootScope.$broadcast('showToast');
19 };
```

Soll ein bestehender Kunde bearbeitet werden muss dieser zunächst über das Suchfeld gesucht werden. Mittels des Lupensymbol im Kopfbereich des Dialog wird die Suchleiste 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 die Liste aller Kunden im Model angefordert. Dieses Liefert ein Array mit allen Kundenobjekten. Bei einer hohen Anzahl an Kunden kann sich die Suche schwierig gestalten. Aus diesem Grund kann der Kunde durch eintippen von Buchstaben gefiltert werden. Jeder weitere Buchstabe schränkt die Suche nach dem Vornamen ein und es

werden nur Kunden angezeigt dessen Nachname die eingegebene Zeichenkette beinhalten.

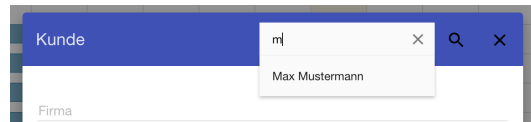


Abbildung 4.5  
Kundensuche

Abbildung 4.6  
Neuen Kunden erstellen

Abbildung 4.7  
Kunden bearbeiten

Wurde ein Kunde ausgewählt, werden alle Daten aus dem JSON-Objekt ausgelesen und in die Input Felder eingefügt. Gleichzeitig wird ein Button angezeigt der das Löschen eines Kunden aus der Datenbank ermöglicht. Um einem Fehler vorzubeugen muss zusätzlich noch ein Dialog bestätigt werden. Wird dieser bestätigt 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 bestätigt werden. Dabei wird die selbe Funktion aufgerufen. In

diesem Fall wird im Controller die `upsert` Funktion im Model aufgerufen. Je nachdem, ob es sich dabei um einen bestehenden Eintrag in der Datenbank handelt oder der Eintrag bereits vorhanden ist wird ein `Update` oder `Insert` durchgeführt. Dafür ist die von der Datenbank vergebene UUID ausschlaggebend.

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. Für den Controller ist das ein und die selbe Funktion. Dieser ruft lediglich die `hide` Methode des Dialogs auf und das Dialogfenster wird ausgeblendet. Dabei werden auch alle Daten aus den Feldern entfernt.

#### 4.2.4 Objekt

Als Objekt wird eine Ferienwohnung oder Haus bezeichnet. Wie auch beim Kunden können Objekte verschiedene Informationen gespeichert werden. Die beiden gewünschten Felder aus den Projektzielen wurden übernommen. Dazu gehört die Eingabe des Namens in ein Textfeld. Wird das Feld fokussiert und ohne Eingabe verlassen erscheint eine Fehlermeldung mit dem Hinweis das Feld auszufüllen. Wird dieser Hinweis ignoriert und der Nutzer versucht das Formular abzuschicken, überprüft der View, ob alle als Pflicht gekennzeichneten Felder ausgefüllt wurden und der `submit` Button gedrückt wurde. Ist eines dieser Angaben `false` wird das Formular erst gar nicht abgeschickt.

Abbildung 4.8  
Objekt erstellen

Abbildung 4.9  
Objekt bearbeiten

**Abbildung 4.10**  
Löschen Bestätigungsdialog

Auch bei der Angabe der maximalen Personenanzahl wurde darauf geachtet Falscheingaben abzufangen. Standardmäßig ist das minimum von 1 eingestellt. Der Nutzer hat die Möglichkeit mit Hilfe von Pfeilen den Wert zu Verändern. Die Maximale Anzahl von Personen ist 10 für jedes Objekt. Da es auch möglich ist Manuell eine Zahl einzugeben, wird ebenfalls überprüft, ob sich die Zahl in der Range 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 Funktion aufgerufen in der die Daten aus den Feldern in ein JSON-Objekt gespeichert werden. Dieses wird der `upsert` Methode des Models übergeben.

Um ein bereits existierendes Objekt bearbeiten zu können muss dies anders als bei den Kunden in der Auflistung in der main ausgewählt werden. 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 des Namens und der Personenanzahl wird die ID des Objektes gespeichert. Ist, wie bei einer Änderung, bereits eine UUID vorhanden, wird diese als Wert gespeichert. Wird eine Objekt neu hinzugefügt ist der Wert `undefined`. Diese Information erlaubt wird in der `upsert` Methode des Models abgefragt und dementsprechend ein neues Objekt hinzugefügt oder ein bestehendes aktualisiert.

### 4.2.5 Buchung

Eine Buchung ist ein Zusammenschluss aus Kunde, Objekt, Personenanzahl, Startdatum und Enddatum. Wie bei der Bearbeitung von Kunden, kann der Nutzer den Kunden über ein Suchfeld filtern und Auswählen. Zusätzlich funktioniert das ebenfalls für das Objekt. Wenn eine neue Buchung hinzugefügt wird, wird zunächst die Auswahl der Personenanzahl ausgeblendet. Erst wenn ein Objekt im Suchfeld ausgewählt wurde, wird die Auswahl in form eines



Dropdown-Menüs im View angezeigt. Im Controller wird vorher 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 zur Verfügung gestellt, der es dem Nutzer erlaubt das Datum durch einfache Auswahl in einem Kalender festzulegen. Standardmäßig ist das aktuelle Datum eingestellt.

Abbildung 4.11  
Objekt erstellen

Abbildung 4.12  
Objekt Fehlereingabe

Um eine Buchung bearbeiten zu können muss diese anders als beim Kunden oder Objekt zunächst in der Buchungsübersicht gesucht werden. Wurde die passende Buchung gefunden und ausgewählt, öffnet sich das Modal mit den hinterlegten Daten. Hier wird die Personenauswahl angezeigt und mit dem eingetragenen Wert vordefiniert. Auch hier besteht die Möglichkeit durch den angezeigten Löschen-Button die Buchung aus der Datenbank zu entfernen.

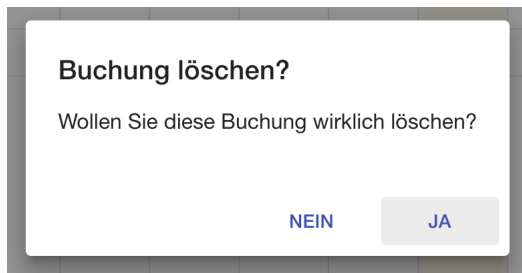


Abbildung 4.13  
Objekt erstellen



Abbildung 4.14  
Objekt Fehlereingabe

Wird ein Dialog bestätigt wird erscheint für drei Sekunden am oberen rechten Fensterrand ein Nachricht. 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.

#### 4.2.6 Kalender

### 4.3 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.

#### 4.3.1 Echtzeit-Datenbank

Bei der von Firebase verwendeten Datenbank handelt es sich um eine sog. NoSQL-Datenbank. Im Gegensatz zu anderen Datenbank-Systemen wie bspw. MySQL, erfolgt die Abspeicherung der Daten dokumentenbasiert im JSON-Format. Dokumentenbasiert bedeutet, dass einzelne Einträge der Datenbank als Objekte abgelegt werden. Dabei sind diese Objekte 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.

## Organisation

Die Benennung der Gruppen von Datensätzen ist von Firebase sehr flexibel gestaltet und ermöglicht beliebige Namen. Wie bereits in dem Kapitel Models erläutert, existieren die folgenden Gruppen:

- bookings
- customers
- resources

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 einzigartige 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 Objekten aufgebaut werden. Behandelt werden diese Schlüssel wie Zeichenketten und erfahren somit keiner speziellen Behandlung sofern sie als Werte eines Objektes abgelegt sind.

## 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 eingeloggte Benutzer. Auf die Authentifizierung-Möglichkeiten wird in dem Kapitel Authentifizierung eingegangen.

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

### Listing 4.2

Beispielhafte Antwort auf eine Registrierungsanfrage

```

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 eingeloggte 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.

## 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 Collections genauer erläutert.

## 4.3.2 Authentifizierung

Bei fast jeder zu entwickelnden Anwendung steht das Thema Nutzer-Authentifizierung auf dem Plan. Standardmäßig erfolgen Implementierungen, die Benutzer mit Hilfe von Email/Benutzernamen und Passwort authentifizieren. Leider kann es hierbei immer wieder zu Fehlern in der Entwicklung oder im Design kommen, sodass es u.U. 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 Email-Passwort-Verfahren aktiviert. Zudem bietet Firebase auch die Authentifizierung mithilfe 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 Email-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.

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

### **4.3.3 Weitere Firebase-Funktionen**

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

## **5 Fazit**

EXAMPLE:

# Abbildungsverzeichnis

<b>4</b>	<b>Entwicklung des Prototypen</b>	
4.1	Anmeldeformular .....	13
4.2	Hauptseite .....	14
4.3	Ansicht auf Mobilgeräten .....	15
4.4	Seitennavigation auf Mobil .....	15
4.5	Kundensuche .....	17
4.6	Neuen Kunden erstellen .....	17
4.7	Kunden bearbeiten .....	17
4.8	Objekt erstellen .....	18
4.9	Objekt bearbeiten .....	18
4.10	Löschen Bestätigungsdialog .....	19
4.11	Objekt erstellen .....	20
4.12	Objekt Fehlereingabe .....	20
4.13	Objekt erstellen .....	21
4.14	Objekt Fehlereingabe .....	21

# **Tabellenverzeichnis**



# Listings

<b>4 Entwicklung des Prototypen</b>	
4.1 Beispielhafte Antwort auf eine Registrierungsanfrage .....	16
4.2 Beispielhafte Antwort auf eine Registrierungsanfrage .....	22





# Glossar

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

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

**Database Management System** System, welches die Datenbank aufbaut und verwaltet. Ziel ist ein möglichst effizientes und einfach zu bedienendes System..

**DBMS** *siehe* Database Management System.

**Extensible Markup Language** Kompaktes Datenformat, in etwa vergleichbar mit JSON..

**Graphical User Interface** Frontend-Ansicht, welche dem Benutzer eines Programms ausgeliefert wird..

**GUI** *siehe* Graphical User Interface.

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

**HTTP** *siehe* Hypertext Transfer Protocol.

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

**Hypertext Transfer Protocol** Protokoll zur Übertragung von Daten über ein Rechnernetz..

**JAVAFX** Framework zum Aufbau von einfachen bis komplexen Java-Frontendsystemen..

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

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

**Plain Old Java Object** Ein Java-Objekt im herkömmlichen Sinne..

**POJO** *siehe* Plain Old Java Object.

**QR** *siehe* Quick Response Code.

**Quick Response Code** Zweidimensionaler Code, der schnell durch Maschinen gelesen werden kann..

**SQL** *siehe* Structured Query Language.

**Structured Query Language** Datenbanksprache, um durch Datenbanken zu navigieren, Bearbeitungen, Löschvorgänge und Neueintragungen vorzunehmen..

**TOKEN** Methode zur Autorisierung von Software-Diensten..

**UI** *siehe* User Interface.

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

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

**User Interface** Siehe Graphical User Interface.

**XML** *siehe* Extensible Markup Language.