Pick & Park (Parking Garage)

1. **Adriane San Gaspar**
    a. Student ID: 028648163
    b. CSULB Email: adriane.sangaspar012@student.csulb.edu
2. **Daphne Rios**
    a. Student ID: 031013643
    b. CSULB Email: daphne.rios01@student.csulb.edu
3. **Flavien Jean Guy Aurelien Maameri**
    a. Student ID: 033780732
    b. CSULB Email: flavienjeanguyaurelien.maameri01@student.csulb.edu
4. **Mia Alvarez**
    a. Student ID: 030261697
    b. CSULB Email: mia.alvarez03@student.csulb.edu
5. **Nicolas Piker**
    a. Student ID: 029966545
    b. CSULB Email: nicolas.piker01@student.csulb.edu
6. **Porter Clevidence**
    a. Student ID: 029238207
    b. CSULB Email: porter.clevidence01@student.csulb.edu
7. **Abram Ibarra**
    a. Student ID: 031652736
    b. CSULB Email: abram.ibarra01@student.cuslb.edu

Preface

Version	Date Changes
1.0	3/15/2025 Initial Version
2.0	5/9/2025 Final Version

Purpose

This document serves as a comprehensive guide for the development and understanding of the software project titled "Pick&Park".

Audience

The intended audience of this document includes project stakeholders, developers, testers, project manager, parking lot owners/operators, and end users(drivers/parkers).

Introduction

Pick&Park is a software system that helps users conveniently find and reserve parking spots in advance, reducing the time spent searching for parking. The system provides real-time availability updates, secure reservations, and user-friendly payment options.

Project Overview

"Pick&Park" is a mobile based application that simplifies the process of finding and reserving parking spots. It secures and streamlines parking space tracking, reservation, management, and reporting.

Project Goals

- Enhance User Convenience.
  - Provide a seamless user experience for finding and reserving parking spots with minimal effort.
- Improve Parking Efficiency.
  - Reduce traffic congestion caused by drivers searching for parking.
- Ensure Secure Transactions.
  - Implement robust security measures for payment processing.
  - Offer trusted third-party payment options to enhance user confidence.

Glossary

- **Spot Inventory**: The stock of parking spots available for reservation.
- **API**: Application Programming Interface.

User Requirements and Use Cases

User Stories

(A collection of user stories that apply to the project.)

1. As a registered parker, I want to log in securely so that I can easily and unquestionably reserve a parking spot.
2. As a prospective parker, I want to search for available parking spots so that I can easily find a place to park.
3. As a prospective parker, I want to reserve a parking spot so that I am guaranteed a space until arrival.
4. As a parking lot owner, I want to automatically update the availability of parking spots so that users see up to date information.
5. As a prospective parker, I want to extend my parking time remotely so that I don't get fined.
6. As a user, I want to view my reservation history so that I can keep track of my parking expenses.
7. As a parking lot owner, I want user payment methods to be valid so that I receive proper compensation in a timely fashion.
8. As a parking lot owner, I want to have a traffic log with timestamps so I can analyze the times of the day my lot is in higher demand.
9. As a prospective parker, I want to see parking lot rates while browsing

locations so I can make an informed decision on where to park.

10. As a prospective parker, I want to navigate to a parking lot I've reserved so I can conveniently get to my desired parking spot.
11. As a parking garage owner, I need to maintain spot inventory so that parking spots shown on the app match their availability in real life.
12. As a sales manager, I want to generate sales reports by date range so that the upper management can be kept up to date on the parking garage's performance & profitability.

Use Case: View Reservation

Identifier	UC-1 View Reservation
Goal	Allow the user to view their current reservation tickets
Requirements	The user must have a registered account; the system must validate user credentials securely; the system should provide appropriate error messages for failed login attempts
Initiating Actor	User
Participating Actor(s)	Authentication Service Provider, Database System
Pre-conditions	User must have a ticket present in the database associated with their id
Post-conditions	The user is able to successfully view their active and completed tickets for parking spots.
Included Use Case(s)	n/a
Extension(s)	No ticket: user hasn't ever made a parking reservation System Downtime: User is notified that login services are currently unavailable.

Table 1: Typical Course of Action

Seq#	Actor's Action	System's Response
1	User opens "My Spots" page	
2		System redirects user
3		System searches for all tickets associated with the user
4		System displays tickets to the user

Table 2: Alternate Course of Action

Seq#	Actor's Action	System's Response
1	User opens "My Spots" page	

2		System redirects user
3		System searches for all tickets associated with the user
4		System doesn't find any ticket from the user
5		Doesn't populate page with tickets

Table 3: Exceptional Course of Action

Seq#	Actor's Action	System's Response
1	User opens "My Spots" page	
2		System redirects user
3		System searches for all tickets associated with the user
4		System experiences a timeout during the searching process
5		System displays a timeout error to the user

Use Case: Reserve a Parking Spot

Identifier	UC-2 Reserve
Goal	Reserve a physical parking space
Requirements	Know the layout of the parking garage
Initiating Actor	User
Participating Actor(s)	Parking Lot Owners, Third party payment vendors
Pre-conditions	Locations for the app turned on, User is logged in, Another User not parked in the spot
Post-conditions	User has a physical parking space reserved in the app
Included Use Case(s)	UC-1 Login, Report, Payment method
Extension(s)	N/A

Table 1: Typical Course of Action

Seq#	Actor's Action	System's Response
1	User opens the app	
2		Display parking garages on a map
3	User selects a parking garage	
4		System displays the bottom level of the parking garage
5	User cycles through parking garage levels	
6	User selects a parking space	
7		System asks the user to input their payment information to pay
8	User submits their payment information	
9		System asks user if System can save their payment information for later
10		System confirms the reservation
11	User review their reservation in the reservation page	
12		System confirms that the transaction was a success
13		System displays the reservation of the User

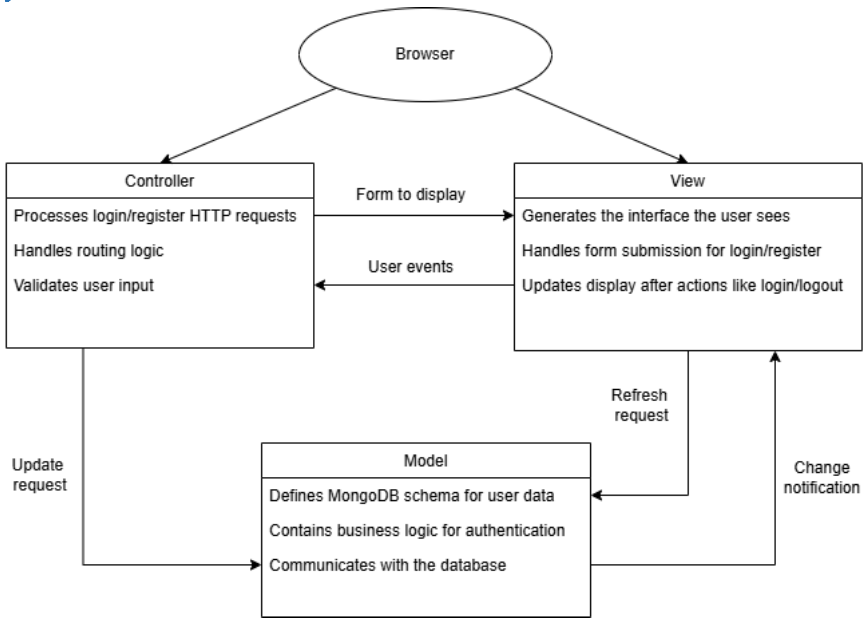
Table 2: Alternate Course of Action

Seq#	Actor's Action	System's Response
1	User opens app, reserves parking spot, and submits their payment information	
2		System checks information
3		System determines submitted payment information is unacceptable
4		System informs User that the payment information was incorrect
5	User submits correct payment information	
6		System checks the submitted payment information
7		System determines submitted information was correct
8		System proceeds to display reservation to User

Table 3: Exceptional Course of Action

Seq#	Actor's Action	System's Response
1	User opens the app	
2		System attempts to display garages to map
		System displays an error message

System Architecture


Browser
Controller
Processes login/register HTTP requests
Handles routing logic
Validates user input
Form to display
User events
View
Generates the interface the user sees
Handles form submission for login/register
Updates display after actions like login/logout
Refresh request
Change notification
Update request
Model
Defines MongoDB schema for user data
Contains business logic for authentication
Communicates with the database

User interface with the view using a browser application on their computer. The view components allow the user to interact with the controller to change the model (Database). In the case of our project, a user reserving a parking spot or searching for an address would require input from the view. These actions will then go to the controller, where the model can safely be queried and results sanitized for the user. The model is where all the data is stored. Both parking lot data, reservation data, and stripe payment data are stored in the model.

Components

Web Application

Languages/IDE: Javascript, Node.js, HTML/CSS, Visual Studio Code

Frameworks/Tools: Express.js, Vite, MongoDB, Postman, Stripe

Version Control: Github

Development: Split into Frontend and Backend teams

Frontend (React + Vite + Tailwind CSS): This handles all user interactions like login, browsing parking spots, and reservations

- Built with React using TypeScript
- Styled using Tailwind CSS
- Uses Vite for fast performance
- Fetches data from the backend using API calls

Backend (Node.js + Express + Stripe): This receives data from the frontend and performs logic like validating users or saving reservations, and securely processing payments using Stripe.

- Built with Node.js and Express.js
- Uses an MVC structure:
    - Routes for API endpoints
    - Controllers for logic
    - Models for MongoDB data handling
- Connects to MongoDb database using Mongoose

Database (MongoDB): Stores data like users, parking garages, spot availability, and reservations

Deployment

Make sure node version 18 is used
Required Install

```
npm install
npm run dev
```

Optional Install (nvm ensures node 18 is being used)

```
nvm use 18
npm install
npm run dev
```

Populate env file with
Backend:
DB_URI= mongodb url

JWT_SECRET= jwt_secret hash key

STRIPE_SECRET_KEY= stripe secret key

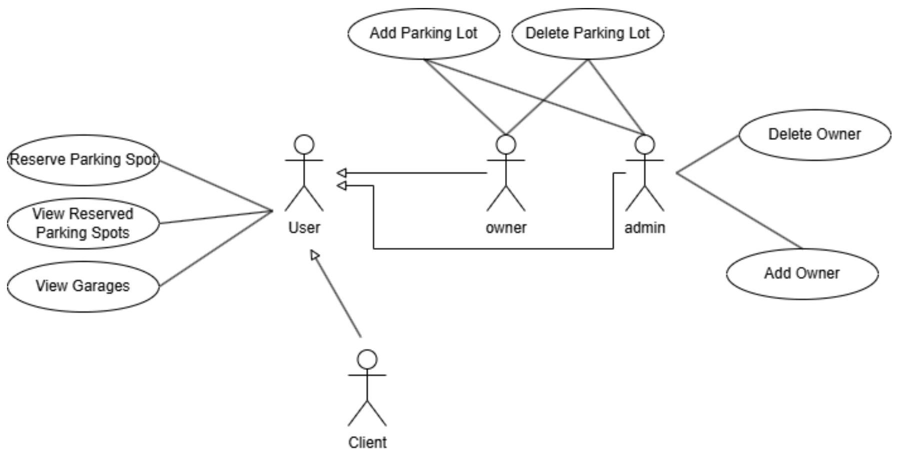
VITE_STRIPE_PUBLIC_KEY= strip public key

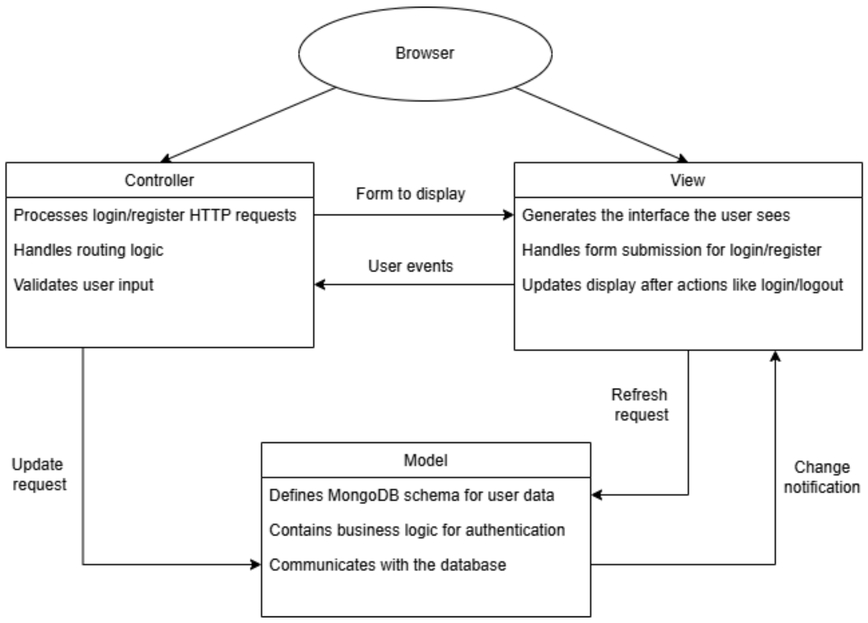
Frontend:
VITE_GOOGLE_MAPS_API_KEY= google maps api key

STRIPE_SECRET_KEY= stripe secret key

VITE_STRIPE_PUBLIC_KEY= strip public key

Diagrams




Browser
Controller
Processes login/register HTTP requests
Handles routing logic
Validates user input
Form to display
User events
View
Generates the interface the user sees
Handles form submission for login/register
Updates display after actions like login/logout
Refresh request
Change notification
Update request
Model
Defines MongoDB schema for user data
Contains business logic for authentication
Communicates with the database