Modul 223 – Abschluss Projekt

David Bischof

Bild von: https://bencrump.com/faqs/what-qualifies-as-pain-and-suffering/

1 Einleitung

1.1 Motivation

Dieses Projekt ist das Abschlussprojekt im Modul 223 entstanden und es war ein Leiden. ChatGPT hat mir bei manchem Programmieren und beim Korrigieren und Strukturieren des Textes geholfen. Geschrieben habe ich ihn selst

1.2 Ausgangslage

Ein Coworking Space in der Agglomeration von Zürich möchte in Zukunft seine Mitglieder und die Nutzung des Angebots digital über eine Webapplikation erfassen. Dazu sollte zuerst ein minimaler Prototyp realisiert werden, um den Kunden besser abholen zu können.

1.3 Aufgabe

Der zu entwickelnder Prototyp soll aus einer Server- und Client-Applikation bestehen. Die Client-Applikation benutzt die Server-Applikation über eine HTTP API. Für den Prototyp sind folgende, menschliche Akteure vorgesehen:

- Administrator
- Mitglied
- Besucher (nicht authentifizierter Benutzer)

Folgende funktionalen Anwendungsfälle sollen mindestens im Prototypen implementiert werden:

- Als Besucher möchte ich mich mit meinem Vor- und Nachnamen, meiner E-Mail-Adresse und einem Passwort registrieren, damit ich die Rolle Mitglied bekommen kann.
- Als Besucher möchte ich mich mit meiner E-Mail-Adresse und meinem Passwort anmelden, damit ich mich als Mitglied oder Administrator authentifizieren kann.
- Als Mitglied möchte ich halbe und ganze Tage an bestimmten Daten im Coworking Space als Buchung anfragen, damit ich die Angebote des Coworking Space nutzen kann.
- Als Mitglied möchte ich den Status meiner Buchungen überprüfen, damit ich erfahre, ob meine Buchung bestätigt oder abgelehnt wurde.
- Als Mitglied kann ich meine zukünftigen Buchungen stornieren, damit ich auf Veränderungen in meiner Terminplanung reagieren kann.
- Als Administrator kann ich Mitglieder verwalten (erstellen, bearbeiten, löschen), damit ich die Mitglieder organisieren kann.
- Als Administrator kann ich Buchungsanfragen akzeptieren und ablehnen, damit die Mitglieder das Angebot des Coworking Space nutzen können. ›
- Als Administrator kann ich Buchungen verwalten (erstellen, bearbeiten, löschen), damit ich die Buchungen organisieren kann.

Folgende nicht-funktionale Anforderungen sollen mindestens im Prototypen umgesetzt werden:

- Das Datenmodell erfüllt die erste, zweite und dritte Normalform nach der relationalen Entwurfstheorie. › Der erste Besucher bekommt nach der Registrierung die Rolle Administrator anstatt Mitglied.
- Die Authentifizierung erfolgt mittels JSON Web Token (JWT nach RFC 7519) über den HTTP Header 'Authorization'. ›
- Das JWT läuft 24 Stunden nach der Ausstellung ab und verliert seine Gültigkeit.
- Das JWT wird clientseitig während dessen Lebensdauer persistent aufbewahrt.

2 Anforderungen Analysieren

2.1 Erweiterte Anforderungen

2.1.1 Drei zusätzliche, einzigartige, funktionale Anforderungen (User Stories)

1. Als Administrator, kann ich Räume und Arbeitsplätze verwalten, damit ich die Nutzung optimieren kann.
    - Der Administrator soll die Möglichkeit haben, Räume und Arbeitsplätze hinzuzufügen, zu bearbeiten und zu löschen, um die Effizienz der Nutzung im Coworking Space zu verbessern. (CRUD)
2. Als Mitglied, kann ich meinen aktuellen Standort teilen, damit ich andere Mitglieder im Coworking Space leicht finden kann.
    - Mitglieder sollen die Option haben, ihren aktuellen Standort in der Webapplikation zu teilen, um die Interaktion und Zusammenarbeit mit anderen Coworkern zu fördern.
3. Als Besucher, kann ich eine virtuelle Tour durch den Coworking Space machen, um die Einrichtungen vor der Registrierung kennenzulernen.
    - Potenzielle Besucher sollen die Möglichkeit haben, eine virtuelle Tour durch den Coworking Space zu unternehmen, um einen Eindruck von den verfügbaren Einrichtungen zu bekommen und ihre Entscheidung zur Registrierung zu unterstützen.

2.1.2 Drei zusätzliche, einzigartige, nicht-funktionale Anforderungen (messbar beschrieben)

1. Die Webapplikation soll eine Ladezeit von unter 3 Sekunden für alle Seiten aufweisen, gemessen mit Werkzeugen wie Google PageSpeed Insights.
    - Die Ladezeit für alle Seiten der Webapplikation wird regelmäßig mit Werkzeugen wie Google PageSpeed Insights überprüft, um sicherzustellen, dass sie unter dem Ziel von 3 Sekunden liegt.
2. Die Verfügbarkeit der Webapplikation soll 99,9% pro Monat erreichen, gemessen durch kontinuierliches Monitoring mit einem Dienst wie Pingdom.
    - Die Verfügbarkeit der Webapplikation wird kontinuierlich mit einem Monitoring-Dienst wie Pingdom überwacht, um sicherzustellen, dass sie mindestens 99,9% Verfügbarkeit pro Monat erreicht.
3. Die Benutzeroberfläche der Webapplikation soll responsive sein und auf verschiedenen Geräten (Desktops, Tablets, Smartphones) gleichmäßig gut funktionieren.
    - Die Benutzeroberfläche wird auf verschiedenen Geräten getestet und die Funktionalität wird durch Benutzerfeedback sowie durch interne Tests gewährleistet.

2.1.3 Einordnung der zusätzlichen Anforderungen

Die zusätzlichen Anforderungen wurden speziell auf die Bedürfnisse eines Coworking Spaces zugeschnitten, um die Nutzererfahrung zu verbessern, die Effizienz der Ressourcennutzung zu steigern und die technische Leistung der Anwendung sicherzustellen.

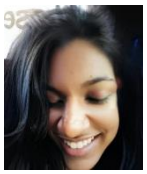
2.2 Personans

2.2.1 Administrator Persona

Name: Lakshana Patel

Alter: 27 Jahre

Geschlecht: Weiblich



Lakshana ist die Betriebsleiterin des Coworking Spaces. Sie hat einen Hintergrund als Sekretärin und Managed nun die täglichen Abläufe im Coworking Spaces.

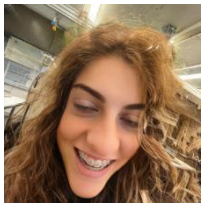
Lakshana nutzt den Coworking Space, um die Bedürfnisse der Mitglieder zu verstehen und sicherzustellen, dass der Betrieb reibungslos läuft. Sie ist verantwortlich für die Mitgliederverwaltung, die Organisation von Veranstaltungen und die Kommunikation mit den Mitgliedern.

2.2.2 Mitglied Persona

Name: Léonita Bürki

Alter: 25 Jahre

Geschlecht: Weiblich



Sarah ist freiberufliche Grafikdesignerin. Sie arbeitet an verschiedenen Projekten für Kunden aus verschiedenen Branchen und benötigt einen flexiblen Arbeitsplatz.

Sarah nutzt den Coworking Space, um Zugang zu professioneller Infrastruktur zu haben, wie schnelle Internetverbindung, Konferenzräume und Netzwerkmöglichkeiten mit anderen Freiberuflern. Sie schätzt die Flexibilität, halbe oder ganze Tage zu buchen, je nach ihren Projektanforderungen.

2.2.3 Besucher Persona

Name: Sonjo Homeless

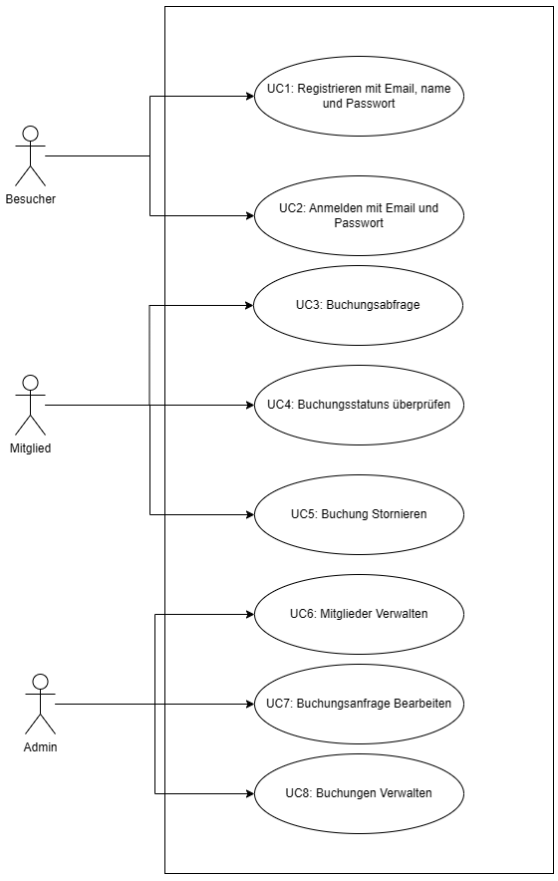
Alter: 69 Jahre

Geschlecht: Männlich

Sonjo ist Softwareentwickler und arbeitet derzeit remote für ein Tech-Startup. Er ist neu in der Stadt und erkundet die lokalen Arbeitsmöglichkeiten.

Sonjo sucht nach einem produktiven Arbeitsumfeld, um Ablenkungen zu Hause zu vermeiden (seine Frau Wasabi). Er möchte die Angebote des Coworking Spaces ausprobieren, bevor er sich für eine Mitgliedschaft entscheidet. Zudem interessiert er sich für die Community-Events und Networking-Möglichkeiten

2.3 UseCases

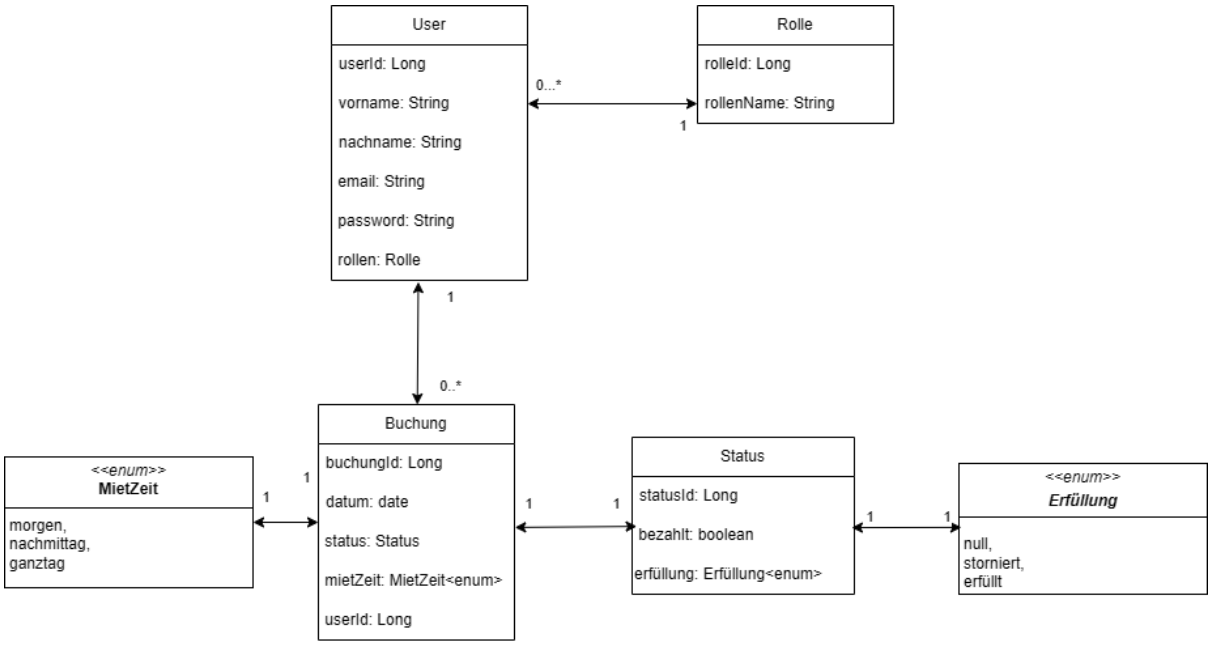


Die Usecases behandeln alle Themen, welche vom Projekt vorgegeben wurden.

Der Admin kann alles, was ein Mitglied kann, und ein Mitglied kann alles was ein Besucher auch schon kann. => Desto höher die Rolle, desto mehr Funktionalität.

3 Persistenzschicht Planen

3.1 Fachklassendiagramm



- User - Rolle:
  - Ein User hat eine Rolle (Many-to-One Beziehung).
  - Die Verbindung wird über «rollen: Rolle» hergestellt.
- Buchung - User:
  - Eine Buchung gehört zu einem User (Many-to-One Beziehung).
  - Die Verbindung wird über die «userId: Long» hergestellt.
- Buchung - Status:
  - Eine Buchung hat einen Status (One-to-One Beziehung).
  - Die Verbindung wird über den «status: Status» hergestellt.

4 Schnittstelle planen

4.1 Schnittstellenplanung

Die angegebenen Variablen, welche in den in den Schnittstellen verwendet werden, sind im Klassendiagramm genauer beschrieben.

4.1.1 Registrierung eines Besuchers

- **Pfad**: /api/register
- **http-Verben**: Post
- **Beschreibung**: Registriert einen neuen Benutzer als Besucher.
- **Parameter**:
  - Body:

```
{
  "vorname": "string",
  "nachname": "string",
  "email": "string",
  "passwort": "string"
}
```

- **Antworten**:
  - **201 Created**: Benutzer erfolgreich registriert
  - **400 Bad Request**: Fehlende oder ungültige Eingabedaten.

4.1.2 Anmeldung eines Benutzers

- **Pfad**: /api/login
- **http-Verben**: POST
- **Beschreibung**: Authentifiziert einen Benutzer mit Rollend gibt ein JWT zurück
- **Parameter**:
  - Body:

```
{
  "email": "string",
  "passwort": "string"
}
```

- **Antworten**:
  - **200 Ok**: Erfolgreiche Authentifizierung, JWT wird zurückgegeben.
  - **401 Unauthorized**: Ungültige Anmeldeinformationen.

4.1.3 Buchung anfragen (Mitglied)

- **Pfad**: /api/bookings
- **http-Verben**: POST
- **Beschreibung**: Ein Mitglied fragt eine Buchung für halbe oder ganze Tage an.
- **Parameter**:
  - Header: Autorisierung mit JWT
  - Body:

```
{
  "datum": Date,
```

```
        "mietZeit": MietZeit<enum>,
        "raumId": Long }
```

- **Antworten:**
  - **201 Created**: Buchungsanfrage erfolgreich erstellt.
  - **400 Bad Request**: Fehlende oder ungültige Eingabedaten.
  - **401 Unauthorized**: Fehlende oder ungültige Authentifizierung.

4.1.4 Buchung überprüfen (Mitglied)

4.1.5 Zukünftige Buchungen stornieren (Mitglied)

- **Pfad**: /api/booking/stornieren
- **http-Verben**: PUT
- **Beschreibung**: Setzt den Status einer zukünftigen Buchung auf «storniert».
- **Parameter**:
  - Header: Autorisation mit JWT
  - Body:

```
{
    "buchungsId": Long
}
```

- **Antworten**:
  - **200 Ok**: Buchung erfolgreich storniert.
  - **404 Not Found**: Buchung nicht gefunden.
  - **401 Unauthorized**: Fehlende oder ungültige Authentifizierung.

4.1.6 Mitglieder verwalten (Administrator)

- **Pfad**: /api/users
- **http-Verben**: GET, POST, PUT, DELETE
- **Beschreibung**: Verwaltung der Mitglieder
- **Parameter**:
  - Header: Autorisierung mit JWT
  - GET
    - Pfad: /api/users/{userId}
    - Oder Get all mit /api/users/

- POST:

```
{
  "vorname": "string",
  "nachname": "string",
  "email": "string",
  "passwort": "string",
  "rollenId": rollenId (Admin oder Mitglied)
}
```

- PUT

```
{
  "userId": Long,
  "vorname": "string",
  "nachname": "string",
  "email": "string",
  "passwort": "string",
  "rollenId": rollenId (Admin oder Mitglied)
}
```

- DELETE
  - Pfad: /api/users/{userId}

- Antworten:
  - **200 Ok:** Anfrage erfolgreich durgeführt
  - **201 Created**: Nur bei POST
  - **404 Not found**: Benutzer nicht gefunden
  - **400 Bad Request**: Ungültig
  - **401 unautorisiert**

4.1.7 Buchungsanfragen akzeptieren/ablehnen (Administrator)

- **Pfad**: /api/booking/{buchungsId/status
- **http-Verben**: PUT
- **Beschreibung**: Akzeptiert oder lehnt eine Buchungsanfrage ab.
- **Parameter**:
  - Header: Autorisierung mit JWT
  - Pfad: {buchungId}
  - Body:

```
{
  "status": "bestätigt" | "abgelehnt"
}
```

- **Antworten**:
  - **200 Ok**: Buchung Akzeptiert
  - **404 Not Found**: Buchung gibt es nicht
  - **400 und 401**

4.1.8 Buchungen verwalten (Administrator)

- **Pfad**: /api/booking
- **http**-Verben: GET, POST, PUT, DELETE
- **Beschreibung**: Verwaltung der Buchungen
- **Parameter**:
  - Header: Autorisierung mit JWT
  - GET:
    - Pfad: /api/bookings/{buchungsId}
  - POST:

```
{
   "userId": Long,
   "datum": Date,
   "mietZeit": MietZeit<enum>,
   "raumId": Long,
   "status": Status
 }
```

  - PUT:

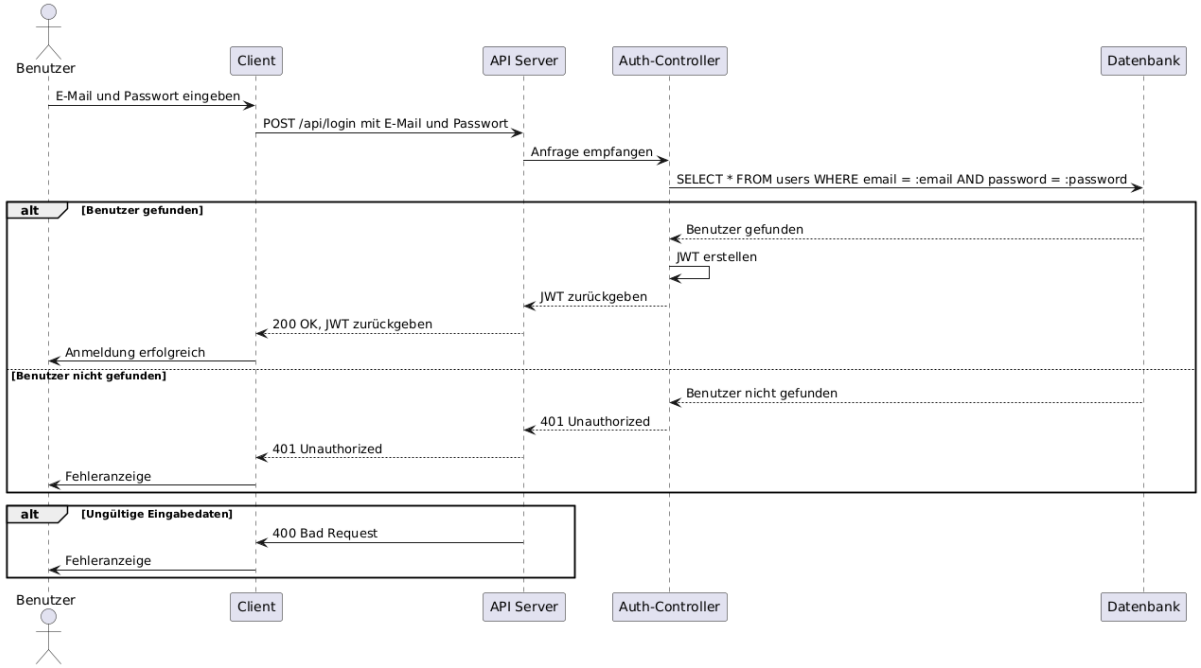
```
 {
   "buchungId": Long,
   "datum": Date,
   "mietZeit": MietZeit<enum>,
   "raumId": Long,
   "status": "angefragt" | "bestätigt" | "abgelehnt"
 }
```

  - DELETE:
    - Pfad: /api/bookings/{buchungsId}

- **Antworten**:
  - **200 Ok**: Anfragen erfolgreich durchgeführt.
  - **201 Created**: Buchung erfolgreich erstellt (für POST).
  - **404 Not Found**: Buchung nicht gefunden.
  - **400 Bad Request**: Fehlende oder ungültige Eingabedaten.
  - **401 Unauthorized**: Fehlende oder ungültige Authentifizierung.

4.2 Sequenzdiagramm

Login Sequenzdiagramm



Das Sequenzdiagramm illustriert den Prozess des Benutzer-Logins in der App.

Erfolgsfall:

- Wenn der Benutzer gefunden wird, erstellt der Auth-Controller ein JWT.
- Der JWT wird an den Client zurückgegeben.
- Der Benutzer erhält eine Erfolgsmeldung.

Fehlerfall:

- Wenn der Benutzer nicht gefunden wird, gibt der Auth-Controller eine 401 Unauthorized zurück.
- Der Benutzer erhält eine Fehlermeldung.

Ungültige Eingabedaten:

- Bei ungültigen Eingabedaten sendet der Server eine 400 Bad Request.
- Der Benutzer wird über die ungültigen Daten informiert.

Quellen:

https://bencrump.com/faqs/what-qualifies-as-pain-and-suffering/

https://app.diagrams.net/

https://plantuml.com/de/

https://chat.openai.com/