

Entwicklung einer Hotelbuchungssoftware

Development of hotel booking software

Alexander Falkenberg, Hasan Alhelal

Teamprojekt

Betreuer: Titel Vorname Nachname

Ort, DD.MM.YYYY

---

## Kurzfassung

In der Kurzfassung soll in kurzer und prägnanter Weise der wesentliche Inhalt der Arbeit beschrieben werden. Dazu zählen vor allem eine kurze Aufgabenbeschreibung, der Lösungsansatz sowie die wesentlichen Ergebnisse der Arbeit. Ein häufiger Fehler für die Kurzfassung ist, dass lediglich die Aufgabenbeschreibung (d.h. das Problem) in Kurzform vorgelegt wird. Die Kurzfassung soll aber die gesamte Arbeit widerspiegeln. Deshalb sind vor allem die erzielten Ergebnisse darzustellen. Die Kurzfassung soll etwa eine halbe bis ganze DIN-A4-Seite umfassen.

Hinweis: Schreiben Sie die Kurzfassung am Ende der Arbeit, denn eventuell ist Ihnen beim Schreiben erst vollends klar geworden, was das Wesentliche der Arbeit ist bzw. welche Schwerpunkte Sie bei der Arbeit gesetzt haben. Andernfalls laufen Sie Gefahr, dass die Kurzfassung nicht zum Rest der Arbeit passt.

---

## **Abstract**

The same in English.

---

# Inhaltsverzeichnis

<b>1 Einleitung</b> .....	1
1.1 Motivation .....	1
1.2 Ziele der Arbeit .....	1
<b>2 Grundlagen</b> .....	2
2.1 HTML, CSS, JS .....	2
2.1.1 HTML .....	2
2.1.2 Less .....	2
2.1.3 JavaScript .....	2
2.2 Node.js .....	3
2.2.1 Express .....	4
2.2.2 Nodemailer .....	4
2.2.3 Formidable .....	4
2.3 MongoDB .....	5
<b>3 Konzept</b> .....	6
<b>4 Implementierung</b> .....	8
4.1 Buchungsdialog .....	8
4.2 Meine Buchung .....	8
4.3 Buchung API .....	9
4.3.1 Erstellen einer Buchung .....	9
4.3.2 Ändern einer Buchung .....	10
4.3.3 Abfragen einer Buchung .....	11
4.3.4 Löschen einer Buchung .....	11
4.4 Bewerbungsformular .....	11
4.5 Dreidimensionale Zimmer .....	12
<b>5 Beispiele</b> .....	14
<b>6 Anwendungsszenarien</b> .....	19
<b>7 Zusammenfassung und Ausblick</b> .....	20

**Literaturverzeichnis . . . . .** 21

**Selbstständigkeitserklärung . . . . .** 22

---

## **Abbildungsverzeichnis**

3.1	Grobe Architektur des Systems .....	6
3.2	Sitemap .....	6
3.3	Datenbankentwurf .....	7
5.1	Schritt 1 .....	14
5.2	Schritt 2 .....	15
5.3	Schritt 3 .....	15
5.4	Schritt 4 .....	16
5.5	Schritt 5 .....	16
5.6	Schritt 6 .....	17
5.7	Schritt 7 .....	17
5.8	Schritt 8 .....	18

# **1**

---

## **Einleitung**

### **1.1 Motivation**

Diese Arbeit handelt von der Umsetzung eines Buchungssystems für ein Hotel. Ein solches System ist besonders wichtig um den allgemeinen Aufwand beim Verwalten von verfügbaren Zimmern deutlich zu verringern und der Fehleranfälligkeit beim Vergeben von eben diesen vorzubeugen. Unternehmen profitieren von solchen Software-Lösungen, da Mitarbeiter entlastet werden und Kunden leichter eine Buchung tätigen können. Außerdem kann diese Software in leicht abgeänderter Form auch problemlos für andere Anwendungen oder Unternehmen mit anderen Produkten oder Dienstleistungen benutzt werden.

### **1.2 Ziele der Arbeit**

Ziel dieser Arbeit ist es ein simples Buchungssystem zu schaffen dass sich ohne großen Arbeitsaufwand verwenden lässt. Es soll den Nutzern des Systems möglich sein ohne die Erstellung eines Nutzerprofils Buchungen tätigen und diese verwalten zu können. Die Informationen über jede Buchung werden den Nutzern anschließend per Email zugesandt. Ein Nutzer soll außerdem die Möglichkeit haben alle Infos sowie 3D-Modelle der Zimmer abrufen und Bewerbungen über die Anwendung versenden können.

## **2**

---

# **Grundlagen**

## **2.1 HTML, CSS, JS**

### **2.1.1 HTML**

HTML-Dokumente bilden das Grundgerüst einer Webanwendung und sind von daher essenziell. Ein solches Dokument besteht aus einzelnen, oft schon vorgefertigten Elementen die ineinander geschachtelt werden können. Diese können anschließend durch referenzierte „Stylesheets“ (im Folgenden anhand von Less) in ihrem Aussehen verändert oder durch JavaScript mit weiteren Funktionalitäten ausgestattet werden.

### **2.1.2 Less**

In dieser Anwendung wird die „Stylesheet-Sprache“ Less verwendet, um die verschiedenen HTML Elemente zu gestalten. Innerhalb des Less-Codes kann normaler CSS-Code verwendet werden, bietet jedoch einige Vorteile im Vergleich zu dieser herkömmlichen Stylesheet-Sprache wie zum Beispiel Variablen, Funktionen und Verschachtelung. Für die endgültige Nutzung dieser Sprache wird eine Kompilierung des Codes zu CSS-Code benötigt. Less bietet außerdem die Möglichkeit den Code auf mehrere Dateien aufzuteilen und diese durch die Kompilierung zu einer CSS-Datei zu bündeln.

### **2.1.3 JavaScript**

Die Programmiersprache JavaScript mit ihren etlichen verfügbaren Modulen kann dafür verwendet werden Webanwendungen dynamisch zu gestalten. Zentraler Bestandteil dafür ist die „DOM-Manipulation“, durch die dynamisch Elemente in einem Webdokument verändert, hinzugefügt und entfernt werden können. Außerdem kann mithilfe von JavaScript eine HTTP-Anfrage erstellt, versendet und auf diese reagiert werden um mit einer serverseitigen Anwendung kommunizieren zu können. Insbesondere in dieser Anwendung wird von dieser Möglichkeit Gebrauch gemacht indem „XMLHttpRequest“-Objekte mit allen dazugehörigen Informationen initialisiert werden. Innerhalb dieser Objekte kann angegeben werden wie auf eine Antwort reagiert werden soll. Eine solche Initialisierung kann wie folgt aussehen:

```
1  const request = new XMLHttpRequest();
2  request.addEventListener('load', () => {
3      console.log(request.response);
4  });
5
6  request.open('POST', '/order');
7  request.setRequestHeader('Accept', 'application/json');
8  request.setRequestHeader('Content-Type', 'application/json');
9  request.responseType = 'json';
10 request.send(data);
```

## Three.js

Werden dreidimensionale Darstellungen von Objekten oder Räumen benötigt, kann das Three.js Modul verwendet werden. Zu jedem Three.js-Projekt gehören folgende drei Bestandteile:

1. scene (engl. für Szene)
2. camera (engl. für Kamera)
3. renderer (engl. für Renderer)

Alle Objekte die gerendert werden sollen müssen sich in einer Scene befinden. Das Kamera-Objekt ermöglicht, die mithilfe von Three.js erstellten Szenen, zu visualisieren. Es existieren verschiedene Arten von Kamera-Objekten wie z.B. die perspektivische Kamera, die sich wie das menschliche Auge verhält, also näher stehende Objekte größer erscheinen lässt, und die orthografische Kamera, die die Größe der Objekte nicht an die Entfernung anpasst. Der Renderer als Hauptkomponente von Three.js bekommt eine erstellte Szene und Kamera übergeben und erzeugt so die finale dreidimensionale Darstellung des gewünschten Objekts.

Um eine solche Darstellung möglichst realistisch wirken zu lassen müssen geeignete Bilder im „.hdr“-Format ausgewählt werden. Damit Kandidaten dieses Formats verwendet werden können wird ein zusätzliches Modul mit dem Namen „RGEBE-Loader“ benötigt.

Mit den sogenannten „Orbit Controls“ aus dem Three.js-Modul können Nutzer die Ausrichtung der Kamera in der gerenderten Szene per Mausbewegung anpassen und so beispielsweise dreidimensionale Räume erkunden.

## 2.2 Node.js

Die serverseitige Grundlage des Systems bildet die Laufzeitumgebung Node.js. Mit dieser ist es möglich JavaScript Programme abseits eines Browsers ausführen zu können. Mithilfe von „npm“, dem Paketmanager von Node.js, werden außerdem externe Pakete bzw. Module für bestimmte Funktionalitäten verwendet um die Implementierung zu vereinfachen.

### 2.2.1 Express

Das Express-Framework ermöglicht eine unkomplizierte Erstellung einer Webanwendung. So wird beispielsweise die Verarbeitung von HTTP-Anfragen mithilfe von spezifischen Methoden deutlich vereinfacht und macht das Erstellen einer API besonders effizient. Innerhalb der Buchungssoftware sorgt Express für die Bereitstellung eines Servers, der alle clientseitigen Dateien zur Verfügung stellt und auf HTTP-Anfragen zur Erzeugung oder zum Abrufen von Buchungen, reagiert. Besonders wichtig ist das Konzept der Routen. Das Definieren von Routen bestimmt wie auf eine Client-Anfrage an eine bestimmte URI des Servers und eine spezifische HTTP-Methode, geantwortet wird. Die Zuweisung einer Route kann wie folgt aussehen:

```
1     app .METHOD(PATH ,  HANDLER);
```

In dem Beispiel steht `app` exemplarisch für ein express-Objekt, `METHOD` für eine HTTP-Methode, `PATH` für den Serverpfad und `HANDLER` für die Funktion die ausgeführt wird wenn eine Client-Anfrage existiert, die mit der Route übereinstimmt. Die Funktion die als `HANDLER` angegeben wird hat außerdem Zugriff auf ein HTTP-Request und ein HTTP-Response Objekt. Über das Request-Objekt kann auf Informationen der HTTP-Anfrage zugegriffen und mit dem Response-Objekt auf die Anfrage geantwortet werden.

### 2.2.2 Nodemailer

Nodemailer ist ein „npm“-Paket, das dafür verwendet wird E-Mails mit Node.js zu versenden. Durch Angabe der Adresse von der die E-Mail verschickt werden soll und verschiedenen Optionen wie z.B. dem Betreff wird das Senden vereinfacht. Um eine E-Mail zu versenden muss im ersten Schritt ein „Transporter“ zusammen mit dem verwendeten Mail Dienst und den Anmeldeinformationen der zu sendenen E-Mail Adresse, erzeugt werden. Anschließend werden die Mail-Optionen initialisiert und über die Methode „`sendMail`“ des Transportes, die E-Mail versandt.

### 2.2.3 Formidable

Das Node.js-Modul Formidable dient zur serverseitigen Verarbeitung von Formulardaten, insbesondere von Formularen die hochgeladene Dateien beinhalten. Da das Hochladen von Dateien durch einen Server behandelt werden muss bieten sich Module wie Formidable besonders an. So können die hochgeladenen Dateien in das Dateisystem des Servers eingefügt und verändert werden. Es werden viele Einstellungsmöglichkeiten geboten. So kann z.B. die maximale Größe einer hochgeladenen Datei festgelegt werden. Außerdem wird es dadurch möglich diese Dateien beispielsweise an eine E-Mail anzuhängen.

## 2.3 MongoDB

Unter den dokumentenorientierten NoSQL-Datenbanken ist MongoDB die meist verwendete. Die wichtigsten Bestandteile einer MongoDB Datenbank sind Dokumente und Collections. Dokumente sind Datenstrukturen die sich aus einem oder mehreren Paaren, bestehend aus Feldern und dazugehörigen Werten, zusammenfügen. In ihrem Aufbau sind diese Dokumente identisch mit JSON Objekten. Collections hingegen sind Sammlungen von verschiedenen Dokumenten und sind vergleichbar mit Tabellen aus relationalen Datenbanken.

Damit eine Node.js-Anwendung mit einer MongoDB kommunizieren kann wird beispielsweise das „npm“-Paket „mongodb“ benötigt. Hier können dann auf den einzelnen Collections, Methoden angewendet werden. Die wichtigsten Methoden für das zu erstellende System sind „find“, „insertOne“, „deleteOne“ und „deleteMany“. Mithilfe von find werden alle Dokumente einer Collection geliefert, auf die ein in der Methode spezifiziertes Kriterium zutrifft. InsertOne ermöglicht es ein neues Dokument in eine Collection hinzuzufügen und deleteOne bzw. deleteMany ermöglicht das Löschen von Dokumenten aus einer Collection.

# 3

---

## Konzept

Das folgenden Bild beschreibt den groben Aufbau der Anwendung und die Technologien die auf den verschiedenen Ebenen verwendet wird.

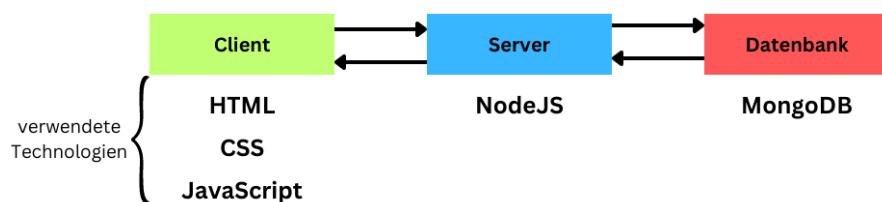


Abbildung 3.1: Grobe Architektur des Systems

Die Sitemap dient dazu die Navigation durch die Anwendung und den Zusammenhang der einzelnen Unterseiten darzustellen.

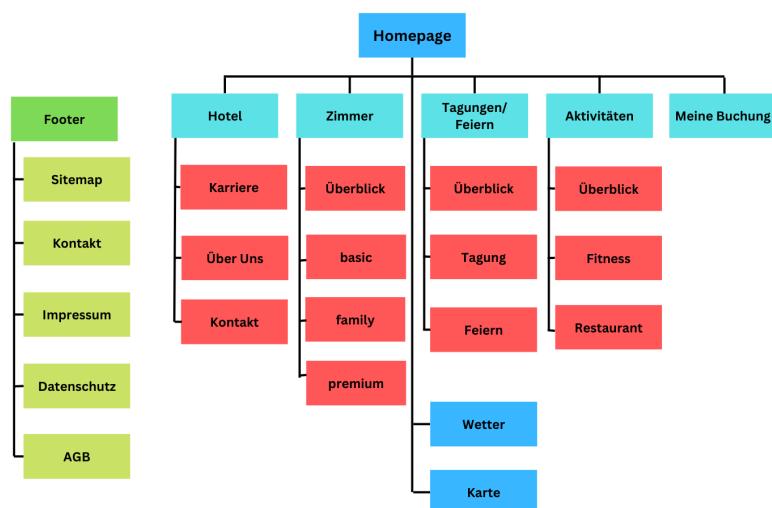


Abbildung 3.2: Sitemap

Diese Abbildung stellt den Aufbau der Datenbank dar, wobei jedes Rechteck eine Collection abbildet mit Dokumenten die jeweils die Ovale als Attribute besitzt.

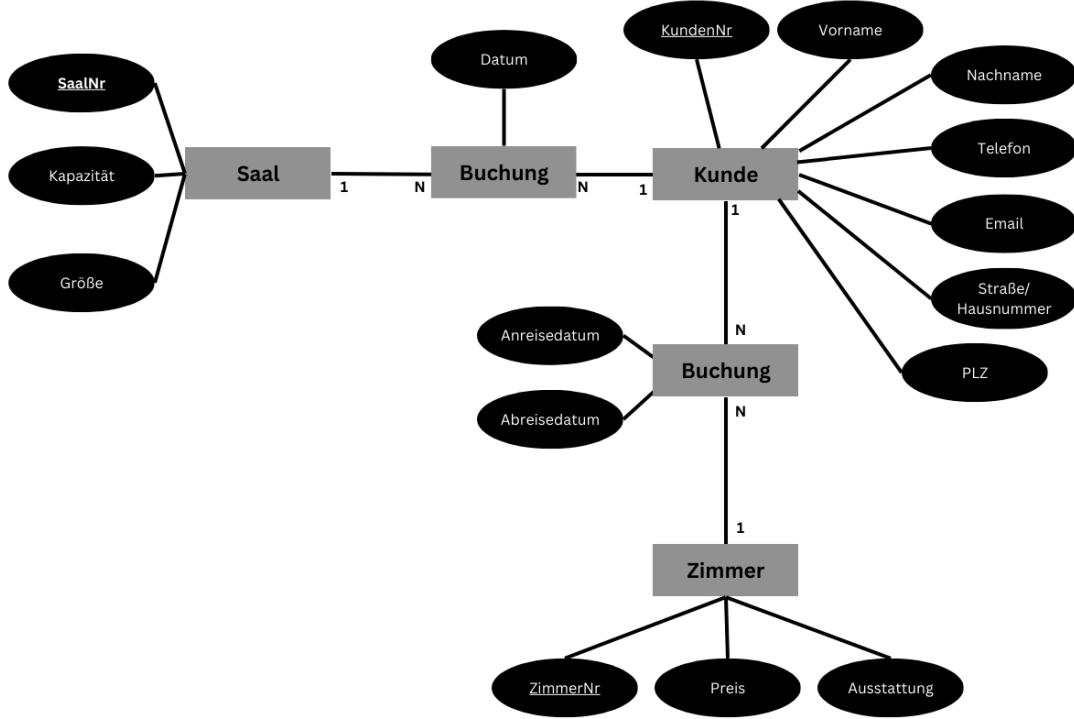


Abbildung 3.3: Datenbankentwurf

Dieser Algorithmus spielt in der finalen Implementierung eine wichtige Rolle, da in vielen Fällen die verfügbaren Zimmer bestimmt werden müssen.

---

**Algorithmus 1:** Algorithmus um alle verfügbaren Räume zu erhalten

---

**Data:** rooms = alle Räume, reservations = alle Buchungen, arrival = Anreisedatum, departure = Abreisedatum

**Result:** available = verfügbare Räume

*notAvailable*  $\leftarrow$  [];

**for** room of rooms **do**

**for** reservation of reservations **do**

**if** (room.id == reservation.room) && !(departure < reservation.arrival || arrival > reservation.departure) **then**  
 | notAvailable.push(room);  
 | break;

---

**return** available = rooms.filter(item => !notAvailable.includes(item))

---

# 4

---

## Implementierung

### 4.1 Buchungsdialog

Dem Client soll es innerhalb der Anwendung ermöglicht werden über einen „Buchungsdialog“ eine Buchung abzuwickeln. Dieser Dialog teilt sich in mehrere Schritte auf. Im ersten Schritt muss der Nutzer ein Formular, bestehend aus Ankunfts-, Abreisedatum, Anzahl der Erwachsenen, Anzahl der Kinder und die Anzahl der Räume, ausfüllen. Will der Nutzer nun mit dem zweiten Schritt fortführen werden zuerst die Eingaben aus dem ersten Schritt auf Gültigkeit geprüft. Gültigkeit heißt in diesem Fall dass beispielsweise das Ankunftsdatum nicht in der Vergangenheit oder vor dem Abreisedatum liegen darf. Sind die Eingaben gültig wird eine HTTP-Anfrage an den Server geschickt mit dem Ziel alle zu der angegeben Aufenthaltszeit verfügbaren Zimmer als HTTP-Antwort zu erhalten. Im zweiten Schritt des Dialogs kann der Nutzer aus allen verfügbaren Zimmern die passenden für seinen Aufenthalt wählen. Bevor mit dem dritten Schritt fortführt wird, muss geprüft werden das genau so viele Zimmer, wie im ersten Schritt angegeben, ausgewählt wurden. Ist dies der Fall wird im nächsten Schritt mit einem Formular für alle persönlichen Daten des Nutzers fortgesetzt. Werden alle Felder des Formulars ordnungsgemäß ausgefüllt kann mit dem vierten und letzten Schritt fortführt werden. In diesem wird dem Nutzer eine Übersicht der angegebenen Daten bzw. der Buchung angezeigt und bietet dem Nutzer über ein Textfeld die Möglichkeit Extrawünsche an das Hotel zu äußern. Will der Nutzer die Buchung abschließen wird eine HTTP-POST-Anfrage an den Server, mit allen wichtigen Daten zu Erstellung der Buchung, versendet.

### 4.2 Meine Buchung

In der Webanwendung wird einem Nutzer unter dem Punkt „Meine Buchung“ ermöglicht Informationen über getätigte Buchungen abzufragen und diese Anzupassen. Hierfür muss in ein dafür bestimmtes Feld die Buchungsnummer des Clients eingegeben werden. Anschließend wird eine HTTP-GET-Anfrage mit dem Pfad „/order/:id“ an den Server gesendet, auf die der Server mit allen Informationen zu der Buchung antwortet, wenn die Buchungsnummer existiert. Im Anschluss wird

dem Nutzer eine Übersicht mit den wichtigsten Informationen dargeboten und zusätzlich die Möglichkeit die Buchung zu stornieren oder die Reisedaten abzuändern.

Entscheidet sich der Client dazu die Buchung zu stornieren wird einem Fenster-Alert gefragt ob sich der Nutzer mit der Stornierung sicher ist. Bestätigt der Client die Löschung wird eine HTTP-DELETE-Anfrage mit dem Pfad „/order/:id“ an den Server gesendet. Ist die Stornierung erfolgreich wird eine Bestätigungs-E-Mail an den Nutzer gesendet.

Wird sich jedoch für die Änderung der Daten entschieden wird in einem neuen Fenster ein Formular zur Eingabe der neuen Reisedaten angezeigt. Werden dort die neuen Daten eingegeben und fortgefahren, sendet der Client eine HTTP-POST-Anfrage mit dem Pfad „/order/dates“ an den Server versendet.

## 4.3 Buchung API

Serverseitig wird eine API erstellt, die die clientseitige Kommunikation mit dem Server ermöglicht. Dafür wird ein express-Router mit verschiedenen Routen benötigt, die unter anderem das Erstellen, Löschen, Verändern und Abfragen von Buchungen abwickeln. Ein solcher Router wird mit express-Methode „express.Router()“ erstellt. Für jede Route wird es außerdem von Nöten sein, Zugriff auf die MongoDB-Datenbank zu besitzen. Mithilfe des MongoDB-Moduls wird dafür ein „MongoClient“ erzeugt, der die Adresse der MongoDB-Datenbank übergeben bekommt.

### 4.3.1 Erstellen einer Buchung

Für die Erstellung einer Buchung wird dem Router eine Route hinzugefügt, die auf einen HTTP-POST mit dem Pfad „/order“ reagiert. Innerhalb der Funktion die die Anfrage behandelt werden zu aller erst alle für die Buchung benötigten Informationen aus dem Request-Objekt in Konstanten gespeichert. Zu den benötigten Informationen gehören: Vorname, Nachname, E-Mail-Adresse, Telefonnummer, Wohnort (mit Ort, Straße, Hausnummer und PLZ), Ankunftsdatum, Abreisedatum und die Art von Zimmern die gebucht werden sollen. Anschließend wird geprüft ob die Konstanten gültig sind. Sollte eine nicht zulässig sein wird eine HTTP-Antwort mit dem Code 400 an den Client gesendet. Besteht jedoch alle Konstanten die Prüfung, wird mit der Verbindung des MongoClients fortgeführt. Der Aufruf der „connect“- und daraufhin der „db“-Methode (mit dem Namen der gesuchten Datenbank) des bereits erstellten MongoClients, liefert ein Datenbank-Objekt mit dem nun einzelne Collections abgerufen werden können.

Um einen neuen Nutzer anzulegen muss ein Collection-Objekt zur Anpassung der Collection die alle Nutzer enthält, mithilfe der „collection“-Methode des Datenbank-Objekts, erzeugt werden. Das Collection-Objekt erlaubt es mit der

Methode „insertOne(elemToInsert)“ den neuen Nutzer mit allen dazugehörigen Informationen aus den gespeicherten Konstanten (Vorname, Nachname, Adresse, E-Mail, Telefonnummer) in die Collection einzufügen.

Im weiteren Verlauf werden die Collection mit allen Buchungen und die Collection mit allen Zimmern benötigt. Werden auf beiden Collections jeweils die Methoden „find().toArray()“ ohne Parameter aufgerufen, können alle Buchungen und Zimmer als Feld in jeweiligen Variablen abgelegt werden. Mithilfe des Algorithmus in 1 werden alle zu der angegebenen Ankunfts- und Abreisezeit verfügbaren Zimmer geliefert. Sind genug Zimmer mit den vom Client angegebenen Zimmertypen nicht vergeben werden für jeden zu buchenden Raum eine neue Buchung in die Buchungs-Collection eingefügt. Dabei besteht jede Buchung aus der ID des neuen Nutzers, der ID des verfügbaren Raums, des Ankunfts- und Abreisedatums.

Zuletzt wird per nodemailer eine Mail an die E-Mail-Adresse des Clients gesendet. Diese enthält die ID des angelegten Nutzers, welche später zu Änderung oder Stornierung der Buchung verwendet werden kann. Als Antwort erhält der Client eine HTTP-Antwort mit dem Status-Code 200 insofern keine Komplikationen auftreten.

#### 4.3.2 Ändern einer Buchung

Um einem Nutzer zu ermöglichen die Daten einer bereits getätigten Buchung zu ändern, wird eine Route hinzugefügt, die auf einen HTTP-POST mit dem Pfad „/order/dates“ reagiert. Dafür müssen im ersten Schritt die Variablen mit den Werten aus dem HTTP-Body initialisiert und anschließend auf Gültigkeit überprüft werden. Die Werte bestehen aus der ID des Nutzers, das neue Ankunfts- und Abreisedatum. Sollte einer der Werte ungültig sein, wird eine HTTP-Antwort mit dem Status-Code 400 an den Client versendet. Sind sie jedoch gültig verbindet sich der MongoClient mit der Datenbank und fordert alle Buchungen an. Anschließend werden mithilfe der find-Methode einerseits alle Buchungen die mit der ID des Nutzers übereinstimmen und andererseits alle anderen Buchungen, in zwei verschiedenen Konstanten gespeichert. Aus den Raum-IDs der Buchungen des Nutzers werden alle Räume aus der Raum-Collection die diese IDs besitzen, in einem Feld gesammelt. Dadurch ist es möglich die in der Buchung gebuchten Zimmertypen in Variablen zu speichern, damit bei der Änderung der Daten auch wieder dieselben Zimmertypen gewählt werden. Wie schon in Kapitel 6.1.1 erklärt müssen auch hier alle Zimmer die während des neuen Ankunftsdatum bis zum neuen Abreisedatum verfügbar sind, ermittelt werden.

Sollte es nicht genügend Zimmer mit denselben Zimmertypen wie in der ursprünglichen Buchung geben, die während der neuen Daten verfügbar sind wird die bestehende Buchung nicht geändert und eine HTTP-Antwort mit dem Status-Code 400 an den Client gesendet. Sollte es jedoch genügend Zimmer geben wird die bisherigen Buchungen aus der „reservation“-Collection mithilfe der „deleteMany“-

Methode gelöscht und anschließend neue Buchungen der gleichen Zimmertypen zu anderen Zeiten in diese Collection eingefügt. Ist der Ablauf problemlos wird der Status-Code 200 an den Client gesendet.

### 4.3.3 Abfragen einer Buchung

Um einem Nutzer zu ermöglichen die wichtigsten Daten wie beispielsweise das Ankunftsdatum, Abreisedatum, Anzahl der Nächte, der Gesamtpreis und die Anzahl der Räume einer bereits getätigten Buchung abzurufen, wird eine Route hinzugefügt, die auf einen HTTP-GET mit dem Pfad „/order/:id“ reagiert. Im ersten Schritt wird die ID der gesuchten Buchung aus den Parametern der HTTP-Anfrage in einer Konstante gespeichert und geprüft ob diese eine gültige MongoDB-ObjectId ist. Wenn die ID zulässig ist werden alle Reservierungen dessen Nutzer-ID mit dieser übereinstimmt in der MongoDB abgerufen. Mithilfe der Buchungen lassen sich das Ankunfts-, Abreisedatum und folglich auch die Anzahl der Nächte ermitteln. Außerdem werden wie schon in 6.1.1 und 6.1.2 alle Räume in Form eines Feldes benötigt. Der Gesamtpreis der Buchung wird dann wie folgt berechnet:

```

1   for (const room of rooms) {
2       for (const reservation of reservations) {
3           if (room._id.equals(reservation.room)) {
4               totalPrice += room.price;
5           }
6       }
7   }
8   totalPrice *= totalNights;

```

Sind alle Daten zusammengetragen werden diese als JSON mit dem Status-Code 200 an den Client versendet.

### 4.3.4 Löschen einer Buchung

Für das Löschen einer Buchung wird eine Route benötigt, die auf ein HTTP-DELETE mit dem Pfad „/order/:id“ reagiert. Zuerst wird die ID der zu löschenen Buchung aus den Parametern der Anfrage auf Gültigkeit geprüft. Anschließend wird der Nutzer der diese ID besitzt, mithilfe der „deleteOne“-Methode der Nutzer-Collection aus dieser entfernt. Außerdem müssen alle Buchungen die diese ID besitzen durch die „deleteMany“-Methode aus der Buchungscollection entfernt werden. Treten keine Probleme bei der Löschung auf wird eine HTTP-Antwort mit dem Status-Code 200 an den Client gesendet.

## 4.4 Bewerbungsformular

Unter dem Punkt Karriere ist es innerhalb der Anwendung möglich über ein eigens dafür erstelltes Formular eine Bewerbung an den Betreiber der Webanwendung zu senden. Hier soll es unter anderem ermöglicht werden, den Lebenslauf und die Bewerbung als PDF hochzuladen. Beim Absenden des Formulars wird die

HTTP-Methode POST mit dem Pfad „“ verwendet. Serverseitig wird eine passende express-Route definiert, in welcher mithilfe von Formidable der Dateien-Upload gewährleistet wird.

Im ersten Schritt wird in der express-Route das eingehende Formular durch „const form = new formidable.IncomingForm()“ in einer Konstante gespeichert. Anschließend können die Attribute des Formulars zugewiesen werden. In diesem Fall wird festgelegt das mehrere Dateien akzeptiert werden, das jede Datei maximal 5MB groß sein darf und in welchem Ordner die Dateien abgelegt werden. Danach wird folgende Methode aufgerufen: „form.parse(request, CALLBACK)“, wobei CALLBACK eine Funktion mit allen Feldern und Dateien des Formulars als Parameter ist. In dieser Funktion wird unter anderem definiert wie mit einem Error umgegangen werden soll. Sollte kein Error auftreten wird mit dem speichern der hochgeladenen Dateien fortgefahrene. Dafür wird im ersten Schritt der Dateien-Parameter in eine JSON umgewandelt und alle Schlüssel dieser in einem Feld gespeichert. In einer for-Schleife wird dann durch alle Schlüssel iteriert und unter anderem geprüft ob jede Datei größer als 0MB ist ein gültiges Format besitzt. Sollte die Größe einer der Dateien, 0MB entsprechen wird diese aus dem Dateisystem entfernt und Dateien im falschen Formate sorgen dafür das der Server dem Client mit dem Status-Code 400 antwortet. Ist die Datei gültig wird sie umbenannt und der neue Pfad in einem neuen Feld gespeichert. Anschließend werden die Dateien aus dem neuen Feld per nodemailer an eine E-Mail angehängt und dem Betreiber der Webanwendung gesendet.

## 4.5 Dreidimensionale Zimmer

Zunächst wird ein Canvas-Element in der HTML Datei benötigt. Diese Element bleibt leer und dient nur als Zeichenfläche für die mit JavaScript bzw. Three.js erstellten Objekte. Als nächstes wird ein Renderer verwendet, der das Ergebnis am Ende in dem Canvas-Element renderet, dazu wird ein Instanz des Renderes wie folgt erzeugt: renderer = new THREE.WebGLRenderer(). Des Weiteren werden zwei weitere Objekte erzeugt, das Szeneobjekt let scene = new THREE.Scene() und das Kameraobjekt let camera = new THREE.PerspectiveCamera(). Diese werden miteinander verbunden in dem sie als Argumente an die Methode renderer.render(scene, camera) übergeben wird. Allerdings müsste diese Methode nach jedem gerenderten Frame aufgerufen werden. Damit es automatisch geschieht eignet sich eine gesonderte namens animate(), in der die vorher erwähnte Funktion einmal aufgerufen wird an den renderer wie folgt zu übergeben: renderer.setAnimationLoop(animate).

Das Kameraobjekt hat folgende Parameter:

1. **Field Of View** (engl. **Sichtfeld**), hier liegt ein passender Wert zwischen 40-80 Grad.
2. **Aspect ratio** (engl. **Seitenverhältnis**), mit `window.innerWidth / window.innerHeight` wird die gerenderte Szene in Vollbild Modus angezeigt und somit die komplette Fläche des Bildschirmes genutzt.
3. **Near** (engl. **Nahe**), hier eignet sich ein geringer Wert zwischen 0.1-1, denn jeder Wert darunter wird abgeschnitten. Dieser Parameter besitzt keine Einheit.
4. **Far** (engl. **Weit**), hier eignet sich ein hoher Wert wie 1000, denn jeder höhere Wert wird abgeschnitten. Dieser Parameter besitzt keine Einheit.

Das Kameraobjekt befindet standardmäßig an der Position  $x = 0, y = 0, z = 0$ . Diese lässt sich aber beliebig mit der Methode „`camera.position.set()`“ ändern. Die Instanz der OrbitControls: `orbit = new OrbitControls(camera, renderer.domElement)`, ermöglicht den Nutzern die Kamera beliebig zu drehen. Das Orbitobjekt nimmt ein Kameraobjekt und `renderer.domElement` als Parameter. Damit die Position der Kamera nach einer Mausbewegung aktualisiert wird, wird die Methode `orbit.update()` aufgerufen.

Die im Projekt verwendeten .hdr-Bilder bieten realistische Beleuchtung, Schatten und Spiegelung. Von daher war es nötig eine Instanz des RGBELoaders: `loader = new RGBELoader()`, zu erzeugen. Die Methode `loader.load` nimmt zwei Parameter, der relative Pfad des .hdr-Bilds und eine Rückruffunktion, die eine Textur als Argument hat. Die Textur bzw. das Bild wird dann als Hintergrund der Szene festgelegt.

## 5

---

### Beispiele

Im Folgenden wird eine beispielhafte Interaktion eines Nutzers mit dem System anhand von Bildschirmabzügen erläutert. In diesem Szenario will der fiktive Nutzer eine Buchung für Zimmer in dem Grandline-Hotel tätigen. Im ersten Schritt besucht der Nutzer die Startseite und klickt auf den „buchen“-Knopf (siehe Abb.5.1).

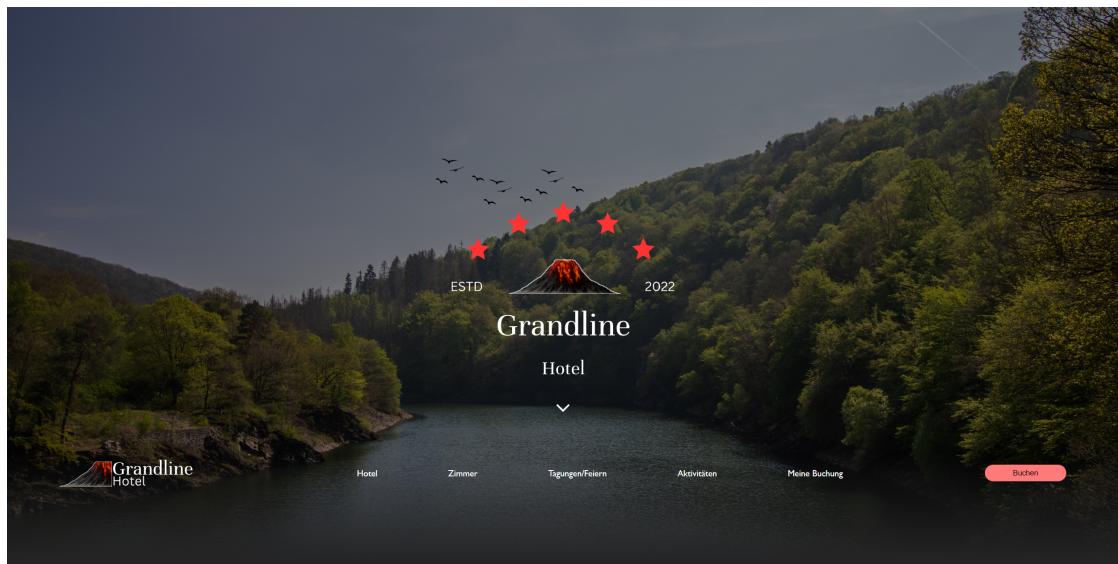


Abbildung 5.1: Schritt 1

Als Folge des Klicks auf den „buchen“-Knopf kommt ein Formular (siehe Abb.5.2) hervor in welchem der Nutzer die Reisedaten und Anzahl der Zimmer für die gewünschte Buchung angeben kann.

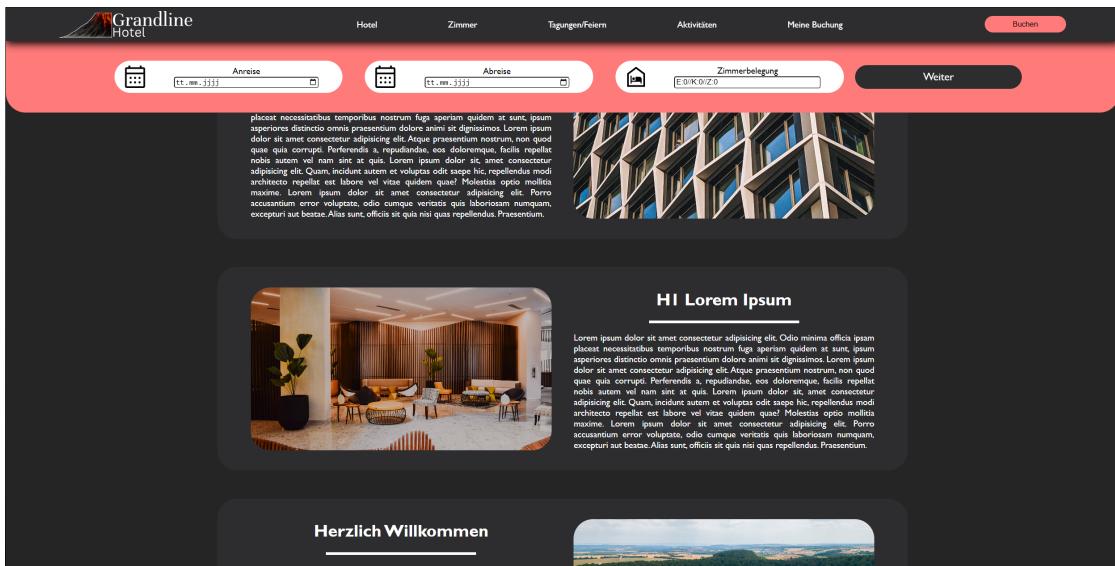


Abbildung 5.2: Schritt 2

In dem 3.Schritt tätigt der Nutzer seine Angaben in das Formular und setzt durch einen Klick auf den „weiter“-Knopf mit der Buchung fort (siehe Abb.5.3).

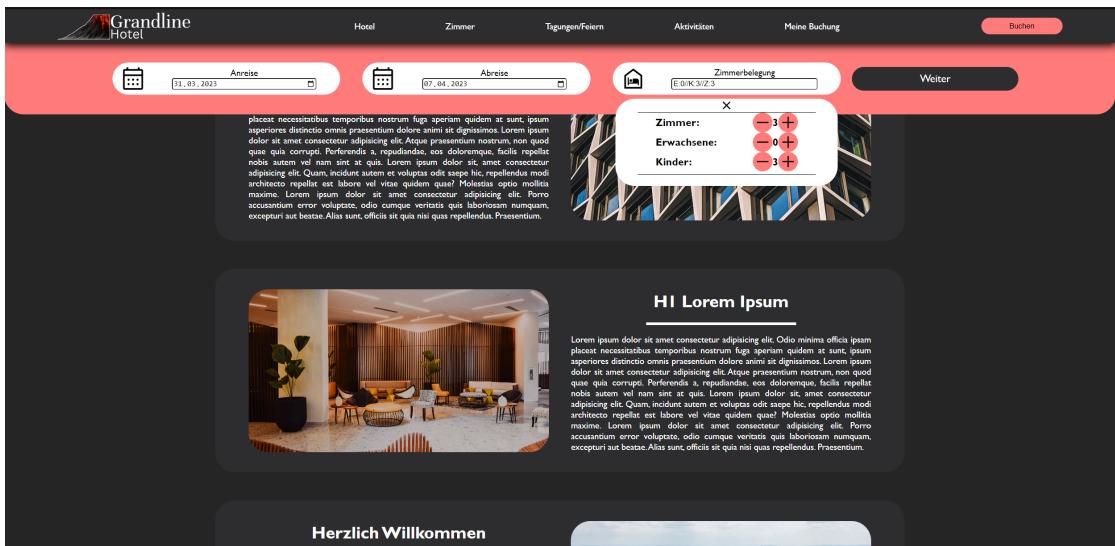


Abbildung 5.3: Schritt 3

Der Nutzer wird nun in den Buchungsdialog weitergeleitet in dem er eine Auswahl der verfügbaren Zimmer dargeboten bekommt (siehe Abb.5.4).

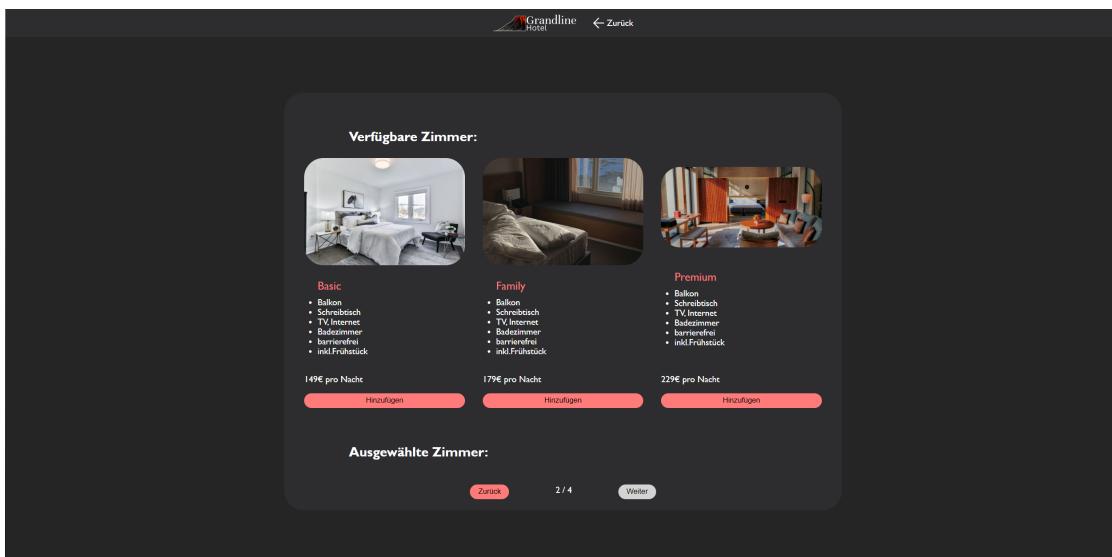


Abbildung 5.4: Schritt 4

Der Nutzer wählt im 5. Schritt aus den verfügbaren Zimmern die Gewünschten aus und betätigt die „Weiter“-Taste (siehe Abb.5.5). Die Menge die der Nutzer auswählen muss entspricht der angegebenen Anzahl der Zimmer aus Schritt 3.

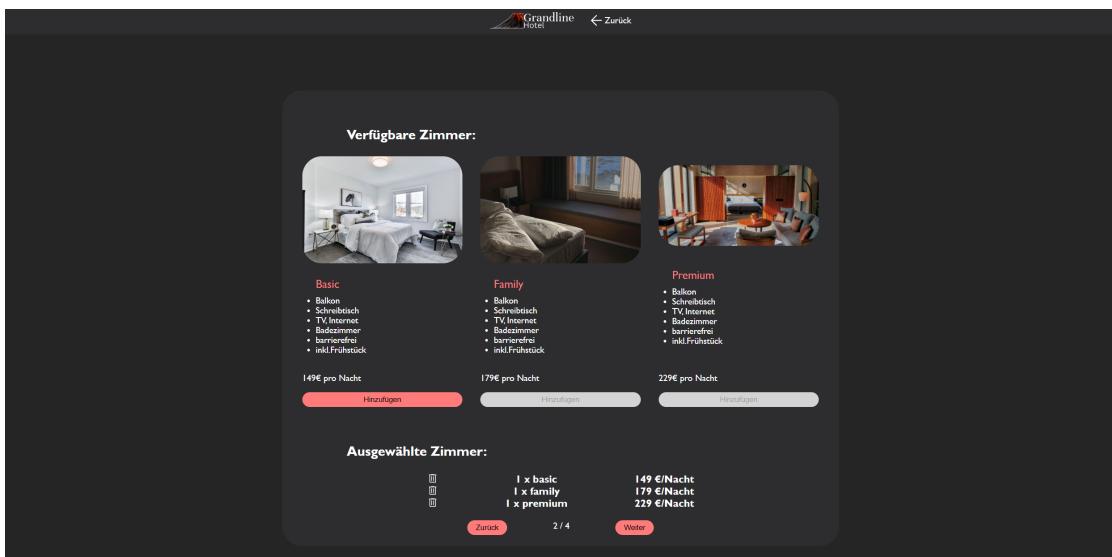


Abbildung 5.5: Schritt 5

In das nun erschienene Formular trägt der Nutzer seine persönlichen Daten ein und fährt über den „Weiter“-Knopf mit dem Buchungsdialog fort (siehe Abb.5.6).

The screenshot shows a dark-themed web form titled "Persönliche Daten". It contains fields for gender (selected as "mannlich"), first name ("Max"), last name ("Mustermann"), email ("max.mustermann@gmail.com"), phone number ("012345678"), street ("Musterstraße"), house number ("5"), and zip code ("12345"). At the bottom of the form are two buttons: "Zurück" (Back) and "Weiter" (Next), with the text "3 / 4" indicating this is the third step of four.

Abbildung 5.6: Schritt 6

Im siebten Schritt wird dem Nutzer eine Übersicht über die zu tätigeende Buchung dargeboten und bietet dem Nutzer vor dem Abschluss die Möglichkeit noch Extrawünsche und Informationen über ein Textfeld an das Hotel weiterzugeben (siehe Abb.5.7). Klickt der Nutzer anschließend auf „Abschließen“, wird der Buchungsprozess abgeschlossen.

The screenshot shows a dark-themed web form. At the top, it displays the user's personal data: Max Mustermann, Musterstraße 5, 12345 Muster, Telefon: 012345678, Email: max.mustermann@gmail.com. Below this is a section titled "Ausgewählte Zimmer" showing three room types selected: "basic" (1 x basic), "family" (1 x family), and "premium" (1 x premium). A section for "Amerkungen und Extras:" contains a placeholder text field: "Hier können Sie Amerkungen/Extra äußern:". Below this, the total amount is displayed as "Gesamtsumme für 7 Nächte in 3 Zimmer: 3899€". At the bottom, there is a note: "Zahle den genannten Betrag beim Einchecken". The form includes "Zurück" and "Abschließen" buttons, with the text "4 / 4" indicating this is the final step.

Abbildung 5.7: Schritt 7

Nach Abschluss der Buchung wird der Nutzer darüber informiert, dass eine E-Mail mit den wichtigsten Informationen an den Nutzer versendet wurde (siehe Abb.5.8).

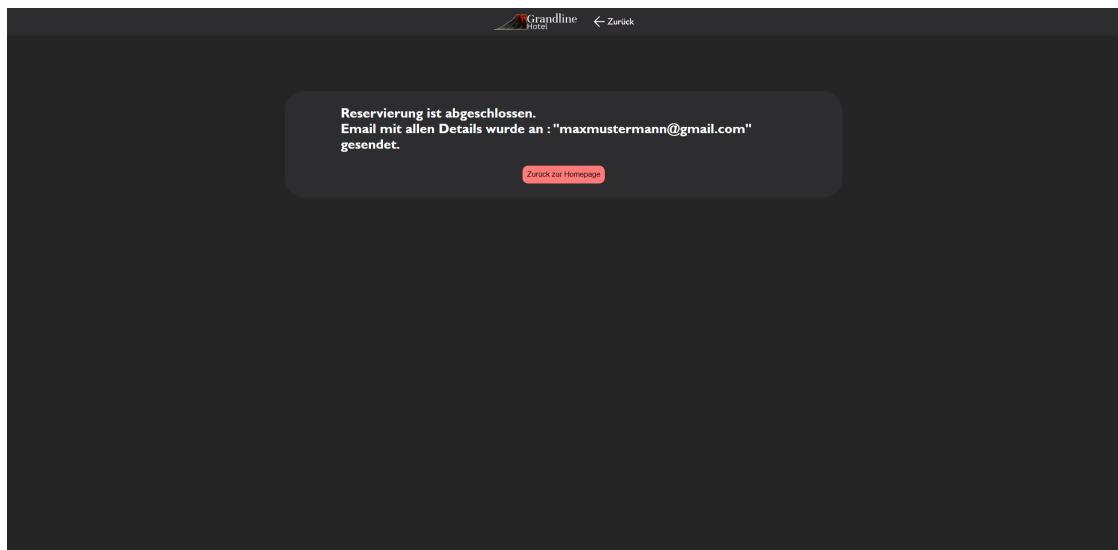


Abbildung 5.8: Schritt 8

# 6

---

## Anwendungsszenarien

## Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit noch einmal kurz zusammengefasst werden. Insbesondere sollen die wesentlichen Ergebnisse Ihrer Arbeit herausgehoben werden. Erfahrungen, die z.B. Benutzer mit der Mensch-Maschine-Schnittstelle gemacht haben oder Ergebnisse von Leistungsmessungen sollen an dieser Stelle präsentiert werden. Sie können in diesem Kapitel auch die Ergebnisse oder das Arbeitsumfeld Ihrer Arbeit kritisch bewerten. Wünschenswerte Erweiterungen sollen als Hinweise auf weiterführende Arbeiten erwähnt werden.

---

## Literaturverzeichnis

- CDK02. COULOURIS, GEORGE, JEAN DOLLIMORE und TIM KINDBERG: *Verteilte Systeme: Konzepte und Design*. Addison Wesley, München, 3. Auflage, 2002.
- Che85. CHERITON, DAVID R.: *Preliminary Thoughts on Problem-oriented Shared Memory: A Decentralized Approach to Distributed Systems*. ACM SIGOPS Operating Systems Review, 19(4):26–33, 1985.
- CLR90. CORMEN, THOMAS H., CHARLES E. LEISERSON und RONALD L. RIVEST: *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.
- GO03. GOTTWALD, TIM und RAINER OECHSLE: *DisASTER (Distributed Algorithms Simulation Terrain): A Platform for the Implementation of Distributed Algorithms*. Diplomarbeit, Fachhochschule Trier, Fachbereich Informatik, 2003.
- Mal97. MALTE, PETER: *Replikation in Mobile Computing*. Seminar 31/1997, Universität Karlsruhe, Institut für Telematik, 1997.
- Mos93. MOSBERGER, DAVID: *Memory Consistency Models*. Technical Report 93/11, University of Arizona, Department of Computer Science, November 1993.
- Nan03. NANNEN, VOLKER: *The Paradox of Overfitting*. Master-Abschlussarbeit, Rijksuniversiteit Groningen, 2003.
- Wik. Wikipedia: *Programmausdruck*. <https://de.wikipedia.org/wiki/Programmausdruck>. Abgerufen am 13.10.2021.

# A

---

## Selbstständigkeitserklärung

- Diese Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.
- Diese Arbeit wurde als Gruppenarbeit angefertigt. Meinen Anteil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser:

Meine eigene Leistung ist:

---

Datum

---

Unterschrift der Kandidatin/des Kandidaten