ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ ПО ИНДУСТРИАЛНИ ТЕХНОЛОГИИ

КУРСОВ ПРОЕКТ

Дисциплина: „Бази Данни“

Тема:

“Electric Bicycle Rental Management System”

Изготвили:

Алек Ноев 361224012
Александър Цонев 361224054
Горан Дулев 361224021
Емил Коруджиев 361224027
Станислав Стаматов 361224005

Група:20

Специалност: ИСИИ, Курс: II

София, 2025

Съдържание

Увод и цел на проекта

Проектът има за цел да разработи база данни на система за наем на електрически велосипеди. За по-лесно имплементиране на бъдещи промени и за покриване на евентуални бъдещи изисквания от страна на клиента, базата от данни е пригодена да работи с всякакъв тип превозни средства – от велосипеди и електрически тротинетки, до леки автомобили и микробуси.

Основната идея е да се предостави на клиентите максимално бърз и лесен начин за наемане на превозно средство за лични нужди, като системата автоматизира ключовите процеси:

- **Гъвкавост**: Поддръжка на разнообразни видове превозни средства без ограничения.
- **Плащания**: Възможност за заплащане на конкретно пътуване или използване на абонаментни планове за определен период.
- **Сигурност**: Изискване на необходими документи за валидиране на потребителите преди наем.
- **Контрол**: Проследяване на всяко пътуване и следене на техническото състояние на активите.

Архитектурата на базата данни е изградена върху **MySQL 8.0** и използва **Flyway** за управление на миграциите.

За да се гарантира висока производителност и надеждност в реална продукционна среда, са имплементирани оптимизирани заявки, стратегически дефинирани индекси и други спомагателни архитектурни механизми.

Проектиране

Структура на базата данни

Таблица: `customers`

Съхранява профилите на потребителите.

Поле	Тип данни	Описание	Ограничения / Връзки
`id`	INT	Уникален идентификатор	Primary Key, Auto Increment
`created_date`	TIMESTAMP	Дата на създаване	Default: Current Timestamp
`updated_date`	TIMESTAMP	Дата на последна редакция	Auto Update
`first_name`	VARCHAR(50)	Име на клиента	Not Null
`last_name`	VARCHAR(50)	Фамилия на клиента	Not Null
`gender`	ENUM	Пол	Values: 'MALE', 'FEMALE', 'OTHER'
`email`	VARCHAR(150)	Електронна поща	**Unique**, Check: съдържа '@' и '.'
`phone`	VARCHAR(15)	Телефонен номер	**Unique**, Check: RegEx формат
`date_of_birth`	DATE	Дата на раждане	Not Null
`status`	ENUM	Статус на акаунта	Values: 'ACTIVE', 'PENDING', 'SUSPENDED', 'CLOSED'

Таблица: `subscription_plans`

Номенклатура на предлаганите абонаменти.

Поле	Тип данни	Описание	Ограничения / Връзки
`id`	INT	Уникален идентификатор	Primary Key, Auto Increment
`name`	VARCHAR(100)	Име на плана	**Unique**, Not Null
`description`	TEXT	Описание на условията	Not Null
`plan_type`	ENUM	Ниво на достъп	Values: 'BASIC', 'VIP'
`billing_period`	ENUM	Период на фактуриране	Values: 'WEEKLY', 'MONTHLY', 'YEARLY'
`price`	DECIMAL(10,2)	Цена на плана	Check: `price > 0`
`currency`	VARCHAR(3)	Валута (ISO код)	Default: 'EUR'
`duration_days`	INT	Продължителност в дни	*Добавено във V1.0.1*, Not Null

Таблица: `subscriptions`

Свързваща таблица между клиенти и планове.

Поле	Тип данни	Описание	Ограничения / Връзки
`id`	INT	Уникален идентификатор	Primary Key, Auto Increment
`customer_id`	INT	Клиент	FK `customers(id)`
`subscription_plan_id`	INT	Избран план	FK `subscription_plans(id)`
`start_date`	DATE	Начална дата	Not Null
`end_date`	DATE	Крайна дата	Check: `end_date >= start_date` (или NULL)
`status`	ENUM	Статус на абонамента	'ACTIVE', 'PAUSED', 'CANCELLED'
`auto_renewal`	BOOLEAN	Автоматично подновяване	Default: TRUE

Таблица: `vehicles`

Основен регистър на активите.

Поле	Тип данни	Описание	Ограничения / Връзки
`id`	INT	Уникален идентификатор	Primary Key, Auto Increment
`identifier`	VARCHAR(50)	Вътрешен номер/код	**Unique**, Not Null
`registration_number`	VARCHAR(20)	Регистрационен номер	**Unique**, Not Null
`brand`	VARCHAR(50)	Марка	Not Null
`model`	VARCHAR(50)	Модел	Not Null
`power_type`	ENUM	Тип задвижване	'ELECTRIC', 'DIESEL', 'PETROL', 'HYBRID', 'HUMAN_POWERED'
`vehicle_type`	ENUM	Категория МПС	'BICYCLE', 'SCOOTER', 'MICROCAR', 'SUPERMINIS', 'SUV', 'VAN'
`status`	ENUM	Оперативен статус	'AVAILABLE', 'RENTED', 'MAINTENANCE', 'INACTIVE'
`last_odometer_km`	DECIMAL(10,2)	Пробег	Default: 0, Check: `>= 0`
`location`	POINT SRID 4326	GPS Координати	Nullable
`price_per_minute`	DECIMAL(10,2)	Цена на минута	Nullable
`price_per_km`	DECIMAL(10,2)	Цена на км	Nullable
`price_for_rental`	DECIMAL(10,2)	Фиксирана цена за наем	Nullable

- Съществуват сложни проверки за съвместимост между `power_type` и `vehicle_type` (напр. Велосипед може да е Human Powered, но Ван не може).

Таблица: `maintenance`

Сервизна история.

Поле	Тип данни	Описание	Ограничения / Връзки
`id`	INT	Уникален идентификатор	Primary Key, Auto Increment
`vehicle_id`	INT	Превозно средство	FK `vehicles(id)`
`scheduled_date`	DATE	Планирана дата	Not Null
`maintenance_type`	ENUM	Вид дейност	'REGULAR', 'REPAIR', 'EMERGENCY'
`cost`	DECIMAL(10,2)	Цена на ремонта	Check: `cost > 0`
`status`	ENUM	Статус на ремонта	'SCHEDULED', 'IN_PROGRESS', 'COMPLETED', 'CANCELLED'

Таблица: `rentals`

Основна транзакционна таблица за ползване на услугите.

Поле	Тип данни	Описание	Ограничения / Връзки
`id`	INT	Уникален идентификатор	Primary Key, Auto Increment
`customer_id`	INT	Клиент	FK `customers(id)`
`vehicle_id`	INT	Превозно средство	FK `vehicles(id)`
`start_datetime`	TIMESTAMP	Начало на наема	Not Null
`end_datetime`	TIMESTAMP	Край на наема	Nullable (ако е активен)
`price`	DECIMAL(10,2)	Крайна цена	Nullable (изчислява се при край)
`status`	ENUM	Статус	'ACTIVE', 'COMPLETED', 'CANCELLED'
`distance_km`	DECIMAL(10,2)	Изминато разстояние	Nullable

- Полето `trajectory` беше премахнато във версия V1.0.4., поради желание за централизация на данни и лимити от MySQL, касаещи комбинации на точки в траектория

Таблица: `rental_waypoints`

Съхранява маршрута на движение.

Поле	Тип данни	Описание	Ограничения / Връзки
`id`	INT	Уникален идентификатор	Primary Key
`rental_id`	INT	Към кой наем се отнася	FK `rentals(id)`
`location`	POINT	Координати	SRID 4326, Not Null
`timestamp`	DATETIME	Време на записа	Not Null
`speed`	FLOAT	Скорост в този момент	Nullable

Таблица: `payments`

Поле	Тип данни	Описание	Ограничения / Връзки
`id`	INT	Уникален идентификатор	Primary Key
`customer_id`	INT	Платец	FK `customers(id)`
`rental_id`	INT	За наем (ако има)	FK `rentals(id)`
`subscription_id`	INT	За абонамент (ако има)	FK `subscriptions(id)`
`amount`	DECIMAL(10,2)	Сума	Check: `amount > 0`
`payment_method`	ENUM	Метод	'CARD', 'BANK_TRANSFER'
`status`	ENUM	Статус на транзакцията	'PENDING', 'COMPLETED', 'FAILED', etc.

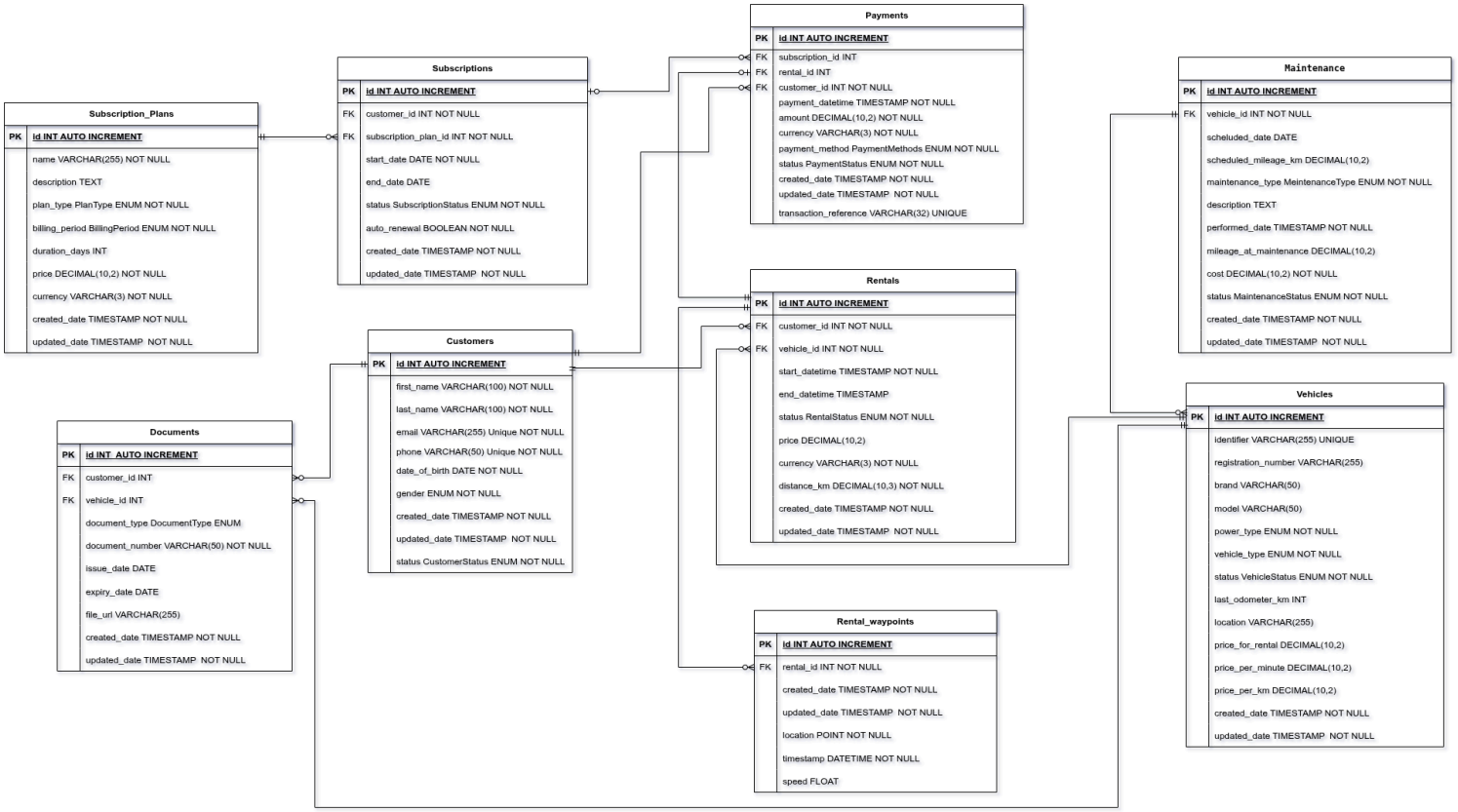
- Плащането трябва да сочи или към `rental_id`, или към `subscription_id`, но никога и към двете едновременно (Exclusive OR logic).

Таблица: documents

Поле	Тип данни	Описание	Ограничения / Връзки
id	INT	Уникален идентификатор	Primary Key
document_number	VARCHAR(20)	Номер на документа	Not Null
document_type	ENUM	Тип документ	'ID_CARD', 'PASSPORT', 'DRIVERS_LICENSE', 'INSURANCE', 'REGISTRATION'
issue_date	DATE	Дата на издаване	Not Null
expiry_date	DATE	Дата на валидност	Check: expiry >= issue
file_url	VARCHAR(255)	Път до файла	Nullable
customer_id	INT	Собственик (за лични док.)	Foreign Key
vehicle_id	INT	За МПС (за техн. док.)	Foreign Key

- Валидира се, че личните документи са свързани само с customer_id, а техническите (застраховка, талон) само с vehicle_id.

Entity Relationship Diagram



Link:
https://drive.google.com/file/d/1G3jXshQiXua6YS_0xuV7RQ0NsHUcdJ-N/view?usp=sharing

Заявки

Реализирани са заявки, които биха се използвали в реална продуктова среда:
Всички имплементации могат да се намерят в директорията *'/queries'* на проекта.

Търсене на най-близки налични превозни средства по тип и дадена мерна единица за разстояние

```
SELECT v.identifier,
       v.brand,
       v.model,
       v.vehicle_type,
       v.power_type,
       v.price_per_minute,
       v.price_per_km,
       v.price_for_rental,
       v.location,
       ROUND(ST_DISTANCE(ST_GeomFromText('POINT(23.28009015708178
42.652249987433666)', 4326), v.location,
                         'metre'), 0) as distance_from_location
```

```
FROM rental_service.vehicles v
WHERE v.vehicle_type = 'BICYCLE'
  AND v.status = 'AVAILABLE'
ORDER BY distance_from_location
    LIMIT 20;
```

Намиране на активен наем за клиент

```
SELECT r.id                                          as rental_id,
       v.identifier,
       v.brand,
       v.model,
       r.start_datetime,
       TIMESTAMPDIFF(MINUTE, r.start_datetime, NOW()) as rental_minutes,
       ST_AsText(v.location)                         as
current_location
FROM rental_service.rentals r
         JOIN rental_service.vehicles v ON r.vehicle_id = v.id
WHERE r.customer_id = 12
  AND r.status = 'ACTIVE'
ORDER BY r.start_datetime DESC;
```

Проверка на документи на клиент

```
SELECT c.first_name,
       c.last_name,
       d.document_type,
       d.document_number,
       d.issue_date,
       d.expiry_date,
       CASE
           WHEN d.expiry_date < CURDATE() THEN 'EXPIRED'
           WHEN d.expiry_date <= DATE_ADD(CURDATE(), INTERVAL 30 DAY)
THEN 'EXPIRING_SOON'
           ELSE 'VALID'
           END as status
FROM rental_service.customers c
         JOIN rental_service.documents d ON c.id = d.customer_id
WHERE c.id = 6119
  AND d.document_type IN ('DRIVERS_LICENSE', 'ID_CARD')
ORDER BY d.expiry_date ASC;
```

Инициализиране на rental

```
INSERT INTO rental_service.rentals (
    customer_id,
    vehicle_id,
    start_datetime,
    status,
    currency
) VALUES (
    123,
    456,
    NOW(),
    'ACTIVE',
    'EUR'
);
```

Завършване на rental и калкулиране на крайна цена.

```
UPDATE rental_service.rentals
SET
    end_datetime = NOW(),
    status = 'COMPLETED',
    distance_km = 15.5,
    price = (
        TIMESTAMPDIFF(MINUTE, start_datetime, NOW()) *
        (SELECT price_per_minute FROM rental_service.vehicles WHERE id =
456) +
        15.5 * (SELECT price_per_km FROM rental_service.vehicles WHERE
id = 456)
    )
WHERE id = 789 AND status = 'ACTIVE';
-- Актуализация на състоянието и локацията на превозното средство
UPDATE rental_service.vehicles
SET
    status = 'AVAILABLE',
    last_odometer_km = last_odometer_km + 15.5,
    location = ST_GeomFromText('POINT(12.34 56.78)', 4326)
WHERE id = 456;
```

Имплементирани са и комплексни заявки (могат да се намерят в “/queries/complex_queries.sql”):

Детайлен отчет за превозно средство

- Извлича пълна информация за конкретно МПС: текущ статус, последен наем, текущи проблеми по поддръжката, статистика за скоростта (от GPS данните) и валидност на документите на последния шофьор.

Финансов отчет: Приходи, Разходи и Печалба (подневна статистика)

- Комплексна справка, която агрегира данни от три източника (наеми, абонаменти, ремонти) за даден период и изчислява дневна печалба.

Индекси

За оптимизиране на заявките са създадени индекси, върху често използвани колони, за филтриране и JOIN.

Индекси:
customers(status)
vehicles(location) - spatial index
vehicles(status)
vehicles(vehicle_type, status)
rentals(customer_id)
rentals(vehicle_id)
rentals(status)
rentals(customer_id, status)
rentals(start_datetime DESC)
rentals(customer_id, start_datetime DESC)
rental_waypoints(rental_id)
rental_waypoints(timestamp)
rental_waypoints(rental_id, speed)
subscriptions(customer_id)
subscriptions(status)
subscriptions(start_date, end_date)
maintenance(vehicle_id)
maintenance(status)
maintenance(scheduled_date)
maintenance(performed_date)
payments(customer_id)
payments(rental_id)
payments(subscription_id)
payments(status)
documents(customer_id)
documents(vehicle_id)
documents(expiry_date)
documents(customer_id, document_type)

Trigger-и

Тригерите автоматизират бизнес логиката и гарантират цялостност на данните.

Автоматична смяна на статуса на автомобил
Когато се създаде нов наем със статус ACTIVE, състоянието на съответното превозно средство автоматично се променя на RENTED. Това предотвратява дублирано наемане.

```
CREATE TRIGGER trg_rentals_set_vehicle_rented
    AFTER INSERT ON rental_service.rentals
    FOR EACH ROW
BEGIN
    IF NEW.status = 'ACTIVE' THEN
    UPDATE rental_service.vehicles SET status = 'RENTED' WHERE id =
NEW.vehicle_id;
END IF;
END;
```

Миграции

Системата използва **Flyway** за контрол на версиите. При стартиране се изпълняват следните скриптове в посочения ред или се проверява тяхната структура, ако са вече приложени:

- **V1.0.0__init.sql**: Създаване на базата и първоначалните таблици.
- **V1.0.1__add_new_columns.sql**: Добавяне на duration_days в плановете за абонаменти и price_for_rental в автомобилите.
- **V1.0.2__fix_document_constraints.sql**: Корекция на логиката за документите (изтриване на стари и създаване на нови constraints).
- **V1.0.3__vehicle_location_change_datatype.sql:** Промяна на vehicles.location от VARCHAR към геометричен тип POINT.
- **V1.0.4__remove_rental_trajectory.sql**: Премахване на излишната колона trajectory от таблица rentals.

Защита на данни

Backup

Тъй като базата данни е сравнително компактна , най-ефективният метод за архивиране е логическият бекъп чрез mysqldump.
Инструмент: mysqldump (влиза в стандартния пакет на MySQL Client).
Честота: Всеки ден, извън пиковите часове.
Процес:

- Стартиране на mysqldump през cron job в Docker контейнер.
- Компресиране на изходния SQL файл.
- Криптиране на архива.
- Изпращане към отдалечено хранилище (S3 / Google Cloud Storage) за защита от хардуерен срив на сървъра.

Примерно стартиране на командата:

```
mysqldump -u [USER] -p[PASS] rental_service --result-file="path..."
--skip-create-options --skip-lock-tables --skip-disable-keys
--skip-add-drop-table --skip-extended-insert --no-create-info
--no-tablespaces --hex-blob --skip-extended-insert --complete-insert
```

Потребители

За да се минимизира риска при пробив (Principle of Least Privilege), приложението не трябва да използва **root** потребител. Дефинирани са следните потребители и роли:

- **app_user**: Основният акаунт, ползван от Back-end приложението. Има права да чете и пише в оперативните таблици, но не може да променя структурата на базата (DROP/ALTER) или да трие логове.
- **analyst_user**: Акаунт за аналитични цели. Има права само за четене (SELECT) върху всички таблици.
- **admin_user**: Администраторски акаунт за миграции и поддръжка.

Физически дизайн и допълнителни технологии

За да се осигури висока производителност, мащабируемост и надеждност на системата, архитектурното решение надгражда стандартния релационен модел (MySQL) с допълнителни специализирани технологии.

Основен Оперативен Слой

MySQL играе ролята на **Source of Truth**. Тук се съхраняват всички критични данни:

- Потребителски профили и документи.
- Финансови транзакции (плащания).
- Активни и приключени наеми (rentals).

Слой за Кеширане и Бързи Данни

За да се намали натоварването върху основната база и да се осигури моментална реакция на интерфейса, се внедрява **Redis** като in-memory data store.

- **Geospatial Indexing (GeoRedis):**
  - *Проблем:* Честите заявки тип "Покажи всички коли в радиус от 500м" са тежки за релационна база, когато потребителите са хиляди.
  - *Решение:* Redis поддържа специализирани команди (`GEOADD`, `GEORADIUS`), които позволяват бързо извличане на свободни превозни средства върху картата в реално време.

Аналитичен Слой

С експоненциалното натрупване на исторически данни, таблиците **rentals** и **rental_waypoints** ще достигнат обем от милиони записи. Изпълнението на тежки аналитични справки директно върху оперативната база данни би довело до значително забавяне на системата. За решаване на този проблем се внедрява Google BigQuery като специализирано решение за **Data Warehousing**. То поема аналитичното натоварване и гарантира бързодействието на основната услуга.

- **Data Warehousing:** Историческите данни се архивират периодично от MySQL в BigQuery. Това позволява таблиците в MySQL да останат леки и бързи.
- **Business Intelligence (BI):** Извършване на комплексни анализи, като:
    - **Heatmaps: (Пример)** Кои зони в града са най-натоварени в 18:00 ч.?
    - **Revenue Reporting:** Финансови отчети за приходи по типове превозни средства и периоди.

Управление на Идентичността (Single Sign-On)

Вместо да се изгражда собствена система за автентикация, проектът залага на интеграция с външни **Identity Providers (IdP)** чрез протоколи като **OAuth2** и **OIDC**.

- **Сигурност на данните:** Най-чувствителната информация – потребителските пароли и лични данни – **не се съхраняват в локалната база данни**. Това елиминира риска от изтичане на пароли при евентуален пробив в нашата система.
- **Удобство:** Потребителите могат да използват съществуващи профили (Google, Facebook, Apple) за незабавен достъп.
- **Бъдещо развитие (Document Storage):** Много модерни IdP услуги предлагат специализирани модули за *Identity Verification*. В бъдеще, съхранението и валидацията на чувствителни документи може да бъде изнесено изцяло към този слой, гарантирайки най-високо ниво на криптиране и съответствие с регулациите, без да се натоварва нашата инфраструктура.

Инструкции за стартиране

За да работите с проекта локално, трябва да имате инсталирани необходимите инструменти и да следвате стъпките за стартиране на базата данни и бекенд приложението.

Предварителни изисквания

- Docker Desktop & Docker Compose: Необходими за контейнеризация и стартиране на MySQL базата данни и Flyway миграциите.
- Java Development Kit (JDK) 21 (или по-нова версия): Необходима за компилиране и стартиране на Spring Boot приложението.
- Git: За клониране на хранилището с изходния код.

Стартиране на Базата Данни (Docker)

Подготовка на Environment променливи
Файлът **docker-compose.yaml** изисква определени променливи на средата. Тъй като те не са "hardcoded", трябва да създадете файл с име .env в същата директория, където е **docker-compose.yaml.**

Примерно съдържание на .env файл:

```
MYSQL_ROOT_PASSWORD=secret_root_pass
MYSQL_USER=rental_user
MYSQL_PASSWORD=pass
MYSQL_PORT=3307
```

Стартиране на контейнерите
Отворете терминал в главната директория на проекта (където е docker-compose.yaml) и изпълнете:

```
docker-compose up -d
```

Процес на инициализация:

- db (MySQL): Стартира се MySQL 8.0 контейнер. Той ще бъде достъпен на порт 3307 (или този, който сте задали в .env).
- flyway: Изчаква базата да стане готова и автоматично прилага SQL миграциите от папката ./flyway/sql.

Стартиране на Spring Boot Приложението

След като базата данни е успешна стартирана и миграциите са приложени, можете да стартирате Back-end приложението.

Навигация
Отворете терминал в основната директория на проекта (където се намират файловете build.gradle и gradlew).

Конфигурация
Уверете се, че настройките в src/main/resources/application.properties съвпадат с данните за връзка, които сте дефинирали в .env файла (хост, порт, потребител, парола), в приложението се използва Google Maps API, като то също трябва да бъде предоставено в конфигорациите.

Стартиране
Използвайте предоставения Gradle Wrapper за стартиране на приложението (не е нужно да имате инсталиран Gradle глобално).

За Linux / macOS:

```
./gradlew bootRun
```

За Windows (Command Prompt / PowerShell):

```
gradlew.bat bootRun
```

При успешно стартиране, ще видите логовете на **Spring Boot** и съобщение, че приложението работи (обикновено на порт **8080**).

Генериране на Тестови Данни (по желание)

За да напълним базата с (близо до) реалистични данни за тестови цели, използвайте предоставения **Python** скрипт **create-data.py.**

Изисквания

- Python 3.8+
- Инсталирани библиотеки чрез pip:

```
pip install mysql-connector-python faker python-dotenv
```

Стартиране

Уверете се, че файлът .env съдържа правилните данни за връзка (същите като за Docker).
Пример:

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=root
DB_NAME=rental_service
DB_CHARSET=utf8mb4
DB_PORT=3306
```

Стартирайте скрипта в терминала:

```
python create-data.py
```

Скриптът автоматично ще се свърже с базата и ще генерира тестови данни.
Освен скрипта, е предоставен и примерен dump с тестови данни, генерирани чрез въпросната python програма.