ParkSnap

ParkSnap Documentation v1.2

Faculty of IT - University of Moratuwa

With LSEG

ParkSnap

ParkSnap Documentation v1.2

Faculty of IT
University of Moratuwa
With LSEG

Contents

Introduction

Park Snap is a compact application for managing parking lots with multiple parking slots. The system provides various features for managing vehicle parks

Proposed Features

The proposed features of the system are as follows.

- View parking slot occupation
- Reserve parking slots
- Web interface
- User registrations
- Admin panel
- Alerts and updates
- Weather information
- Analytics and Insights

Technologies

The following technologies and software are to be used in the development of the system.

- Technologies
  - HTML
  - CSS/Bootstrap
  - React/Js
  - SpringBoot
  - MySQL
- Software
  - Postman
  - Canva
  - Figma
  - IntelliJ IDEA
  - phpMyAdmin
  - MySQL Workbench
  - Git

Important links

- FigJam: The initial concepts, plans, and designs can be found in our FigJam board [Here]
- UX and wireframe design on Canva [Here]
- GitHub Repository [Here]
- Resources on GitHub Repository [Here]

Technical Specifications

ParkSnap is designed with a react frontend and a spring boot backend connected with a REST API.

Front-end

ParkSnap frontend is implemented with React. The initial UI/UX wireframe and design flow have been constructed with Canva. The UI design has been implemented with Figma. The UI has been developed with a combination of HTML, and CSS alongside Bootstrap.

Requests are handled with the Axios library. The authenticated sessions are managed using local storage features. Each request includes auth headers using an axios instance throughout the project.

Back-end

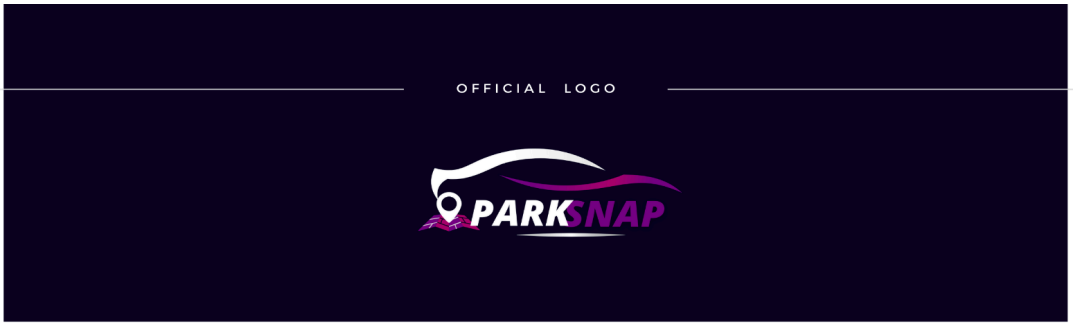
ParkSnap Backend is implemented with SpringBoot 3 on Java 21 with Maven. The system is compatible with a MySQL Database. The system utilizes spring data JPA for database access.

The system uses Spring Security for authentication. Authentication is handled using a JWT bearer token approach.

Brand Guidelines

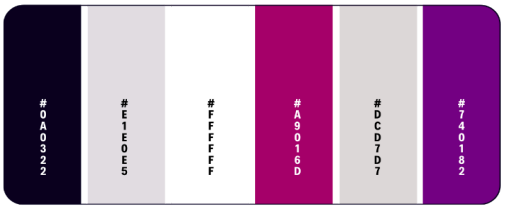
These styles are to be used throughout the project.

OFFICIAL LOGO



ALTERNATIVE LOGO

COLOR PALETTE



FONTS

FONT 1

MONTSERRAT

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890

FONT 2

Raleway

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890

MOODBOARD INSPIRATION

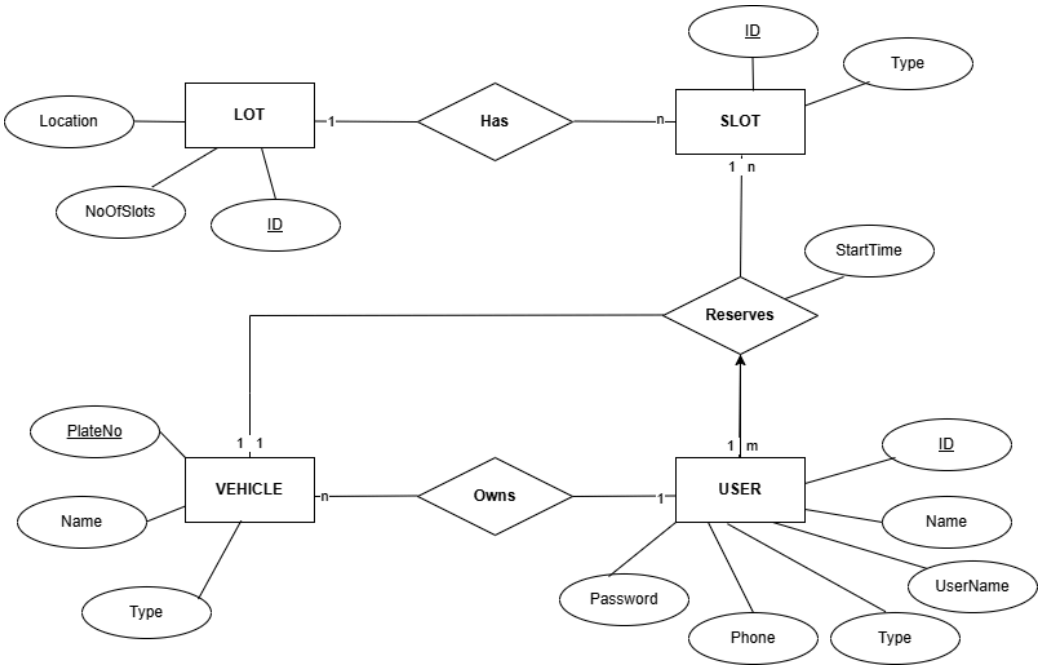
Database Specifications

The database is designed to account for multiple entities regarding the parking system. These entities are

- User
- Lot
- Slot
- Vehicle

Database design

The initial database schema was designed with the help of ER diagrams.



The design was originally drawn on a whiteboard [files], and then on draw.io [design files].

Database Schema

The database schema was derived from the ER diagrams.

Revision 1.0

The following are the relations (tables) decided for the application.

```
Lot(lot_id [PK],location,no_of_slots)
Slot(slot_id [PK], type, lot_id[FK])
User(userId[PK], name, type, phone, username, password)
Vehicle(vehicleId [PK], userId [FK], name, vehicle_type, license_plate)
Reservation(slotId [PK][FK], userId [PK][FK], vehicleId [PK][FK],
duration, startTime)
```

Additional tables to store user role names in the database. It can also implemented in the front end or in configs. It's optional on this scale but should be implemented if possible depending on the time frame.

```
UserType(typeId [PK], typeName)
SlotType(typeId [PK], typeName)
```

Note that there have been some changes since the initial whiteboard ER design

- Slot -> isAvailable; attribute present in the original whiteboard ER design should be derived (by checking the Reservations table) and thus removed from the schema. This is to minimize redundancies. otherwise, the same data is written in two places, which may lead to inconsistencies in making room for
- Vehicle -> vehicleId; is the PK. plate is now a string and should be a normal attribute.

Revision 2.0

The database schema has been changed to avoid some potential problems.

```
Lot(lot_id [PK], location, no_of_slots)
Slot(slot_id [PK], type_id [FK], lot_id[FK])
User(user_id[PK], name, type_id [FK], phone, username, password)
Vehicle(vehicle_id [PK], user_id [FK], name, vehicle_type, license_plate)
Reservation(reservation_id [PK], slot_id [FK], user_id [FK], vehicle_id
[FK], duration, start_time)
UserType(type_id [PK], type_name)
SlotType(type_id [PK], type_name)
```

This allows any reservations to be stored in the database with minimal redundancy. Additionally, it is now possible to store history reservations. Furthermore, it's not necessary to delete records after the day. Having records of past reservations allows statistics to be processed to be displayed on the dashboards. Queries should be programmed to retrieve reservation information for a specific day.

UserType and SlotType are now in tables for easiness of maintaining clarity and consistency throughout the program.

Revision 3.0

The database schema has been changed to maintain vehicle types in a table. An additional relation has been added to store contact requests.

```
lot(lot_id [PK], location, no_of_slots)
slot(slot_id [PK], type_id [FK], lot_id[FK])
user(user_id[PK], name, type_id [FK], phone, username, password)
vehicle(vehicle_id [PK], user_id [FK], name, type_id, license_plate)
reservation(reservation_id [PK], slot_id [FK], user_id [FK], vehicle_id
[FK], duration, start_time)
user_type(type_id [PK], type_name)
slot_type(type_id [PK], type_name)
vehicle_type(type_id [PK], type_name)
contact_form(user_code [PK], user_email, user_mesasge, user_name,
user_phone)
```

`vehicle_type` is now in a table to maintain clarity and consistency throughout the program. `contact_form` is used to store data submitted on the Contact Us page.

Database Implementation in Application

The database has been implemented in the application using Spring Data JPA. Entity classes have been programmed for each of the above entities.

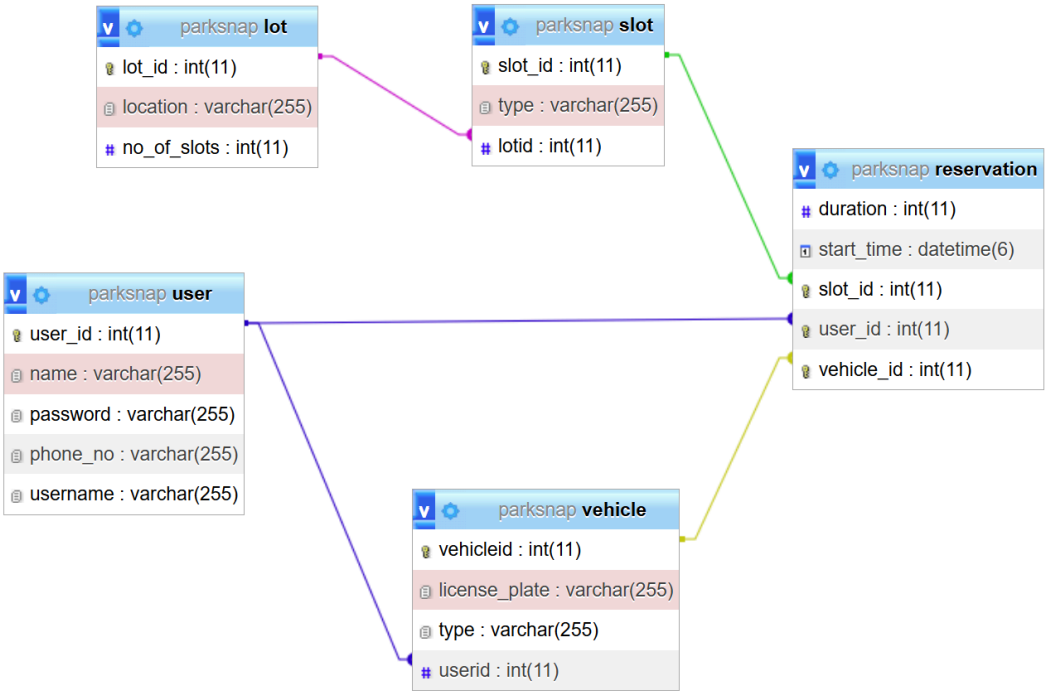
The system automatically creates relations with all the primary and foreign keys on the first established connection with the database.

Note: Make sure the database is empty or made with the same schema when connecting it to the system for the first time. Otherwise, there might be conflicts.

All the database-related resources can be found in our GitHub repository [here].

Revision 1.0

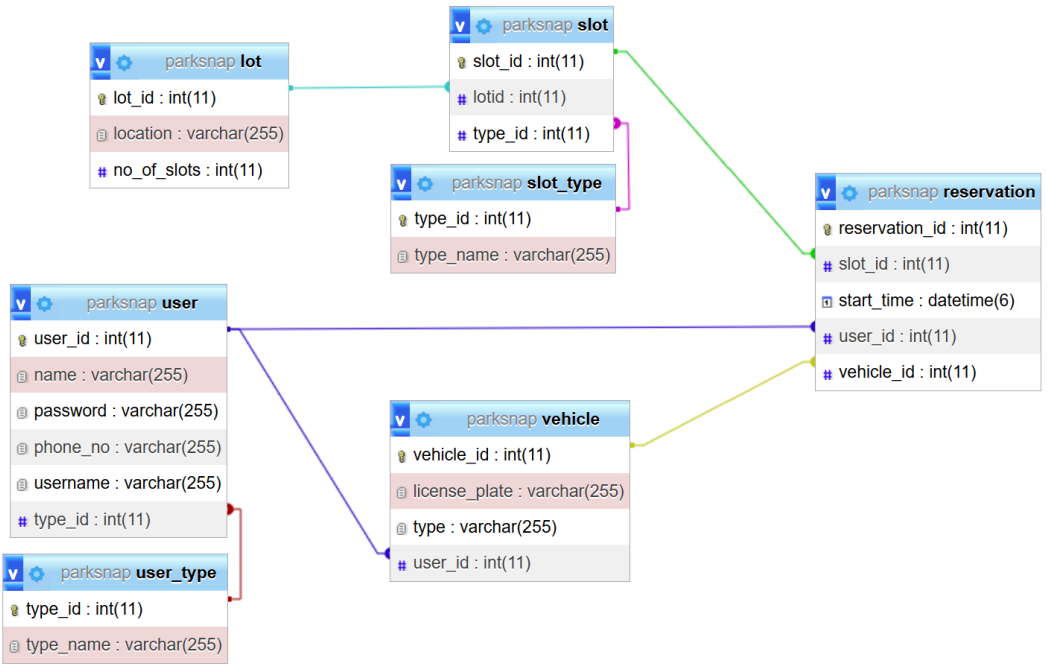
The database schema of the database automatically generated by the system can be summarized in the following image. The image has been generated by phpMyAdmin, a database management tool.



The database specifications generated by phpMyAdmin for revision 1.0 can be found in our GitHub repository [here]. The specifications include all the relations, attributes, data types, keys, and other important aspects.

Revision 2.0

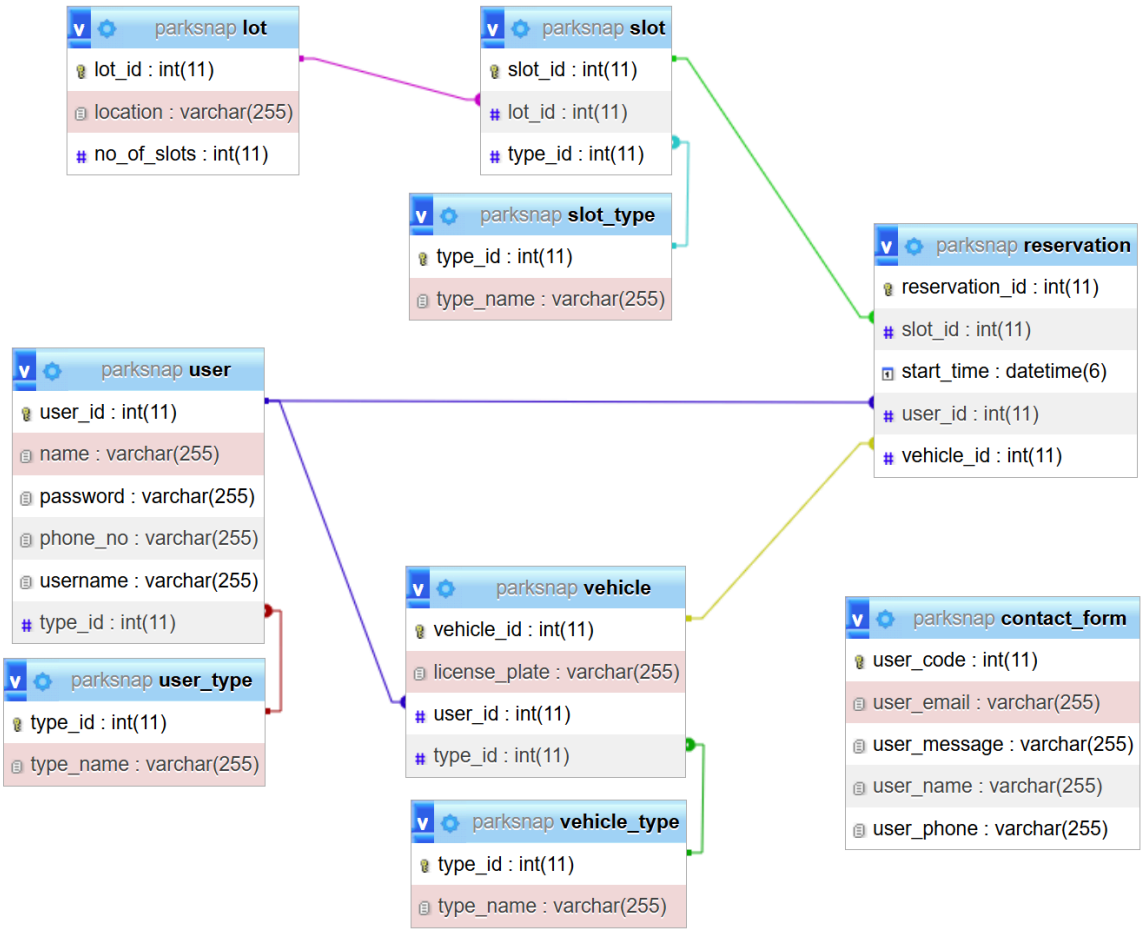
The database schema of the revision 2 is summarized in the following image. The image has been generated by phpMyAdmin.



The database specifications generated by phpMyAdmin for revision 2.0 can be found in our GitHub repository [here]. The specifications include all the relations, attributes, data types, keys, and other important aspects.

Revision 3.0

The database schema of the revision 3 is summarized in the following image. The image has been generated by phpMyAdmin.



The database specifications generated by phpMyAdmin for revision 2.0 can be found in our GitHub repository [here]. The specifications include all the relations, attributes, data types, keys, and other important aspects.

Further improvements

The system could be enhanced with certain enhancements and features. Some of these are mentioned here.

System-Wide

- A mobile number OTP-based login system would be more suitable.

Frontend

Improvements

- Security
    - Currently, the system stores auth tokens in local storage. More secure implementation would be preferable.
    - User type_id is still submitted in the registration form. It should be secured by making it backend only.
- Features
    - Right now, only 2 vehicles can be added on registration. It should be improved to add a list of vehicles.
- Improvements
    - reserveSlot page does not refresh in real-time upon reservation deletion by admin. Instead, just reload the page. This can be improved.
    - In the reserveSlot page, GetSlotData is done by sending multiple getSlotDataById requests and filtering out 403 responses. This should be improved by adding a singular request to fetch all slots reserved by a user. Or, get slot status should be improved. The current approach may rarely render partially correct and partially incorrect information.
- Known issues
    - Occasionally getting Vehicle data is not an iterative error on the user profile page. Reloading the page fixes the issues. Not reproducible yet.

Backend

- Improvements
    - Using email instead of a username would be better.
    - deleteReservation endpoint should be secured to allow only reserved users to delete a reservation
- Known issues
    - Total slots, Occupied slots, and Available slots do not compensate for today

The document end reached.

ParkSnap Documentation v1.2

Faculty of IT
University of Moratuwa
With LSEG