Create a small program or script that implements at least the following:

- Load the json containing scooters & users into memory.
- Scooters that are not assigned to a user and are decommissioned should be deleted.
- Scooters that are not assigned to a user and not decommissioned should be assigned to users without a scooter.

Additionally, please add tests for the functions that you implement. Feel free to add any more relevant functionality to showcase your coding skills. Your program/script should be runnable, and it should logically log the operations that it is performing. When all operations are done the final version of the data should be printed.

Dokumentation des Programms

Überblick

Dieses Programm ist ein Express-Backend, das JSON-Daten über E-Scooter und Benutzer lädt, manipuliert und die Ergebnisse über eine API bereitstellt. Es ist modular aufgebaut und enthält separate Komponenten für das Laden, Speichern und Verarbeiten der Daten. Die manipulierten Daten werden in `src/database/data.json` gespeichert, während die ursprünglichen Daten in `dataUnmanipulated.json` aufbewahrt werden. (Zur veranschaulichung)

Komponenten

- **src/components/dataComponent.ts**: Enthält Funktionen zum Laden und Speichern der Daten.
- **src/components/processComponent.ts**: Enthält die Funktion zum Verarbeiten der Scooter-Daten.
- **src/router/processScooters.ts**: Definiert die API-Routen zum Verarbeiten der Scooter-Daten.

`loadData`: Liest die JSON-Daten aus der Datei und gibt sie als JavaScript-Objekt zurück.

`saveData`: Speichert die manipulierten Daten zurück in die Datei.

`processScooters`: Entfernt unzugewiesene und außer Betrieb befindliche Scooter und weist verfügbare Scooter nicht zugewiesenen Benutzern zu.

```
import type {Router, Request, Response} from 'express';
import express from 'express';
import ServerError from '../utility/serverError';
import {processScooters} from '../components/processComponent';
import {saveData} from '../components/dataComponent';
const processScootersRouter: Router = express.Router();

processScootersRouter.get('/', async (req: Request, res: Response) => {
 try {
   const updatedData = processScooters();
   saveData(updatedData);
   return res.send(updatedData);
 } catch (e) {
   throw new ServerError('TEST_ERROR', {info: 500});
 }
});

export default processScootersRouter;
```

Testfälle

Tests zum Entfernen und Zuweisen von Scootern

Test-Datei (`src/tests/processScooters.test.ts`):

Erklärung der Tests:

- **Test: Entfernen von außer Betrieb befindlichen und nicht zugewiesenen Scootern**
  - **Ziel:** Sicherstellen, dass Scooter, die weder zugewiesen noch betriebsfähig sind, aus den Daten entfernt werden.
  - **Vorgehen:** Mock-Daten werden geladen und die Funktion `processScooters` wird aufgerufen. Der Test überprüft dann, ob die entsprechenden Scooter nicht mehr in den Daten enthalten sind.
- **Test: Zuweisen verfügbarer Scooter an unzugewiesene Benutzer**
  - **Ziel:** Sicherstellen, dass verfügbare Scooter an Benutzer ohne Scooter zugewiesen werden.
  - **Vorgehen:** Mock-Daten werden geladen und die Funktion `processScooters` wird aufgerufen. Der Test überprüft dann, ob die Benutzer ohne Scooter einen verfügbaren Scooter zugewiesen bekommen haben.

Erweiterung für Task 2

Das Backend ist mit Express.js aufgebaut, was eine einfache Erweiterung für zukünftige Aufgaben ermöglicht. Zum Beispiel können weitere Routen und Controller für die Verwaltung von Benutzern, Fahrzeugen und Anforderungen hinzugefügt werden. Dies bietet eine skalierbare und flexible Grundlage für die Entwicklung eines umfangreichen Backend-Systems.

Durch die modulare Struktur des Codes können neue Funktionen leicht integriert und bestehende Komponenten einfach gewartet werden.

Technische Implementierung

Verwendete Technologien

- **Prisma** als ORM
- **PostgreSQL** als Datenbank
- **Express.js** als Web-Framework
- **OpenAPI** für die API-Spezifikation und Dokumentation (Swagger UI)
- **TypeScript** für die Typisierung

Datenbankschema

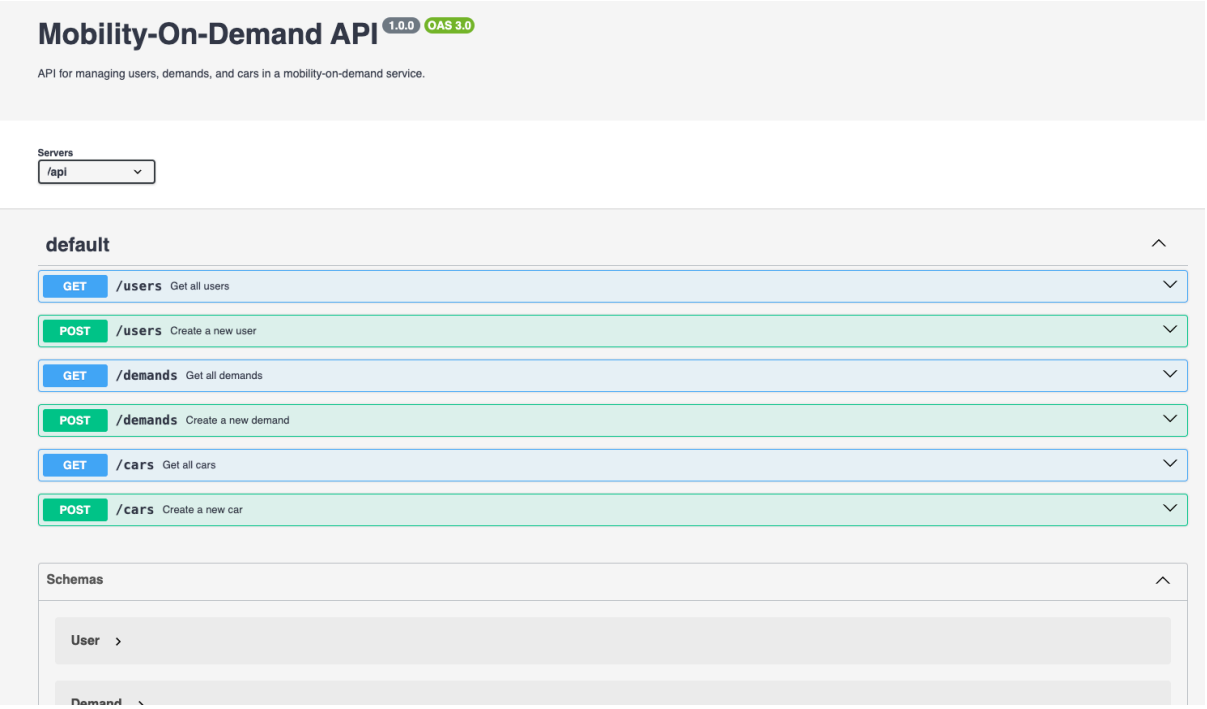
Das Datenbankschema wurde unter Verwendung von Prisma wie folgt erstellt: Siehe ER-Diagram

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id      Int     @id @default(autoincrement())
  name    String
  address String
  age     Int
  demands Demand[]
}

model Demand {
```

```
  id              Int     @id @default(autoincrement())
  user_id         Int
  user            User    @relation(fields: [user_id], references: [id])
  pickupLocation  String
  dropoffLocation String
  time            DateTime
  passengers      Int
}

model Car {
  id              Int     @id @default(autoincrement())
  engineType      String
  availableSeats  Int
  locationType    String
  locationCoordinates Float[]
  status          String
}
```

API-Spezifikation

Die API-Spezifikation wurde mithilfe von OpenAPI erstellt und in der Datei `api-doc.yml` definiert. Diese Datei wird verwendet, um die Swagger UI zu konfigurieren, die unter `/api-docs` verfügbar ist.



API-Routen

Die API-Routen sind wie folgt definiert:

```
app.use('/getDemands', getDemandsRouter);
app.use('/getCars', getCarsRouter);
app.use('/postCar', postCarRouter);
app.use('/postDemand', postDemandRouter);
app.use('/postUser', postUserRouter);
app.use('/getUser', getUserRouter);
```

Systemarchitektur

Die Systemarchitektur umfasst folgende Komponenten:

1. **Backend**: Das Backend basiert auf Express.js und verwendet Prisma als ORM zur Interaktion mit der PostgreSQL-Datenbank. Die API-Spezifikation wird mit OpenAPI definiert und mit Swagger UI dokumentiert.
2. **Datenbank**: Eine PostgreSQL-Datenbank speichert alle Daten zu Benutzern, Anforderungen und Autos.
3.

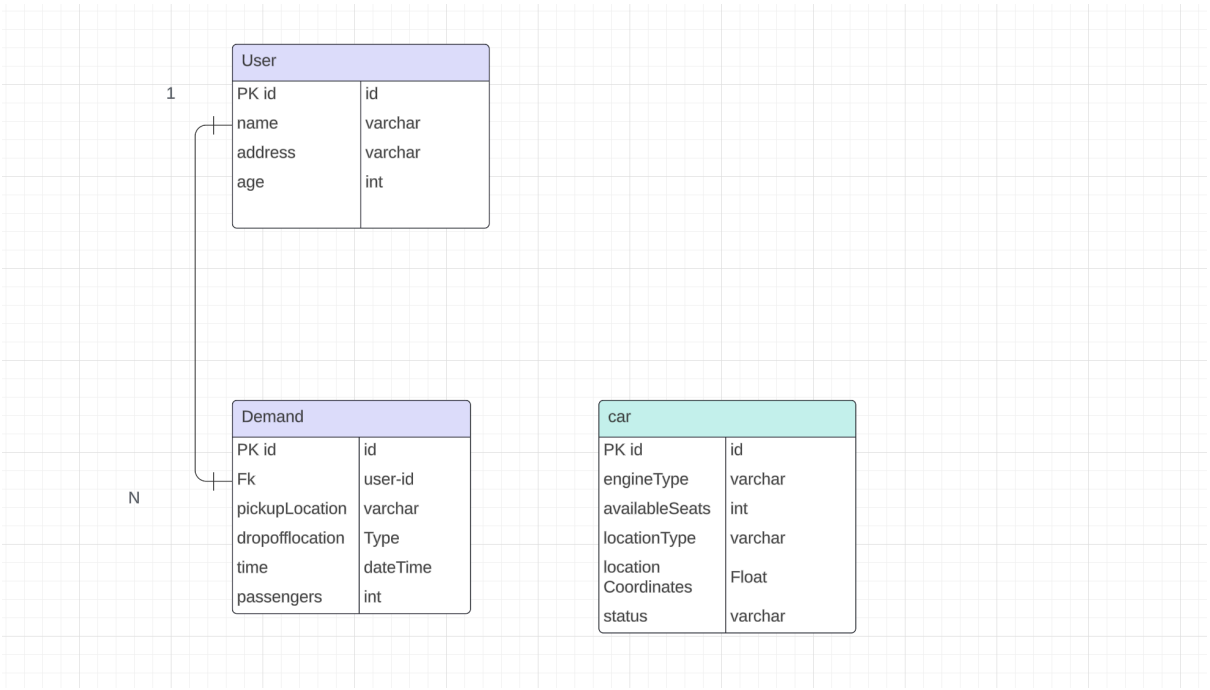
Ausführung

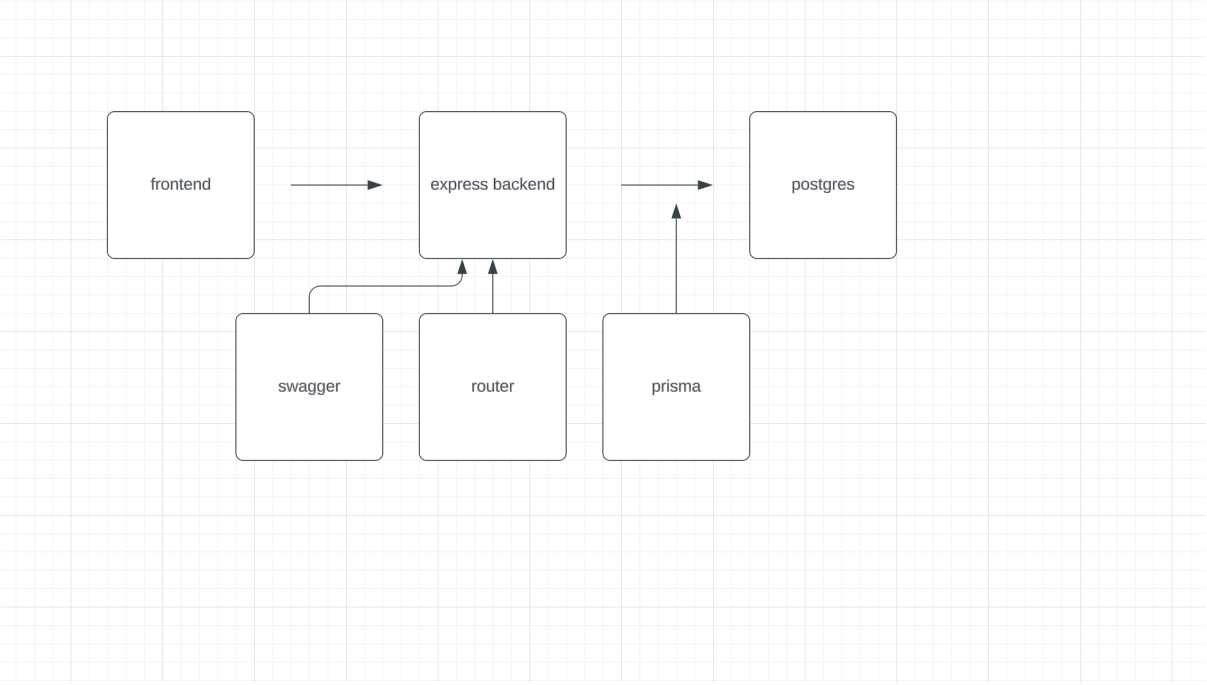
Um das Projekt auszuführen, stellen Sie sicher, dass alle Abhängigkeiten installiert sind und starten Sie den Server mit:

```
npm start
```

Die API-Dokumentation ist unter `http://localhost:<port>/api-docs` verfügbar.

ER Diagramm


User
PK id
id
name
varchar
address
varchar
age
int
1
N
Demand
PK id
id
Fk
user-id
pickupLocation
varchar
dropofflocation
Type
time
dateTime
passengers
int
car
PK id
id
engineType
varchar
availableSeats
int
locationType
varchar
location Coordinates
Float
status
varchar


frontend
express backend
postgres
swagger
router
prisma

Architektur