Parkour - Parking Assistant

Project Supervisor

Engr. Abdul Rahman

Project Team

Mahad Khalid K180187
Abdullah Raheel K180170
Ammar Nasir K181037

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science

FAST SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
KARACHI CAMPUS
July 2021

Project Supervisor	**Engr. Abdul Rahman**
Project Team	Abdullah Raheel K180170 Mahad Khalid K180187 Ammar Nasir K181037
Submition Date	13 July 2022

Engr. Abdul Rahman ____________________________

Supervisor

Dr. Zulfiqar Ali Memon ____________________________

Head of Department

FAST SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
KARACHI CAMPUS
July 2021

Acknowledgements

Throughout these 8 months we have recieved tremendous support from our Supervisor **Engr. Abdul Rahman**. From the start **Engr. Abdul Rahman** helped groom and shape our idea to the best of his ability. He has also provided us with the resources to learn and to grow. We would like to thank **Engr. Abdul Rahman** for his time and effort.

We are immensely thankful to **Sir. Sayed Yousaf** for his guidance not only as a great teacher but also as a career counselor. He has been a great mentor to us, giving us the guidance we needed in many areas of life.

We would also like to thank **Sir. Danish** and **Sir. Zeeshan** for the materials they provided to us for FYP and also helping us with any FYP related queries.

We would like to thank some of the best Teachers in the University for their guidance and grooming us to people we are today. Some of the teachers are but not limited to: **Ms. Anum Qureshi**, **Ms. Farah Sadia**, **Ms. Nida Fatima**, **Dr. Zulfiqar Ali Memon** and **Dr. Jawwad Shamsi**.

I would also like to thank my Project Team for their unyielding grit in the face of dificulties. Their hard work and dedication has helped us to achieve our goals.

Abstract

In small to large metropolitan cities alike, finding parking is a principal problem.
Over population of major cities in Pakistan means high density of vehicles which the city cannot accommodate to begin with. This leads to lack of parking spots which in turn make parkers spend their valuable time to find a spot, increase their fuel consumption and contribute to traffic congestion.

To tackle this problem, we came up with the idea of an app with an inbuilt map that shows parking spots all around the city to help drivers find parking spots quickly, saving time and fuel. The app will also show open parking spots close to drivers where they can easily park and also show the route to these spots. Drivers can also see how much fuel it will take to reach said spot. Since the available parking spots are visible on the map, drivers can even pre-book the spots beforehand.

The project aims to provide parkers an application to find parking spots using their phone in real time and save them the trouble of looking for a parking spot manually. Meanwhile, the app gives the parking provider an opportunity to run a small business by putting their unused parking spot(s) up for sale. Furthermore, the project aims to provide residents who own a car but do not have a suitable parking spot near their home or residents who have trouble finding a parking spot near their workplace, a long-term parking solution by giving them the option to purchase a parking spot nearby.

Contents

List of Figures

List of Tables

Introduction

1.1 Background

Commuting is a daily part of most people's lives and for many people who own a private vehicle finding a suitable and appropriate parking spot is a daily struggle. It becomes even more difficult to find a spot when hundreds of people come out for an event or holiday festivals. The fact that no private vehicle is perpetually in motion; most private vehicles spend most of their time at rest, either during working hours or over the night, means that there should be two places for every car in the city to be parked in. The two places should be at the both ends of every trip. However the number of vehicles keeps increasing, while parking space has remained constant or reduced due to a growing population.This imbalance between parking supply and parking demand has been considered as the main reason for metropolis parking problems.

1.2 Problem

Lack of parking spots truly is a global issue. As global living standards rise and urbanization accelerates, especially in India and China, cities around the world are seeing huge spikes in motor vehicle ownership accompanied by demand for parking. In India, the number of private automobiles grew nearly 400% between 2001 and 2015, going from 55 million to 210 million. In China, as of 2017, there was a shortage of 50 million parking spaces, according to the central government.Cities face immense challenges from climate change and rising heat, increased urbanization and housing affordability.

Besides the lack of parking spots, there is another problem that needs to be addressed. Surprisingly, there are metropolitan cities which incorporate plenty of parking spaces. For example, one study shows that there are 2.2 million total parking spaces in Philadelphia and 1.85 million in New York City. So the problem is not just about the general lack of parking spaces but the lack of real-time data about vacant ones. Due to this, drivers are often left frustrated and spend too much time searching for a spot. In peak hours drivers in large cities have to circle around the desired destination to find a place where they can leave their vehicles. Those who run out of time might park illegally. By bouncing between parking spaces that are full, drivers become restless and may choose to park illegally or leave altogether.

An unregulated tariff structure is another issue which leads to a scarcity of parking spaces. For example, in Indian and Pakistani metros, parking is either free or minimally priced, the fees being unregulated for many years now. Because parking prices stop increasing after a certain period of time, the longer one stays in a parking space, the less one has to pay. This is a problem because parking space is a scarce commodity today and should come with a price. A low parking price encourages more vehicles on the road, contributing to air and noise pollution.

1.3 Proposed System

The project aims to solve all the problems addressed above by attempting to create a self-managing peer to peer parking system where people who have unused parking spots can both earn and contribute by making their spots available for others to park and charge a fare for the spot. The system will embody a map that provides reliable, real-time data that allows drivers to choose from the available spots or even pre-book a parking spot beforehand.

1.4 Pricing

The system will further implement a regulated tariff system. One could say the best way to manage the parking is by charging the right price for it. This can be done by using demand to price parking and optimize occupancy. If the price is too high and spaces remain vacant, operators lose revenue, nearby shops lose customers, employees lose jobs, and governments lose tax revenue. If the price is too low and no spaces are available, it leads to traffic congestion and chaos. Pricing can thus be a very effective tool for the management of travel demand as a whole. Deploying a cost-effective parking management and guidance solution ultimately generates more revenue for a city, as existing parking spaces are properly monetized. Drivers are more motivated to pay for a spot when they know they'll be able to find it quickly, without having to circle around in vain. The awareness by drivers that all spaces are monitored by a modern system further increases the understanding that it is fair to pay for the valuable public space and service.

1.5 Conclusion

Due to the growing population, the growth of private vehicles is not going to halt and in turn the demand for parking. And increasing the parking supply by creating more and more parking spaces is not a feasible solution. Because land resources are limited, it isn't possible to conveniently plan parking spaces according to demand. Instead of increasing available parking spaces, an effective technology-based solution must be employed to optimize the use of available spaces and to provide drivers with a real-time map of available spaces and allow drivers to reserve spots based on their convenience. Not only does this save drivers time spent searching for a spot, but it also reduces environmental degradation resulting from congestion caused by parking. Finally, the parking system plays a key role in the metropolitan traffic system, and lacking it shows closed relations with traffic congestion, traffic accidents, and environmental pollution. And even with all of these factors, the parking problem is an often-overlooked aspect of urban planning and transportation. Therefore an efficient parking system can improve urban transportation and city environment besides raising the quality of life for citizens.

Related Work

To provide parking spots it is absolutely necessary to rely on either people or sensors. If the application is dependent on people it must either implement a peer to peer system like **Parkour** or use a smart phone to provide the parking spots. If the application is dependent on sensors, it must be able to provide the parking spots without the need of people.

The other method of determined the parking spots is to use a combination of sensors and people. This is called **Sensor-based Parking** [1] This type of methodology uses both the sensor and data from people to accurately predict of the spots are empty.

Parking Spots can also be determined using data from the general public. This is called **Crowd-Sourced Parking** [2] This type of methodology uses people to verify if the parking spots are indeed empty.

One unique method of determining if parking spots were empty or occupied was through the use of ultrasonic sensors [3]. This method was found to be the most accurate and efficient method of determining if parking spots were empty or occupied without entailing the heavy cost of image recognition based parking.

The most effective method of determining the parking without relying on people is by using some sort of image recognition. This is called **Image-Detection-based Parking** [4] This is not only very expensive to implement but also comes with it's challenges of accurate image detection.

Parkour uses peer to peer communication to determine the parking spots. This is a peer to peer system that is implemented using a smart phone. This is a system that is designed to be used by people to communicate with each other.

Requirements

Parkour - Parking Assistant is an Android / Web based application that requires and active internet connection. The application also requires a GPS to be able to track the user's location. The location must be enabled on the device for the application to work properly.

Services required on the phone:

- The application requires an active internet connection, To connect to google maps and google services. The application requires an active internet connection to download the map data. without an active internet connection, the application will not be able to talk to the server.
- Gps / Location services must be enabled on the device to detect the current location of the user and show them parking spots near them in a certain radius.

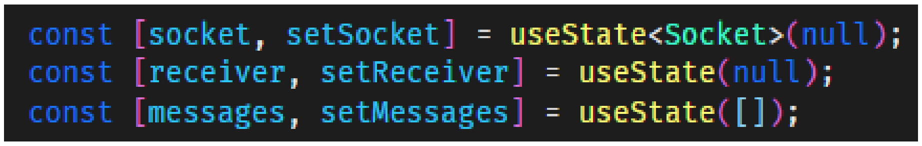
Design

A significant amount of time was spent on the design of the system. A careful consideration was given to working of the frontend and the backend. The system was designed to be as modular as possible. The system was designed to be able to be used in multiple ways. Special attention to the design of the frontend was given including contrast ratios, colors and rhythm was given. User experience was also given a high priority.

4.1 Design Patterns

- State

  The useState hook in react exposes a way to change the state of the component, this allows us to re-render the ui whenever change the state of the component. The functional component might behave differently depending on the state of the component. This is how a the state pattern is used here.

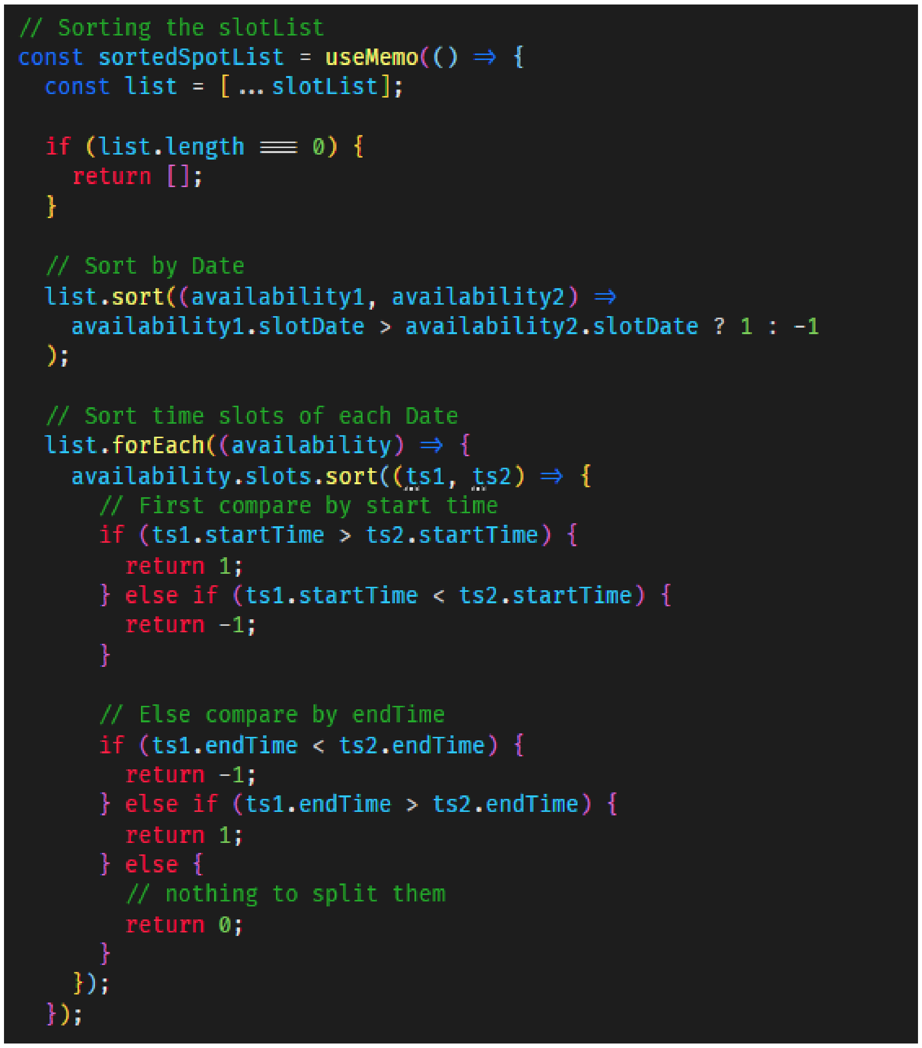


```
const [socket, setSocket] = useState<Socket>(null);
const [receiver, setReceiver] = useState(null);
const [messages, setMessages] = useState([]);
```

Figure 4.1: State Pattern

- Memento

  The useMemo hook allows us to memoize the output of the component. This is how the memento pattern is used here. Here we are memoizing the slotList. This is done to avoid re-calculating and re-sorting the slotList when the component re-renders.



```
// Sorting the slotList
const sortedSpotList = useMemo(() => {
  const list = [...slotList];

  if (list.length === 0) {
    return [];
  }

  // Sort by Date
  list.sort((availability1, availability2) =>
    availability1.slotDate > availability2.slotDate ? 1 : -1
  );

  // Sort time slots of each Date
  list.forEach((availability) => {
    availability.slots.sort((ts1, ts2) => {
      // First compare by start time
      if (ts1.startTime > ts2.startTime) {
        return 1;
      } else if (ts1.startTime < ts2.startTime) {
        return -1;
      }

      // Else compare by endTime
      if (ts1.endTime < ts2.endTime) {
        return -1;
      } else if (ts1.endTime > ts2.endTime) {
        return 1;
      } else {
        // nothing to split them
        return 0;
      }
    });
  });
```

Figure 4.2: Memento Pattern

4.2 Frontend

4.2.1 Wireframe

Before the development of the UI the design of the wireframe was considered. A wireframe of the UI was created that gave the general understanding of how the UI will be broken into components.

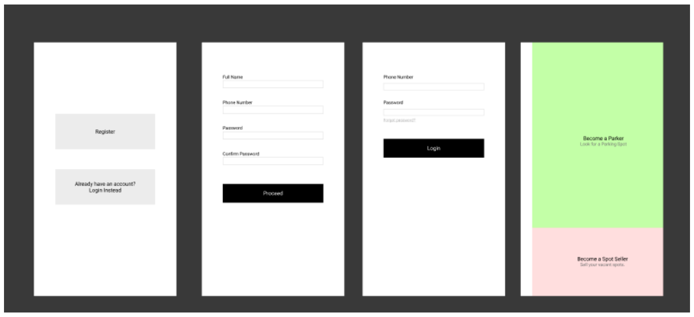


Figure 4.3: Homepage Wireframe

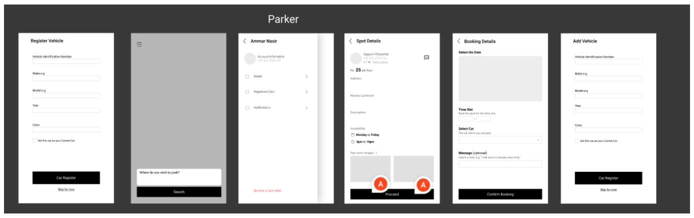


Figure 4.4: Parker Wireframe

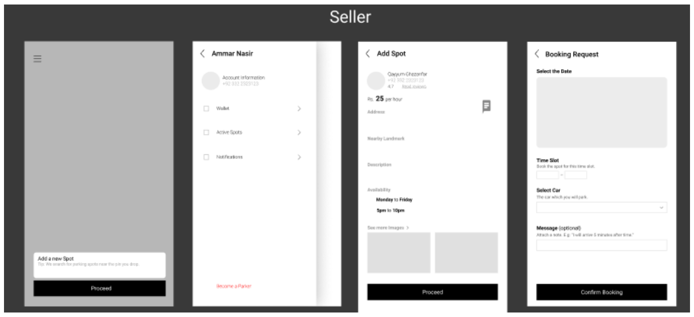
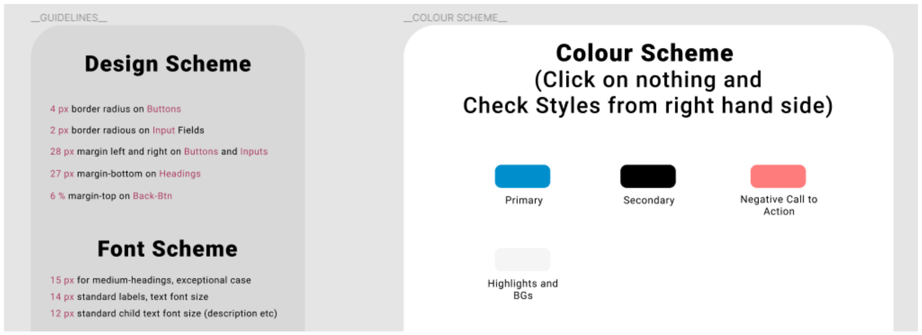


Figure 4.5: Seller Wireframe

4.2.2 UI

At the start of UI design the Font, Colors and Design schemes were solidified to ease the process and keep the consistency throughout the application. The aim was to make the UI look and feel like one cohesive application.



__GUIDELINES__

Design Scheme

4 px border radius on Buttons

2 px border radious on Input Fields

28 px margin left and right on Buttons and Inputs

27 px margin-bottom on Headings

6 % margin-top on Back-Btn

Font Scheme

15 px for medium-headings, exceptional case

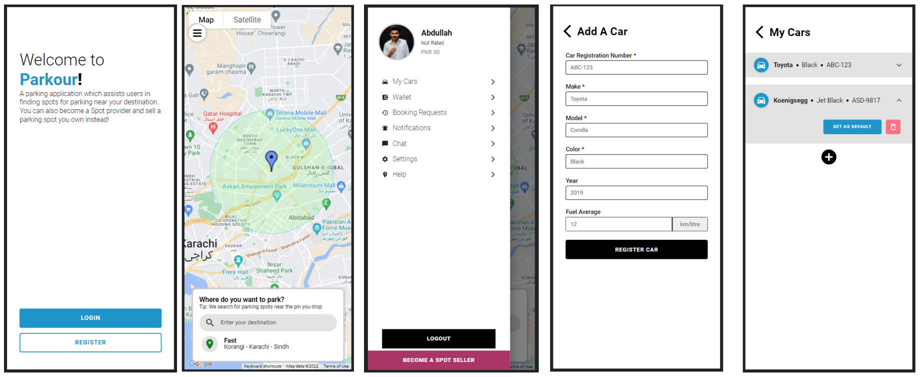
14 px standard labels, text font size

12 px standard child text font size (description etc)

__COLOUR SCHEME__

Colour Scheme
(Click on nothing and
Check Styles from right hand side)

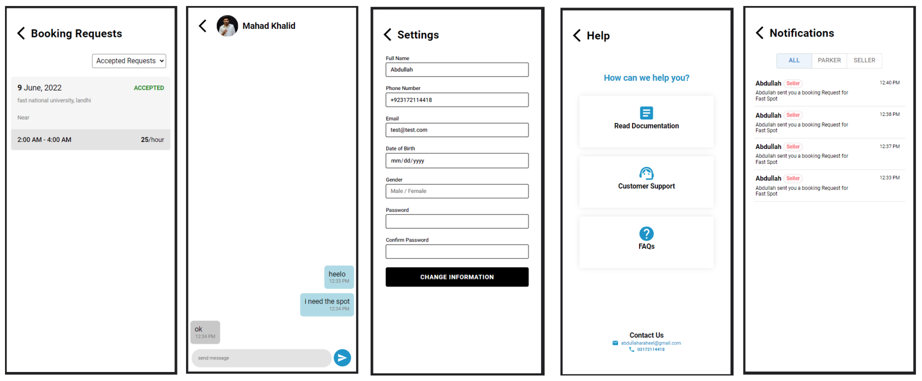
Figure 4.6: Color, Font, Design Schemes



A parking application which assists users in finding spots for parking near your destination. You can also become a Spot provider and sell a parking spot you own instead!

Where do you want to park?
Tip: We search for parking spots near the pin you drop
Enter your destination
Fast
Korangi - Karachi - Sindh

Figure 4.7: UI (1)



9 June, 2022
fast national university, landhi
Near
2:00 AM - 4:00 AM 25/hour

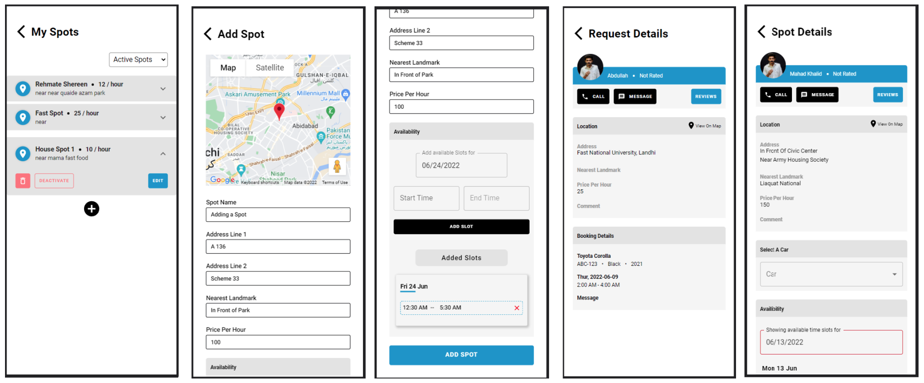
heelo 12:33 PM
i need the spot 12:34 PM
ok 12:34 PM

Full Name: Abdullah
Phone Number: +923172114418
Email: test@test.com
Date of Birth: mm/dd/yyyy
Gender: Male / Female
Password:
Confirm Password:

abdullaharaheel@gmail.com
03172114418

Abdullah Seller 12:40 PM
Abdullah sent you a booking Request for Fast Spot
Abdullah Seller 12:38 PM
Abdullah sent you a booking Request for Fast Spot
Abdullah Seller 12:37 PM
Abdullah sent you a booking Request for Fast Spot
Abdullah Seller 12:33 PM
Abdullah sent you a booking Request for Fast Spot

Figure 4.8: UI (2)



Rehmate Shereen • 12 / hour
near near quaide azam park
Fast Spot • 25 / hour
near
House Spot 1 • 10 / hour
near mama fast food

Spot Name: Adding a Spot
Address Line 1: A 136
Address Line 2: Scheme 33
Nearest Landmark: In Front of Park
Price Per Hour: 100

Add available Slots for 06/24/2022
Start Time
End Time
Fri 24 Jun
12:30 AM – 5:30 AM

Abdullah • Not Rated
Address: Fast National University, Landhi
Nearest Landmark
Price Per Hour: 25
Comment
Toyota Corolla
ABC-123 • Black • 2021
Thur, 2022-06-09
2:00 AM - 4:00 AM
Message

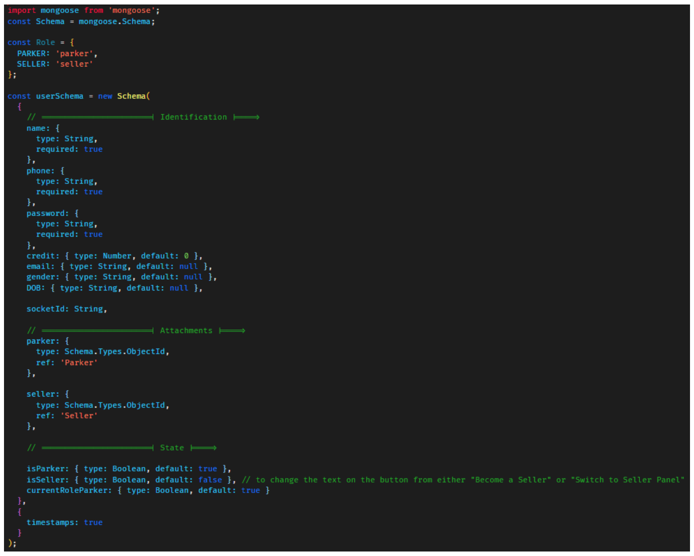
Mahad Khalid • Not Rated
Address: In Front Of Civic Center
Near Army Housing Society
Nearest Landmark: Liaquat National
Price Per Hour: 150
Comment
Car
Showing available time slots for 06/13/2022
Mon 13 Jun

Figure 4.9: UI (3)

4.3 Backend

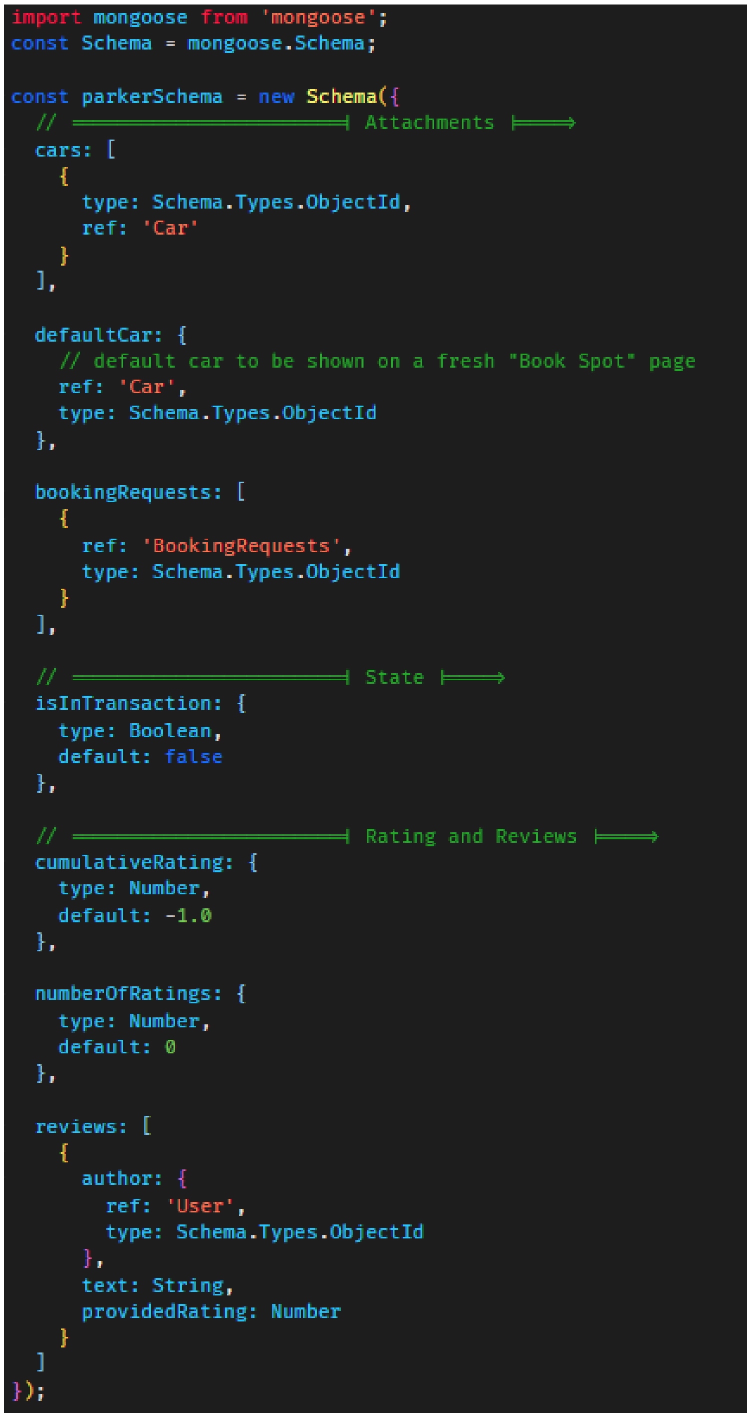
4.3.1 Models

It is important to highlight the models that were designed and used.
Each modelhad been given a careful consideration to make sure that the model is as modular as possible. The models were designed to be able to be used in multiple ways.The models were also designed in such a way that it will reduce redundancy wherever possible.



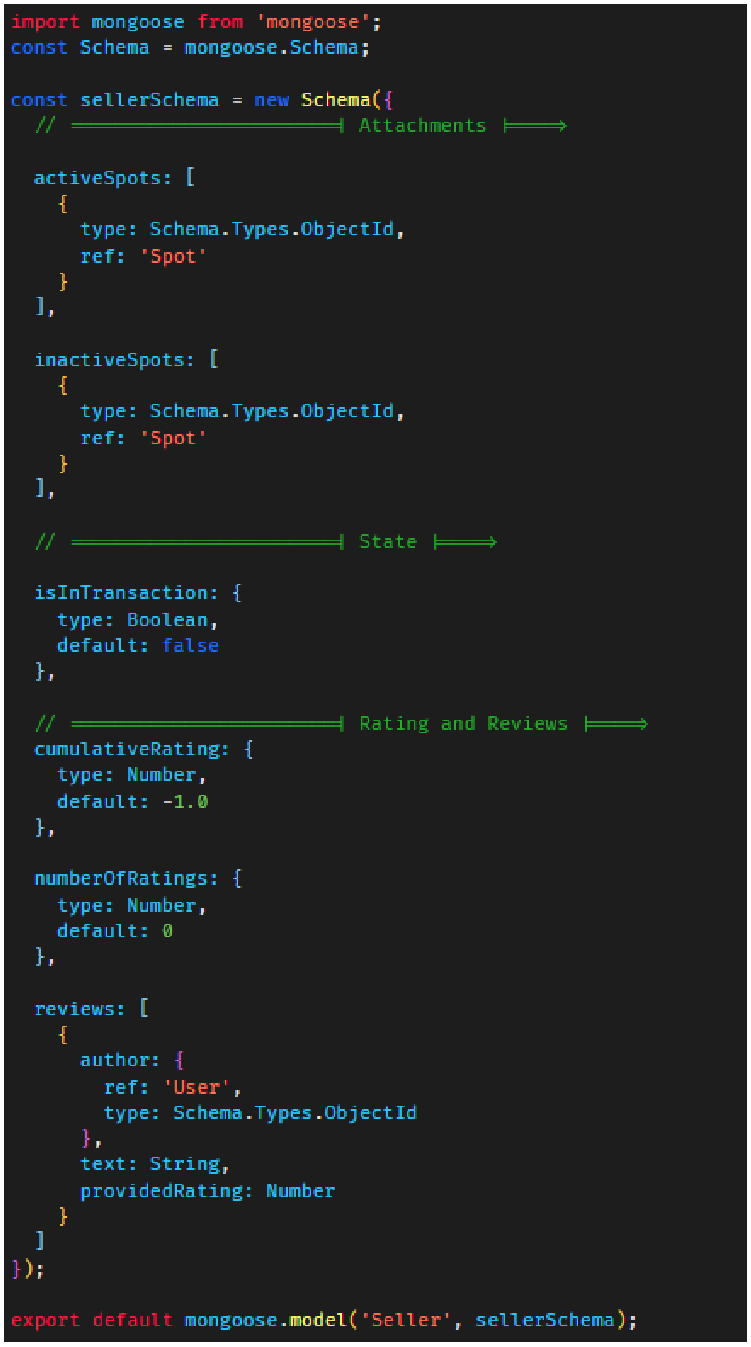
```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const Role = {
  PARKER: 'parker',
  SELLER: 'seller'
};

const userSchema = new Schema(
  {
    // ═══════════════════════╡ Identification ╞════⟹
    name: {
      type: String,
      required: true
    },
    phone: {
      type: String,
      required: true
    },
    password: {
      type: String,
      required: true
    },
    credit: { type: Number, default: 0 },
    email: { type: String, default: null },
    gender: { type: String, default: null },
    DOB: { type: String, default: null },

    socketId: String,

    // ═══════════════════════╡ Attachments ╞════⟹
    parker: {
      type: Schema.Types.ObjectId,
      ref: 'Parker'
    },

    seller: {
      type: Schema.Types.ObjectId,
      ref: 'Seller'
    },

    // ═══════════════════════╡ State ╞════⟹

    isParker: { type: Boolean, default: true },
    isSeller: { type: Boolean, default: false }, // to change the text on the button from either "Become a Seller" or "Switch to Seller Panel"
    currentRoleParker: { type: Boolean, default: true }
  },
  {
    timestamps: true
  }
);
```

Figure 4.10: User Model



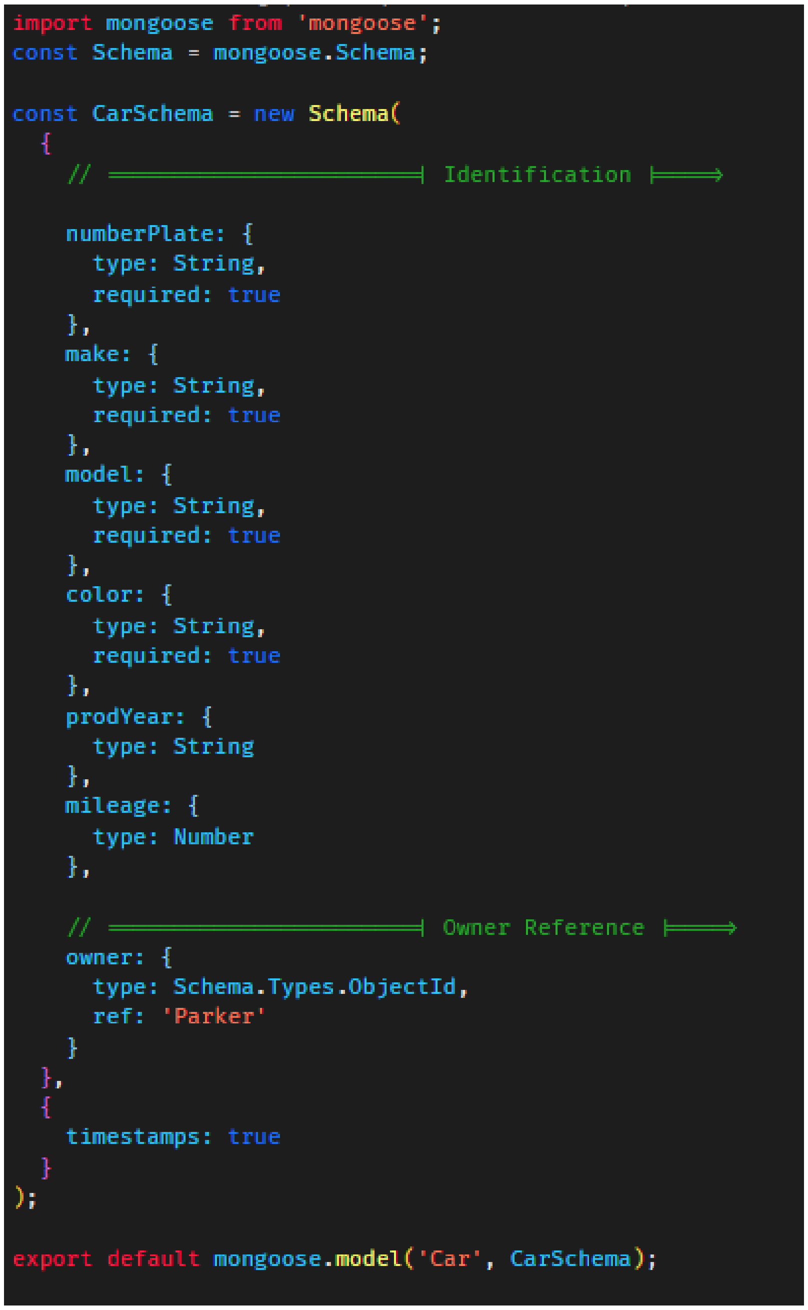
```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const parkerSchema = new Schema({
  // ════════════════════════╡ Attachments ╞════⟹
  cars: [
    {
      type: Schema.Types.ObjectId,
      ref: 'Car'
    }
  ],

  defaultCar: {
    // default car to be shown on a fresh "Book Spot" page
    ref: 'Car',
    type: Schema.Types.ObjectId
  },

  bookingRequests: [
    {
      ref: 'BookingRequests',
      type: Schema.Types.ObjectId
    }
  ],

  // ════════════════════════╡ State ╞════⟹
  isInTransaction: {
    type: Boolean,
    default: false
  },

  // ════════════════════════╡ Rating and Reviews ╞════⟹
  cumulativeRating: {
    type: Number,
    default: -1.0
  },

  numberOfRatings: {
    type: Number,
    default: 0
  },

  reviews: [
    {
      author: {
        ref: 'User',
        type: Schema.Types.ObjectId
      },
      text: String,
      providedRating: Number
    }
  ]
});
```

Figure 4.11: Parker Model



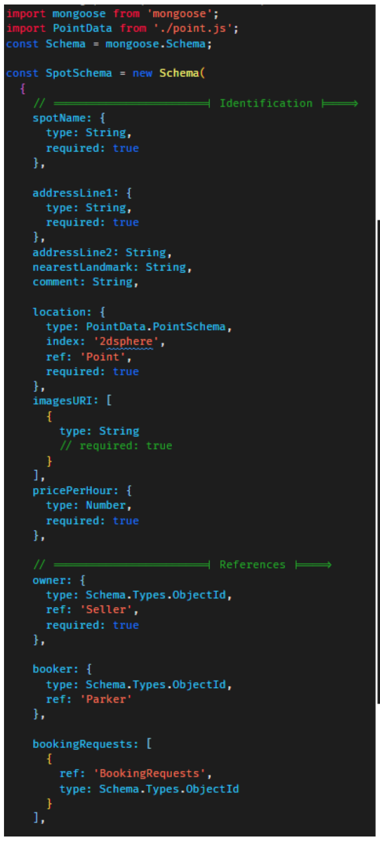
```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const sellerSchema = new Schema({
  // ═══════════════════════════╡ Attachments ╞═════⟹

  activeSpots: [
    {
      type: Schema.Types.ObjectId,
      ref: 'Spot'
    }
  ],

  inactiveSpots: [
    {
      type: Schema.Types.ObjectId,
      ref: 'Spot'
    }
  ],

  // ═══════════════════════════╡ State ╞═════⟹

  isInTransaction: {
    type: Boolean,
    default: false
  },

  // ═══════════════════════════╡ Rating and Reviews ╞═════⟹
  cumulativeRating: {
    type: Number,
    default: -1.0
  },

  numberOfRatings: {
    type: Number,
    default: 0
  },

  reviews: [
    {
      author: {
        ref: 'User',
        type: Schema.Types.ObjectId
      },
      text: String,
      providedRating: Number
    }
  ]
});

export default mongoose.model('Seller', sellerSchema);
```

Figure 4.12: Seller Model

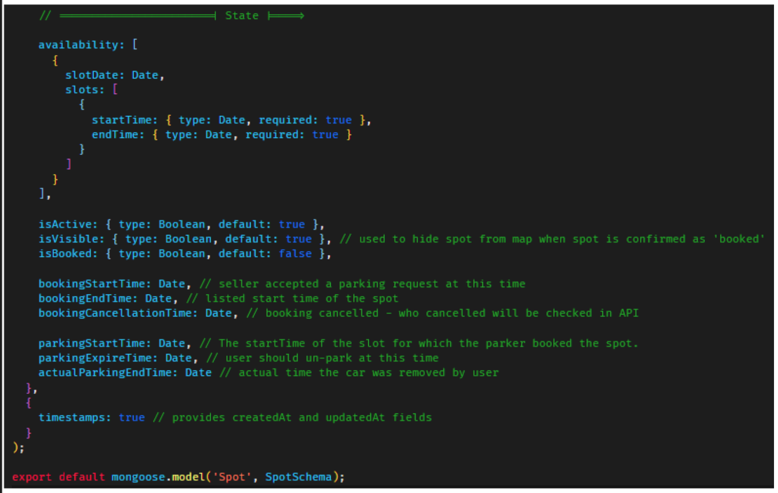


```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const CarSchema = new Schema(
  {
    // ══════════════════════╡ Identification ╞═════⟹

    numberPlate: {
      type: String,
      required: true
    },
    make: {
      type: String,
      required: true
    },
    model: {
      type: String,
      required: true
    },
    color: {
      type: String,
      required: true
    },
    prodYear: {
      type: String
    },
    mileage: {
      type: Number
    },

    // ══════════════════════╡ Owner Reference ╞═════⟹
    owner: {
      type: Schema.Types.ObjectId,
      ref: 'Parker'
    }
  },
  {
    timestamps: true
  }
);

export default mongoose.model('Car', CarSchema);
```

Figure 4.13: Car Model

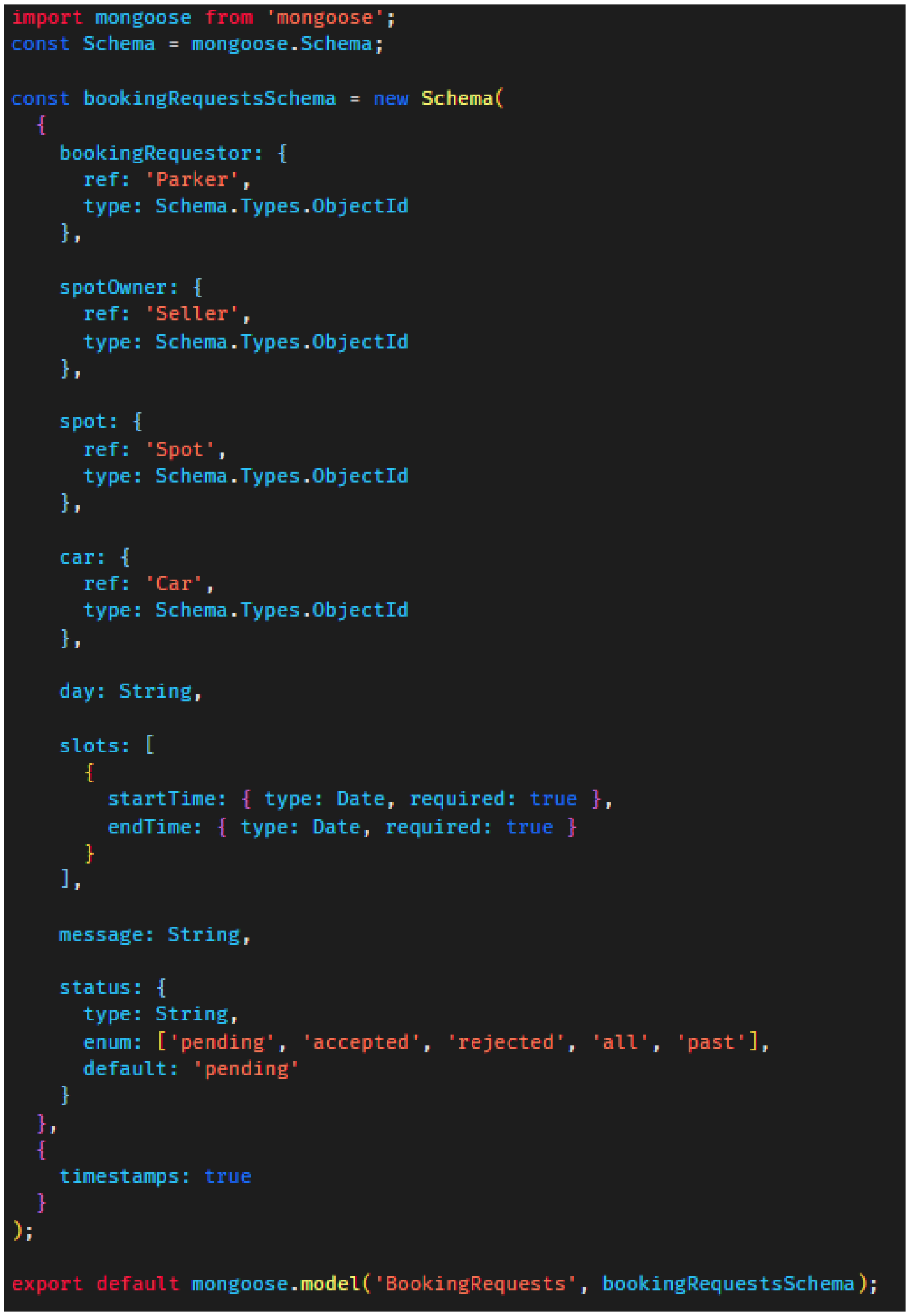


```
import mongoose from 'mongoose';
import PointData from './point.js';
const Schema = mongoose.Schema;

const SpotSchema = new Schema(
  {
    // ═══════════════════════╡ Identification ╞════⟹
    spotName: {
      type: String,
      required: true
    },

    addressLine1: {
      type: String,
      required: true
    },
    addressLine2: String,
    nearestLandmark: String,
    comment: String,

    location: {
      type: PointData.PointSchema,
      index: '2dsphere',
      ref: 'Point',
      required: true
    },
    imagesURI: [
      {
        type: String
        // required: true
      }
    ],
    pricePerHour: {
      type: Number,
      required: true
    },

    // ═══════════════════════╡ References ╞════⟹
    owner: {
      type: Schema.Types.ObjectId,
      ref: 'Seller',
      required: true
    },

    booker: {
      type: Schema.Types.ObjectId,
      ref: 'Parker'
    },

    bookingRequests: [
      {
        ref: 'BookingRequests',
        type: Schema.Types.ObjectId
      }
    ],
```



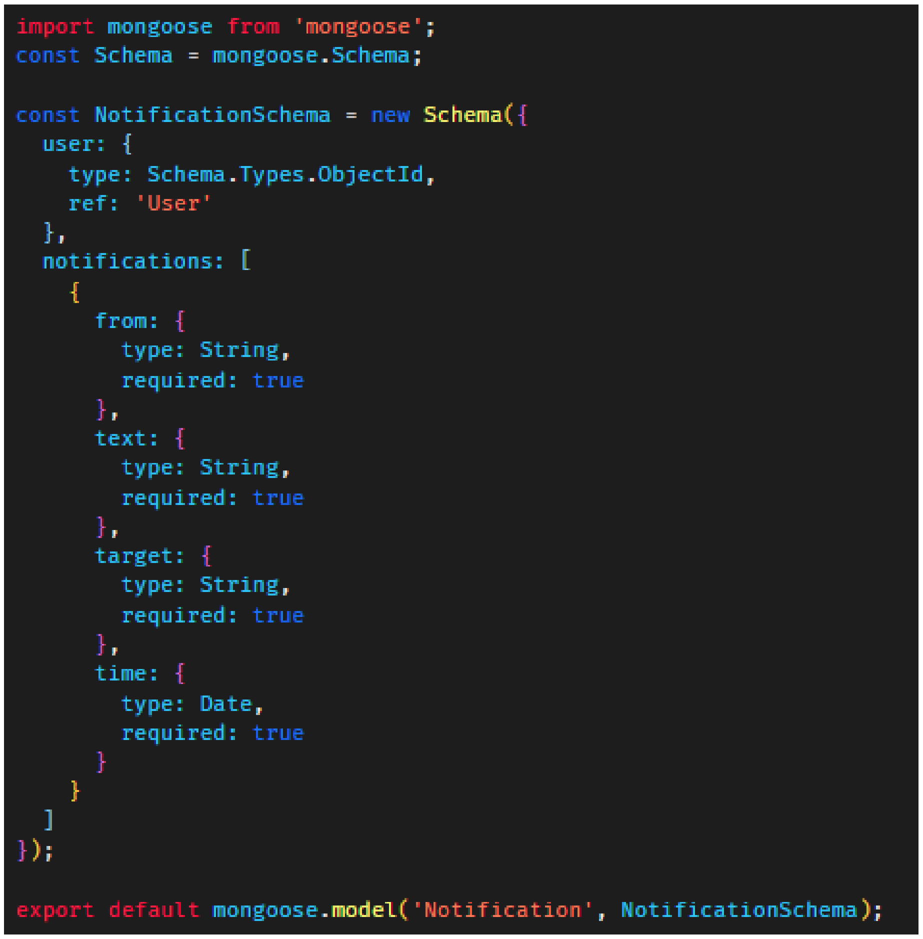
```
    // ═══════════════════════╡ State ╞════⟹

    availability: [
      {
        slotDate: Date,
        slots: [
          {
            startTime: { type: Date, required: true },
            endTime: { type: Date, required: true }
          }
        ]
      }
    ],

    isActive: { type: Boolean, default: true },
    isVisible: { type: Boolean, default: true }, // used to hide spot from map when spot is confirmed as 'booked'
    isBooked: { type: Boolean, default: false },

    bookingStartTime: Date, // seller accepted a parking request at this time
    bookingEndTime: Date, // listed start time of the spot
    bookingCancellationTime: Date, // booking cancelled - who cancelled will be checked in API

    parkingStartTime: Date, // The startTime of the slot for which the parker booked the spot.
    parkingExpireTime: Date, // user should un-park at this time
    actualParkingEndTime: Date // actual time the car was removed by user
  },
  {
    timestamps: true // provides createdAt and updatedAt fields
  }
);

export default mongoose.model('Spot', SpotSchema);
```

Figure 4.14: Spots Model



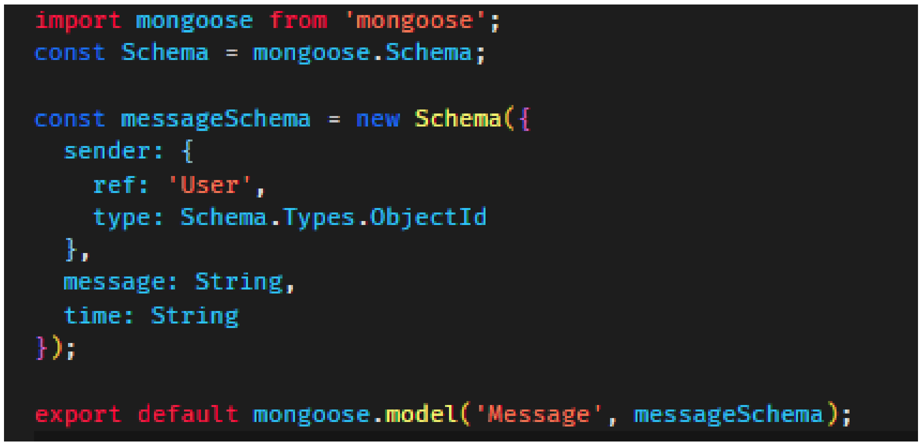
```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const bookingRequestsSchema = new Schema(
  {
    bookingRequestor: {
      ref: 'Parker',
      type: Schema.Types.ObjectId
    },

    spotOwner: {
      ref: 'Seller',
      type: Schema.Types.ObjectId
    },

    spot: {
      ref: 'Spot',
      type: Schema.Types.ObjectId
    },

    car: {
      ref: 'Car',
      type: Schema.Types.ObjectId
    },

    day: String,

    slots: [
      {
        startTime: { type: Date, required: true },
        endTime: { type: Date, required: true }
      }
    ],

    message: String,

    status: {
      type: String,
      enum: ['pending', 'accepted', 'rejected', 'all', 'past'],
      default: 'pending'
    }
  },
  {
    timestamps: true
  }
);

export default mongoose.model('BookingRequests', bookingRequestsSchema);
```

Figure 4.15: Booking Request Model



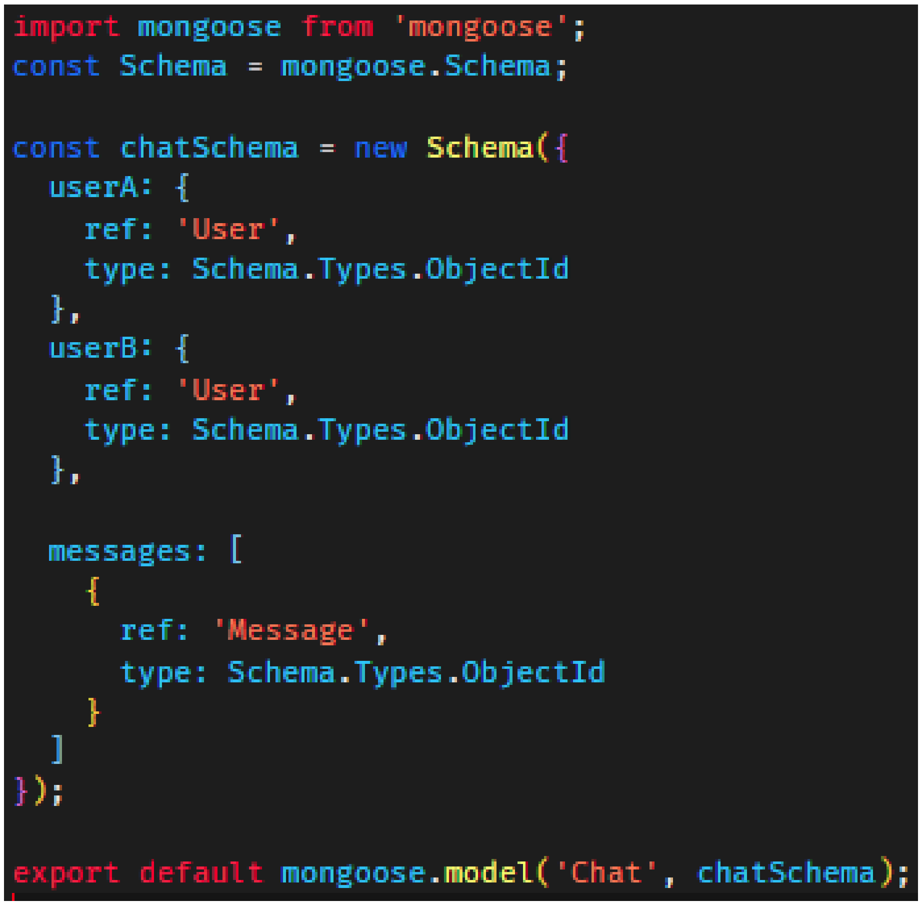
```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const NotificationSchema = new Schema({
  user: {
    type: Schema.Types.ObjectId,
    ref: 'User'
  },
  notifications: [
    {
      from: {
        type: String,
        required: true
      },
      text: {
        type: String,
        required: true
      },
      target: {
        type: String,
        required: true
      },
      time: {
        type: Date,
        required: true
      }
    }
  ]
});

export default mongoose.model('Notification', NotificationSchema);
```

Figure 4.16: Notification Model



```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const messageSchema = new Schema({
  sender: {
    ref: 'User',
    type: Schema.Types.ObjectId
  },
  message: String,
  time: String
});

export default mongoose.model('Message', messageSchema);
```

Figure 4.17: Message Model



```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const chatSchema = new Schema({
  userA: {
    ref: 'User',
    type: Schema.Types.ObjectId
  },
  userB: {
    ref: 'User',
    type: Schema.Types.ObjectId
  },

  messages: [
    {
      ref: 'Message',
      type: Schema.Types.ObjectId
    }
  ]
});

export default mongoose.model('Chat', chatSchema);
```

Figure 4.18: Chat Model

Described the models in the following table.

Model	Description
User	The model that contains all the information about the user.
Parker	The model that contains all the information about the parker.
Seller	The model that contains all the information about the seller.
Car	The model that contains all the information about the car.
Spot	The model that contains all the information about the spot.
Booking	The model that contains all the information about the booking.
Notification	The model that contains all the information about the notification.
Message	The model that contains all the information about the message.
Chat	The model that contains all the information about the chat.

Table 4.1: Models

Implementation

testing

5.1 Architecture

5.2 Frontend

5.3 Backend

5.4 Database

5.5 Behavior

Testing and Evaluation

lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Conclusion

lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

References

[1] Wenyu Cai, Dong Zhang, and Yongjie Pan. Implementation of smart parking guidance system based on parking lots sensors networks. In *2015 IEEE 16th International Conference on Communication Technology (ICCT)*, pages 419–424. IEEE, 2015.

[2] Ellen Mitsopoulou and Vana Kalogeraki. Efficient parking allocation for smartcities. In *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*, pages 265–268, 2017.

[3] Wan-Joo Park, Byung-Sung Kim, Dong-Eun Seo, Dong-Suk Kim, and Kwae-Hi Lee. Parking space detection using ultrasonic sensor in parking assistance system. In *2008 IEEE intelligent vehicles symposium*, pages 1039–1044. IEEE, 2008.

[4] Rahayu Yusnita, Fariza Norbaya, and Norazwinawati Basharuddin. Intelligent parking space detection system based on image processing. *International Journal of Innovation, Management and Technology*, 3(3):232–235, 2012.