

# **Projektgruppe: ss\_2023\_wa\_foodiemate**

**Maris Engel & Hendrik Lendeckel**

## **Installation**

1. git clone [https://git.ai.fh-erfurt.de/ma4163sp1/wa/ss23/ss\\_2023\\_wa\\_foodiemate.git](https://git.ai.fh-erfurt.de/ma4163sp1/wa/ss23/ss_2023_wa_foodiemate.git)
2. Projekt öffnen
3. Im Projekt ein Terminal öffnen und docker compose up ausführen:

```
docker compose up
```

## **Bug-fixing**

→ wenn die Datenbank nicht startet

**Reason:** Die Daten im Database-Ordner werden als inkonsistent erkannt. Erkennbar an WiredTiger-Fehlermeldung. Grund dafür ist, dass die DB im Produktivsystem lief und weitere Daten erzeugt hat. Bei einem erneuten Pull kommt die Datenbank nicht mit den Änderungen zurecht. ##### Solution: Der lokale Database Ordner im Stammverzeichnis muss gelöscht werden: 1. `git status` 2. schauen ob der DB Ordner unversioniert ist (bei nur einem Start ohne Änderungen ist das der Fall) wenn ja: 3a: `rm -rf database` wenn nein: 3b: `git restore --staged database` und dann `git checkout -- database` und danach `rm -rf database` (MacOs) 4. `git pull` Danach funktioniert der Neustart mit den Daten aus dem Git, da der Ordner frisch gezogen wurde.

## **Datenbank Backup**

Pfad: /mongodump Befehl für Sicherung auf System:

1. `docker exec -it ss2023_wa_foodiemate_database /bin/bash`
2. `mongodump --host localhost --port 27017 --db foodiemate --out /backup`

Rücksicherung:

1. `docker exec -it ss2023_wa_foodiemate_database /bin/bash`
2. `mongorestore --host localhost --port 27017 --db foodiemate /backup/foodiemate`

## **Verwendete Bibliotheken und Frameworks**

### **Backend und Datenbank**

- nodeJS-Bibliothek: Mongoose <https://mongoosejs.com/docs/>
- JavaScript-Bibliothek: Axios <https://axios-http.com/docs/intro>
- DB-connection: connect-mongo <https://www.npmjs.com/package/connect-mongo>
- Encryption: bcryptjs <https://github.com/dcodeIO/bcrypt.js/blob/master/README.md>

- Authentication: JWT-Token: <https://jwt.io/>
- Navigation: react-router-dom: <https://reactrouter.com/en/main>
- cookie-parser #### Frontend
- Bootstrap: MDBBootstrap <https://mdbootstrap.com/docs/react/>
- Authentication: React-AuthKit: <https://authkit.arkadip.dev/installation/>

## Container

Im Produktivsystem statt localhost -> IP-Adresse #### ss2023\_wa\_foodiemate\_frontend

<http://localhost:20061/>

**ss2023\_wa\_foodiemate\_database**

<http://localhost:20062/>

<http://194.94.204.27:20062/>

**ss2023\_wa\_foodiemate\_user\_api**

API: <http://localhost:20063/users>

Swagger: <http://localhost:20063/api-docs>

**ss2023\_wa\_foodiemate\_recipe\_api**

API: <http://localhost:20064/recipe>

Swagger: <http://localhost:20064/api-docs>

**ss2023\_wa\_foodiemate\_cooking\_together\_api**

API: <http://localhost:20065/cookingTogether>

Swagger: <http://localhost:20065/api-docs>

## API's welche von FoodieMate angeboten werden

### findCookingBuddy

Die "findCookingBuddy" - API liefert eine Liste von Benutzern zurück, die bereit sind, sich mit anderen Menschen zum Kochen zu verabreden. Zurückgegeben werden userName und E-Mail-Adresse. Diese kann in der "inviteCookingBuddy" - API genutzt werden.

GET <http://localhost:20065/cookingTogether/findCookingBuddy>

### inviteCookingBuddy

Die "inviteCookingBuddy" - API schickt eine E-Mail an die angegebene Adresse. Man kann in den Body die Kontaktdaten hinterlegen, bei denen sich der FoodieMate-User melden kann.

```
POST http://localhost:20065/cookingTogether/inviteCookingBuddy
```

Body:

```
{  
    "email": "max@mustermann.de",  
    "contactData": "Meine Kontaktdaten"  
}
```

### getRecipeToDoDrinK

Die "getRecipeToDoDrinK" - API erwartet als param den Key: Drink und als Value ein beliebiges Getränk und liefert ein dazu passendes Gericht.

```
GET http://localhost:20064/recipe/recipesToDoDrinK/Bier
```

Testbar mit Bier, Wasser und Rotwein

### RecipeOfTheDay

Die "RecipeOfTheDay" - API liefert ein zufälliges Rezept aus der Datenbank.

## Testen des Backends

Folgende Funktionalitäten sind z.B. via Postman testbar: ## user\_api

```
POST http://localhost:20063/users/register
```

Body:

```
{  
    "firstName": "Max",  
    "lastName": "Mustermann",  
    "userName": "userMaxMustermann",  
    "emailAddress": "max@mm.de",  
    "password": "pass"  
}
```

```
POST http://localhost:20063/users/login
```

Body:

```
{  
    "userName": "userMaxMustermann",  
    "password": "pass"  
}
```

```
GET http://localhost:20063/users/getUserCTG
```

```
GET http://localhost:20063/users/getByUserName?userName=userMaxMustermann
```

Alle anderen Funktionen der user\_api lassen sich besser über das Frontend testen, da einige der Funktionen auf der Anmeldung und übergebener Authentifikation basiert.

### **recipe\_api**

```
GET http://localhost:20064/recipe/allRecipes  
  
GET http://localhost:20064/recipe/recipeByTitle?title=Nudelsalat  
  
GET http://localhost:20064/recipe/recipeByCount  
Body:  
{  
    "count": 5  
}  
  
GET http://localhost:20064/recipe/recipesToDrink/Bier  
  
GET http://localhost:20064/recipe/recipeOfDay
```

### **cookingTogetherApi**

```
GET http://localhost:20065/cookingTogether/findCookingBuddy  
  
GET http://localhost:20065/cookingTogether/inviteCookingBuddy  
Body  
{  
    "email": "EMAILADDRESS OF COOKINGBUDDY",  
    "contactData": "YOUR MESSAGE"  
}
```

## **Testen des Frontends**

### **Registration**

Um einen neuen Account zu erstellen, klickt man im Login-Kontext auf “Noch keinen Account”:

Als Nächstes das Formular ausfüllen:

### **Login**

Um die Funktionen der WebApp zu testen, muss sich der User zunächst Authentifizieren. Um zum Login zu gelangen, wählt man oben rechts das öffnende Tür-Symbol: Testuser:

userName: h.lendeckel password: pass

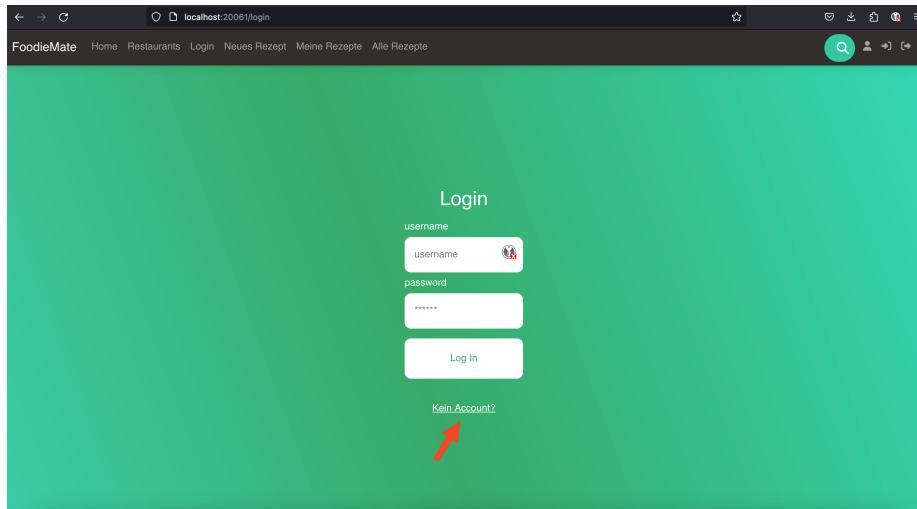


Figure 1: KeinAccount.png

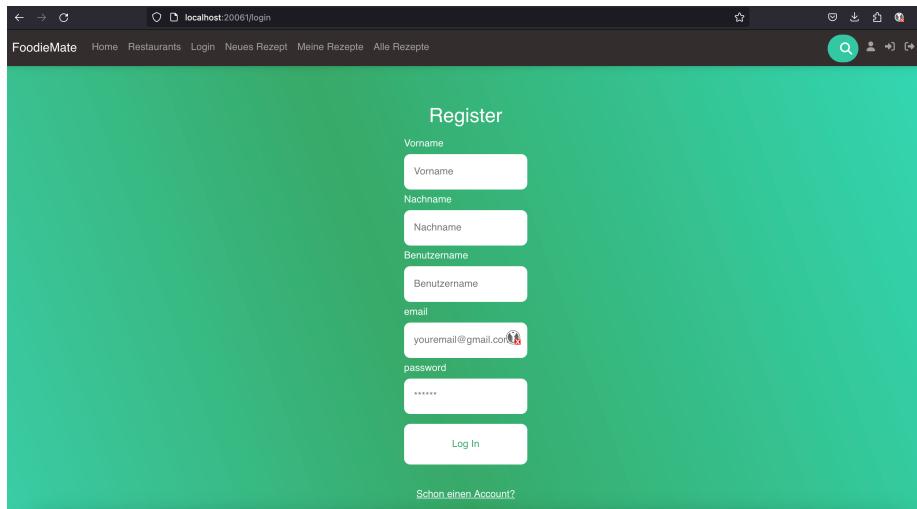


Figure 2: Register.png

userName: m.engel password: pass

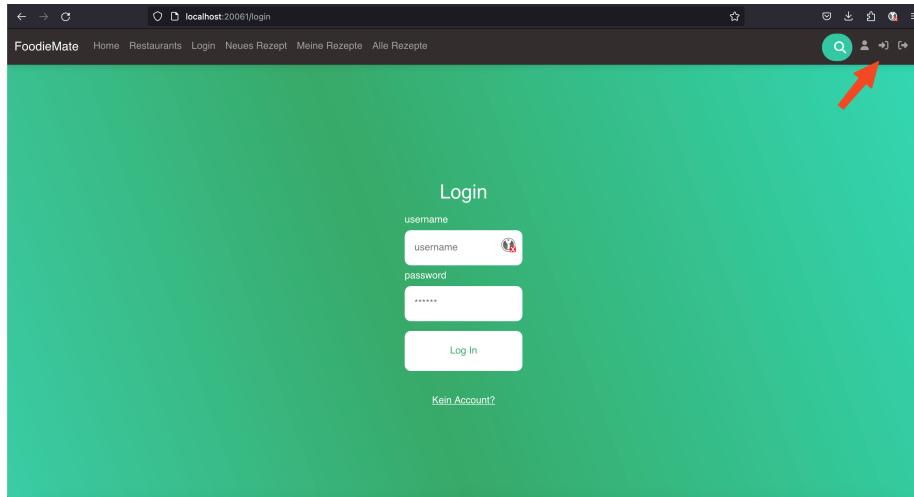


Figure 3: Login.png

## Eigenes Profil

Über den Profil-Button sieht man sein eigenes Profil: - 1 Cooking Together Date löschen - 2 Cooking Together Date setzen - 3 Follower einsehen - 4 Profil bearbeiten - 5 Profil löschen

## Profil bearbeiten

Hinter den aktuellen Einträgen befinden sich Eingabefelder über diese die neuen Informationen eingegeben werden können:

## Andere User suchen

Um andere User zu suchen, muss man den genauen Benutzernamen in das Suchfeld eingeben und mit der Lupe die Suche abschicken:

## Profil eines anderen Users

Auf der Profil-Card eines anderen Nutzers kann man seine rudimentären Informationen sowie sein, wenn vorhanden, Cooking Together Date sehen. Außerdem ist es möglich ihm eine Nachricht über die InviteCookingBuddy-API zu senden. Diese Nachricht geht an die hinterlegte E-Mail-Adresse.

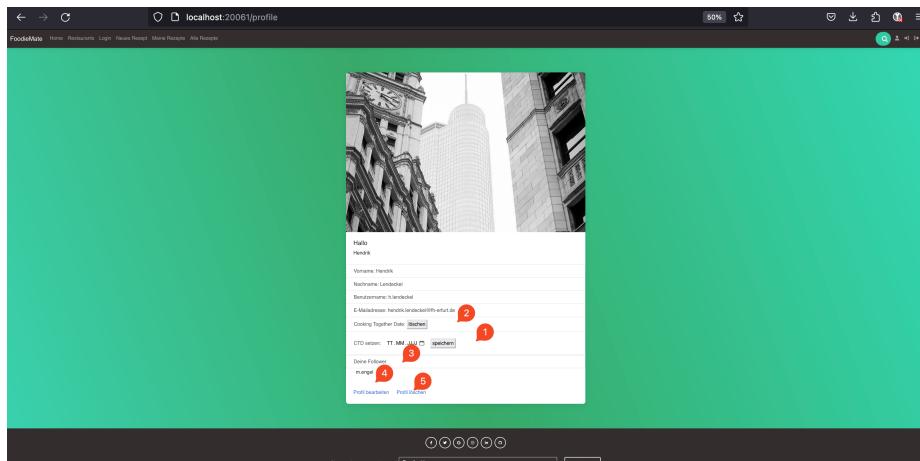


Figure 4: MeinProfil.png

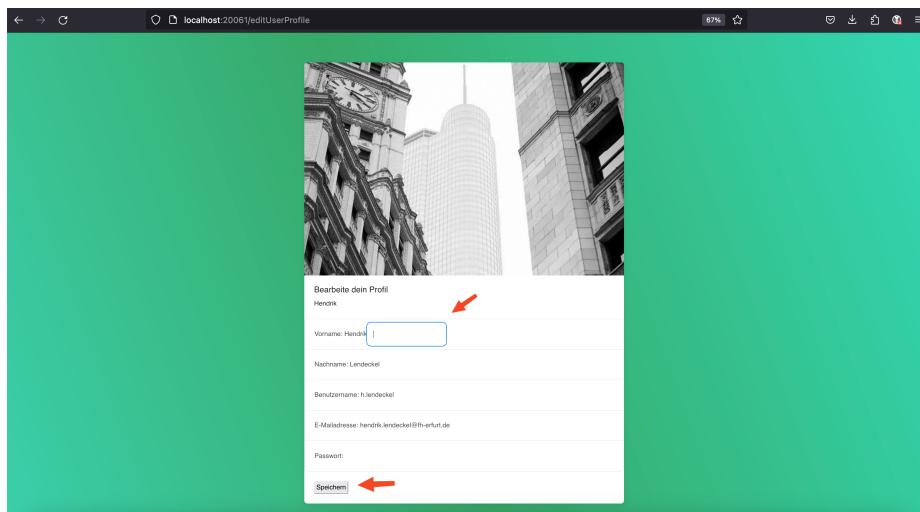


Figure 5: ProfilBearbeiten.png

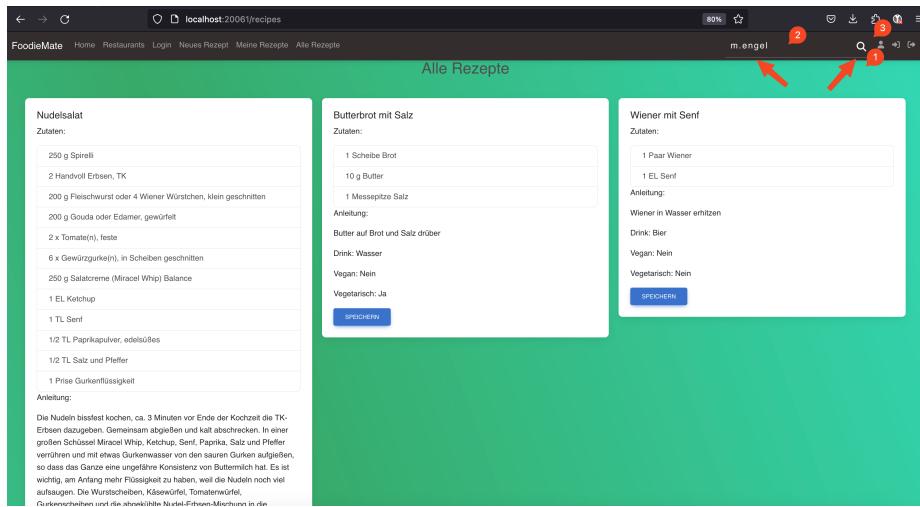


Figure 6: UserSuche.png

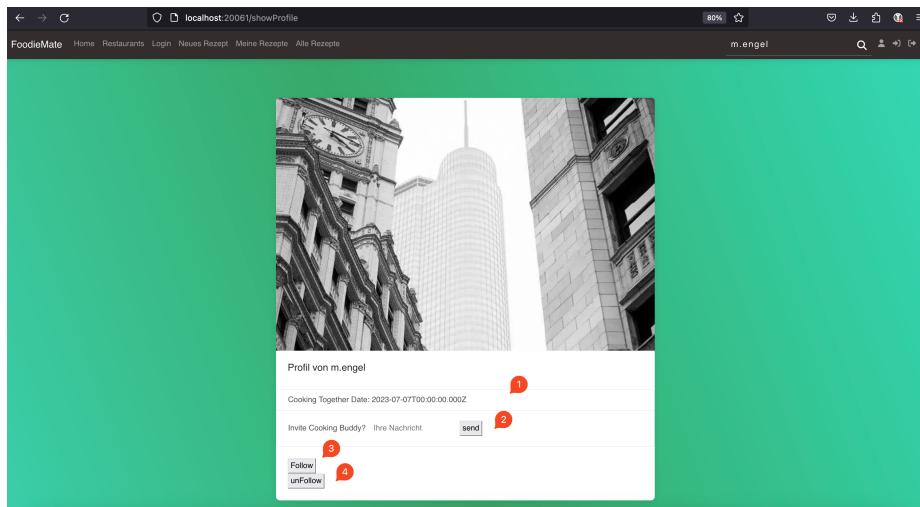


Figure 7: UserSearchProfile.png

## Rezepte

Nach dem Login wird man zunächst auf die Seite “Alle Rezepte” weitergeleitet. Hier werden alle Rezepte aus der Datenbank angezeigt. Er kann diese über den “Speichern” Button zu seinen Rezepten hinzufügen:

AlleRezepte.png

Über den Reiter “Meine Rezepte” kann der User die von ihm angelegten Rezepte und zusätzlich die gespeicherten Rezepte einsehen:

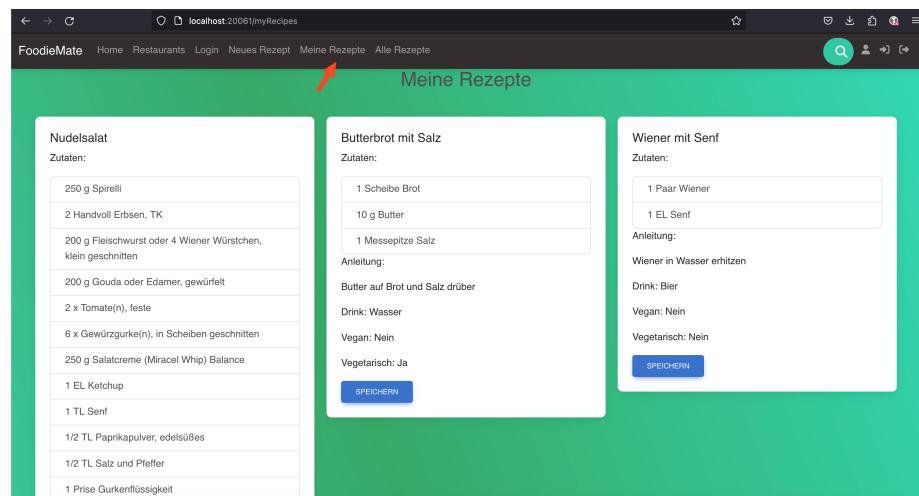
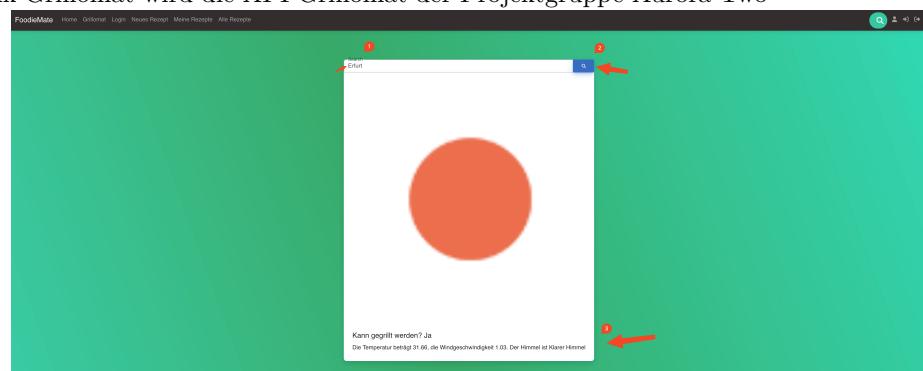


Figure 8: MeineRezepte.png

Über “Neues Rezept” ist es möglich seine eigenen Rezepte anzulegen. 1-Zutat hinzufügen, 2-Zutat entfernen

## Grillomat der Gruppe AuroraTwo

Über den Link Grillomat wird die API Grillomat der Projektgruppe Aurora Two eingebunden:



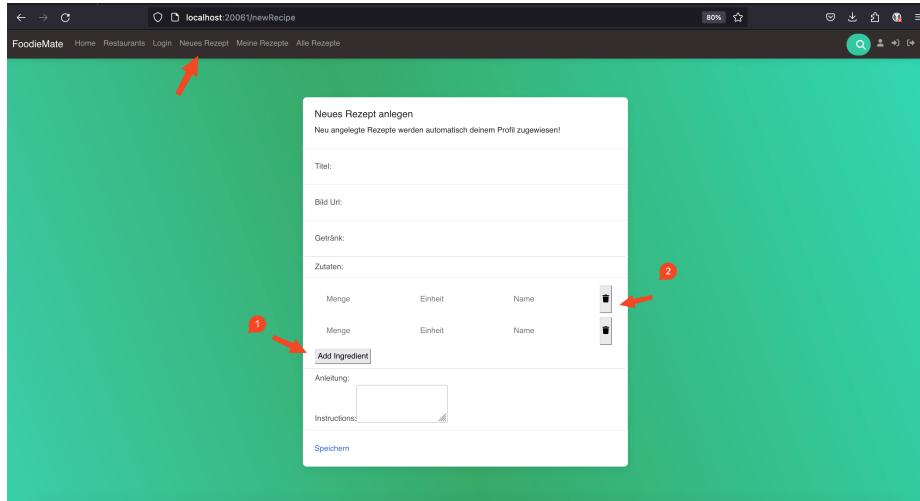


Figure 9: NeuesRezept.png

## Quellen

### Bilder

<https://www.pexels.com/de-de/> ### ChatGPT <https://chat.openai.com/>  
 ChatGPT wurde benutzt, um ein grobes Beispiel Schema zu erzeugen für die Routen im Backend. Dieses wurde als Grundlage für alle Routen im Backend genutzt:  
 \* Prompt: "gib mir ein Schema für eine node.js route für den Login einer users"  
 \* Antwort:

```
// POST-Anfrage für den Login
router.post('/login', (req, res) => {
  // Benutzername und Passwort aus der Anfrage extrahieren
  const { username, password } = req.body;

  // Überprüfung der Benutzeranmeldeinformationen
  if (username === 'admin' && password === 'pass123') {
    // Authentifizierung erfolgreich
    res.status(200).json({ message: 'Login erfolgreich' });
  } else {
    // Authentifizierung fehlgeschlagen
    res.status(401).json({ message: 'Ungültige Anmeldeinformationen' });
  }
});
```