ParkSmart

CSCD350 Project Report

Team 2 Boolean Bros

Submitted by:

Nicholas Burlakov	**nburlakov@ewu.edu**
Tyler Woody	**twoody1@ewu.edu**
Caleb Stewart	**cstewart15@ewu.edu**
Austin Harrison	**aharrison13@ewu.edu**
Dillon Pikulik	**dpikulik@ewu.edu**

Dr. Kaur Sanmeet

Computer Science and Electrical Engineering Department

Eastern Washington University

June 2024

ABSTRACT

ParkSmart is a web application designed to streamline the process of reserving parking spots. Developed using Flask, W3.CSS, HTML, Python3, and JavaScript, ParkSmart offers a seamless user experience for locating and reserving parking spaces in advance. With ParkSmart, users can easily find and secure parking spots, reducing the hassle and time associated with parking in busy areas.

https://www.youtube.com/watch?v=pk4n-XI_ZxU&ab_channel=ParkSmart

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS/GLOSSARY

API	Application Programming Interface

Team Members - Bios and Project Roles

Nicholas Burlakov is a senior at Eastern Washington University, graduating in Winter 2025 with a degree in Computer Science and a minor in Cybersecurity. He enjoys reading books and playing chess in his free time. His role in the creation of ParkSmart was Azure orchestration, w3.css alignments, proof of concept payment API, and timed parking database queries.

Tyler Woody is a senior at Eastern Washington University and is expected to graduate in Spring 2205 with a Bachelor of Science in Computer Science. Tyler is interested in game development, software development, and Cyber Security. In his free time, Tyler likes to play video games, go rock climbing, or just spend time outside. His responsibilities for the project were to make sure the parking lot was working, which included dynamically creating parking lots, having a status of reserved or not, and working with the database.

Caleb Stewart is a junior at Eastern Washington University who is expected to graduate in Spring 2025 with a Bachelor of Science with a major in Computer Science. Caleb's interests are VR, AR, and game development, and would like to make a career from any of those options. His responsibilities for this project included his backend implementation using Python Flask, user login and implementation, maintaining the database, and team coordination.

Austin Harrison is a senior at Eastern Washington University; he is expected to graduate Spring of 2025. The areas that interest him most are software development, big data, and networks. In his free time, he likes to be outside, whether that is kayaking, hiking, or just going on a walk with his dog. At the same time, he also enjoys just staying in and playing some video games, or going to the gym. In regards to this project, his roles ranged from activity diagrams, and database construction, to writing tests and helping figure out how to use Flask servers with the project.

Dillon Pikulik is a senior at Eastern Washington University working on obtaining a Bachelor of Computer Science and a minor in Cybersecurity. Currently working for a start-up and working to develop new software from the ground up. His roles for ParkSmart include database management and implementation, Azure orchestration, and front-end work.

1. Introduction

1.1 Project Overview

ParkSmart is an innovative parking management system designed to alleviate the challenges drivers and parking lot owners face in urban environments. The project focuses on providing real-time information on parking availability, payment options, and dynamic fee adjustments based on events and holidays. The system aims to streamline the parking process, reduce traffic congestion, and enhance the efficiency of urban mobility. ParkSmart, where parking meets precision, making every spot count.

1.2 Problem Definition and Scope

Problem Definition:

In urban areas, finding parking spaces can be a real challenge for drivers. The traditional parking search method typically relies on manual observation, which can result in wasted time and congested traffic, leading to inefficient urban mobility.

Additionally, parking lot owners may need help managing their parking facilities efficiently. Current methods of monitoring parking occupancy are often done by manually driving a vehicle around the parking lot at specific times of the day and using parking enforcement equipment to scan license plates. This method can be labor-intensive and prone to inaccuracies.

Scope:

This project targets users who consistently need to and struggle to find parking spots in an urban environment. Parking lot owners and venue owners could also utilize this application to ensure consistent and adequate parking for their customers.

1.3 Assumptions and Constraints

Assumptions:

- Parking lot owners will need to invest in initial setup costs for installing cameras above their parking lot and any equipment needed to scan parking spots
- Drivers will need to have access to a smartphone and internet to use ParkSmart

Constraints

- The cost barrier of the initial camera equipment
- Poor weather (rain or snow) may cause inaccurate results in the parking availability view
- Creating the parking availability view in software may take a lot of labor resources.

1.4 Objectives

By guiding drivers to available parking spots efficiently, ParkSmart will help reduce traffic congestion. With fewer cars circling searching for a parking spot, not only will the user be happy to find a spot easily and waste less gas, but every other driver will be happy that one less person is taking up road space.

Parking lot owners will also be able to save on labor costs by only deploying parking enforcement when they notice a discrepancy in how many customers paid and how many are parked. Additionally, this application will give parking lot owners insight into how busy a parking lot is. If the parking lot owners notice a parking lot isn't busy and/or losing profit, the owner may want to consider renting the lot to other companies or completely selling it.

1.5 Methodology Used

This project followed a waterfall and iterative development approach. We comprehensively planned and revised the requirement analysis and architectural design

before moving to the project's development. Once the planning was done, an iterative approach was used for implementation, testing, and deployment.

1.6 Project Outcomes and Deliverables

ParkSmart's primary deliverable is a desktop and mobile-friendly web application. The application serves as the central platform for users, providing information on parking availability, fees, and other details. Users will be able to access the application anywhere with an internet connection, allowing them to view and plan their parking ahead of time or find available spots on the go.

We also implemented an integrated database into ParkSmart, which holds user, owner, parking lots, parking spots, and reservation information. Utilizing all of this separate data ParkSmart efficiently and comprehensively provides a view of parking lot occupancy.

1.7 Novelty of Work

ParkSmart distinguishes itself through several innovative features for parking management problems.

- Scalable Architecture: This project can be easily scaled for multiple parking lots.
- Real-time parking availability viewing: Users can see their favorite parking lots in real-time and see which spots are available.
- User-Centric Design: ParkSmart was designed for the user. Made with an intuitive interface, the application is easy to understand and convenient to use.

2. Requirement Analysis

2.1 Overall Description

2.1.1 Product Perspective

ParkSmart is an application that allows users to view real-time parking lot availabilities, save user information, and allow parking lot owners/venue owners to manage their lots online. ParkSmart also offers payment options for users once they park their car. Users interact with ParkSmart through a web application.

2.1.2 User Stories and Use Case Diagrams

User 1: Driver

- An individual vehicle owner who travels and needs to park at various locations.

As a Driver

I need to know of available spots

So that I can save time parking

Details and Assumptions

- There will be an ability to see how many parking spots are available in real time.
- There will be a way to see payment fees for each parking lot.
- There will be a way to see how holidays/events/after-hours affect the fees.

Acceptance Criteria

Given there are 15 spots available,

When 10 are taken,

Then I will have 5 spots to choose from.

User 2: Parking Lot Owner

- A group or individual who manages or owns a parking facility.

As a Parking Lot Owner

I need an application that checks available parking spots

So that I can notice discrepancies between paid customers and actual parked cars

Details and Assumptions

- Parking lots may have a set amount of time a car could park.

- Parking lots may only be open certain hours of the day
- Payment rates change throughout the day

Acceptance Criteria

Given there are 20 parking spots

When 15 spots are taken, but only 12 spots are paid.

Then I know 3 cars have not paid, and parking enforcement can take action.

User 3: Venue Owner

- A group or individual who operates or owns a venue, such as an arena, mall, stadium, etc.

As a Venue Owner, who owns a private business where many customers come to park on my premises.

I need to be able to see that my customers have adequate parking, for a reasonable price. I would also need to ensure that my customers would be able to park during holidays/events and filter them to nearby open lots if the on-site parking is full.

So that I have more time to focus on my business and customer satisfaction. Also, if a customer arrives when the lot is full they will not be discouraged from visiting my venue due to difficult parking.

Details and Assumptions

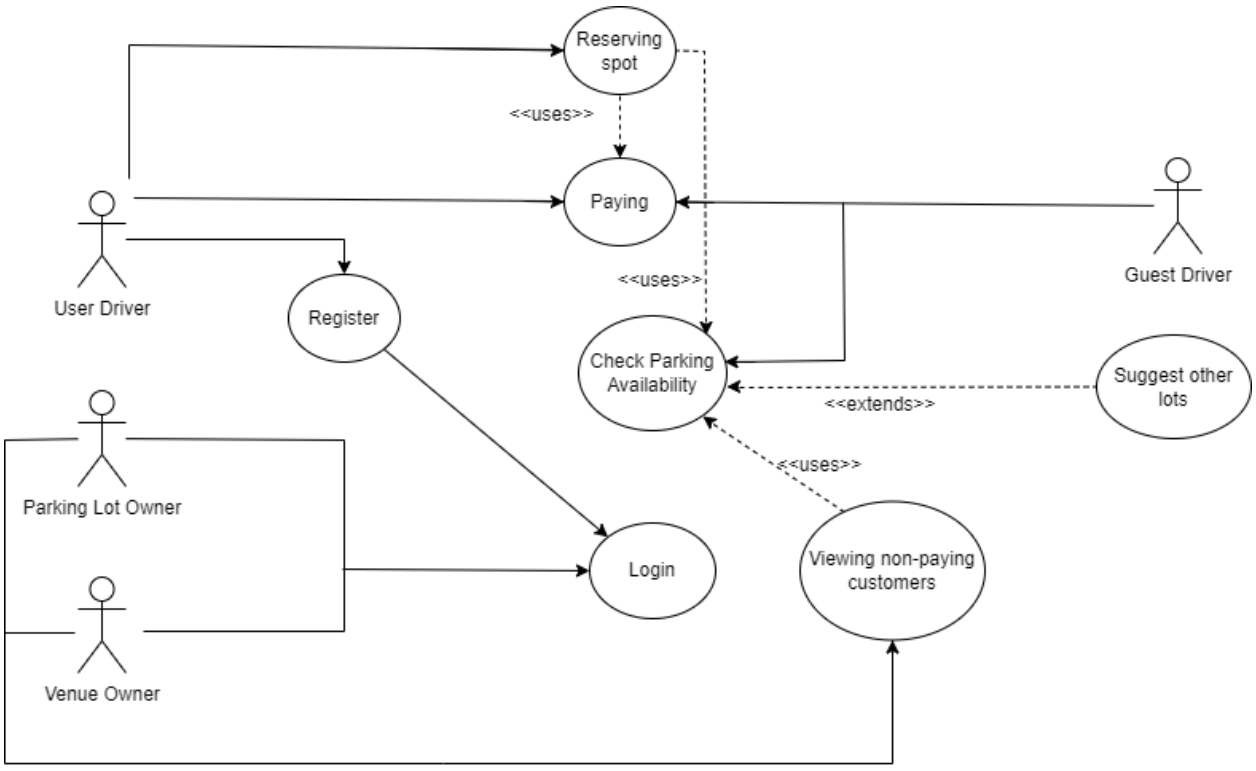
- I'm paying the parking lot owner for clients to have quick access to my venue.
- I'm paying the parking lot owner a base price.
- I will get notified when the parking lot is full.

Acceptance Criteria

Given there are 100 parking spots available,

When 70 spots are taken,

Then there will be at least 70 people at my venue.



2.1.3 Product Features

The features of ParkSmart will include a parking reservation system, real-time parking availability view, payment options, and owner viewing/analysis options.

2.2 External Interface Requirements

2.2.1 User Interfaces

The ParkSmart application will be available on devices that have access to the internet. Such devices include computers, smartphones, and tablets. No matter the users' choice of device as long as they have internet the user will be able to interact with all features of ParkSmart.

2.2.2 Hardware Interfaces

ParkSmart will interface with hardware components such as cameras to capture the view of a parking lot. Depending on the development of ParkSmart as a whole, this may also include installing QR codes along with QR code scanners in the camera to

view the open availabilities of a parking lot. The cameras will capture the data of the parking lot and relay the information back to the central system.

2.2.3 Software Interfaces

Depending on the development of ParkSmart as a whole, external software components such as APIs may be utilized to provide the user with additional parking availabilities. An API that could be utilized for such a need would be ParkMobiles. ParkMobiles API provides details of parking availability at street meters, so by making use of ParkMobiles API, ParkSmart would be able to provide the user with parking availability information from parking lots and street parking.

2.3 Other Non-Functional Requirements

2.3.1 Performance Requirements

ParkSmart will be able to handle a large amount of users viewing and reserving parking spots with minimal to no delays in the system. This will ensure a seamless experience for users no matter the time of day or how many customers are using the application.

2.3.2 Safety Requirements

ParkSmart prioritizes the safety of its users. ParkSmart will prompt and remind the user to not use their mobile device and drive at the same time.

2.3.3 Security Requirements

To protect users' private information, ParkSmart will implement security features such as authentication mechanisms and encryption of sensitive information.

3. Solution Approach

3.1 Investigative Techniques

To develop the idea of ParkSmart the Boolean Bros team came together and discussed their shared hatred; parking, mainly at the Catalyst building. Once our team came to this realization we realized we could do something about it. After a brainstorming session, we formulated the base idea of ParkSmart and proposed our

idea to the class in our project proposal presentation. This presentation acted as a kind of user survey, where we found the class was ecstatic and loved the idea of ParkSmart. This is when we knew ParkSmart could change urban parking for good.

3.2 Proposed Solution

To address the problems with traditional parking management, we proposed the ParkSmart application to have these key features:

- Real-time parking availability view: This will give users real-time information on how busy various parking lots would be.
- Parking reservation system: This gives users a way to reserve spots in advance. This idea is similar to reserving seats in a movie theater.
- All-in-one system for payment integration: Instead of having to go to an exterior application for payment, with ParkSmart the users can pay directly through the application once they reserve their spot.
- User accounts and log-in: Having the user be able to create an account prevents the user from needing to input tedious vehicle information every time they want to reserve a spot. Accounts also create a personal touch between ParkSmart and the user

3.3 Tools and Technologies Used

To develop ParkSmart various technologies were used to ensure a robust and scalable system.

Back-end:

- Python was used for the core logic of the backend. This is the language we manipulated the database with.
- Flask: A framework for Python. Allowed us to handle routing and user authentication within ParkSmart

Front-end:

- HTML: The basic structure of the webpage.
- W3 CSS: A prebuilt CSS framework that is modern, responsive, and designed for mobile applications.
- JavaScript: Used to add interactivity to the availabilities and reservation pages.

Database:

- SQLite: This was the choice of database because it is lightweight and built directly within Python.

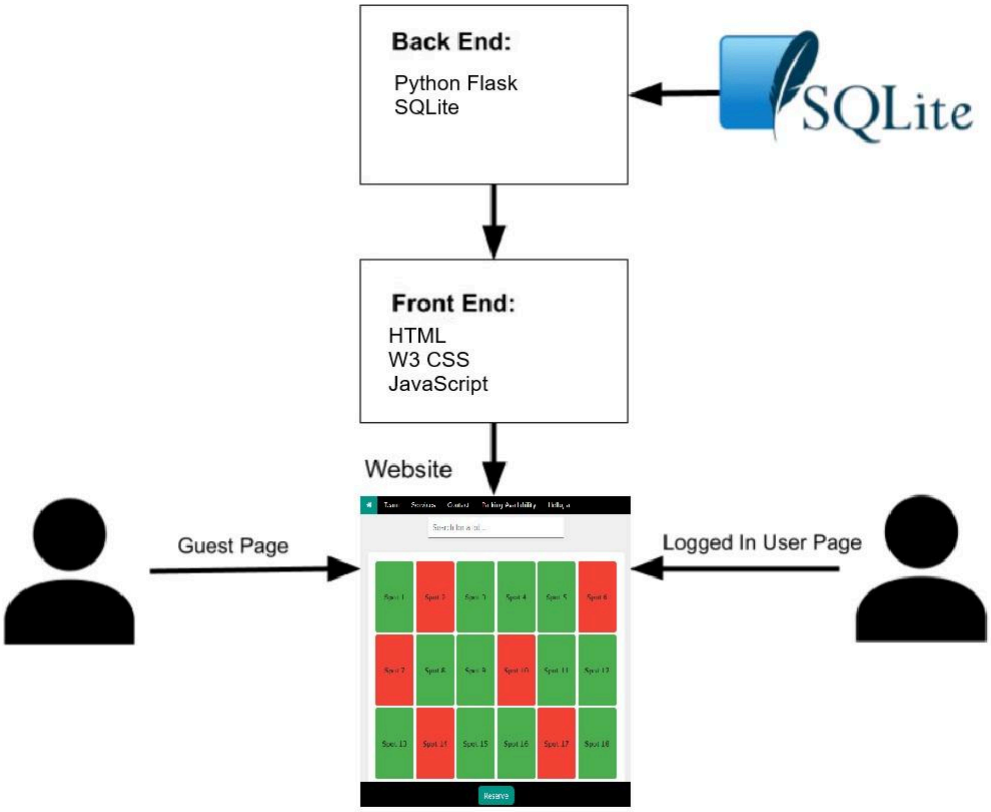
Version Control and Development Tools:

- GitHub: Used for project collaboration, planning, and issue tracking. This is where the ParkSmart code is hosted, and can be accessed for future development.

4. DESIGN SPECIFICATIONS

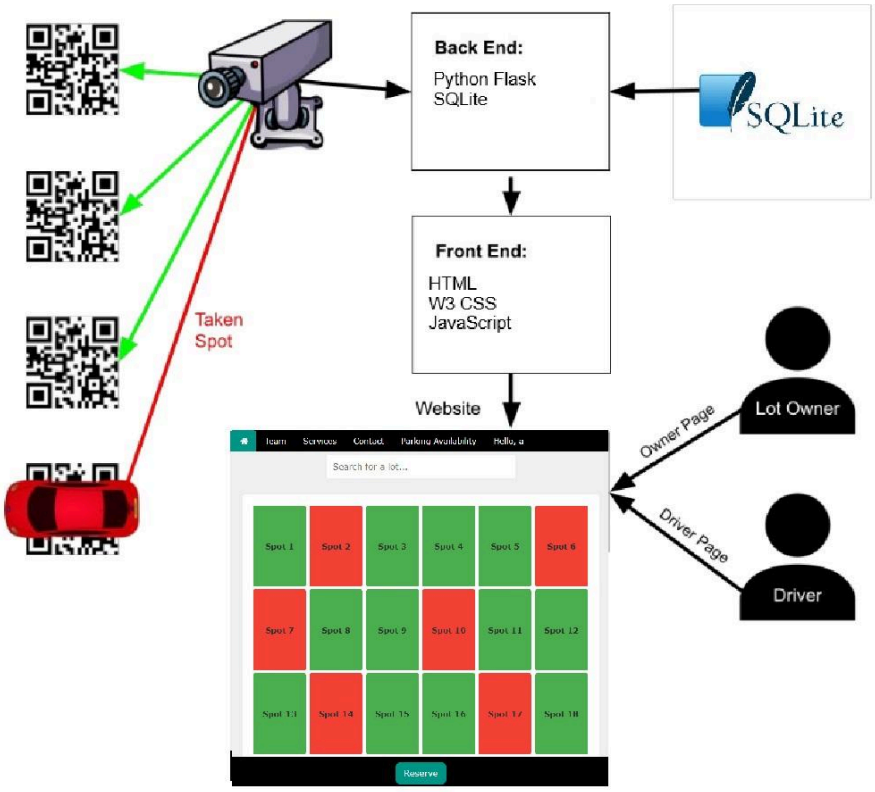
4.1 System Architecture (Eg. Block Diagram / Component Diagram)

Current System Architecture:



This is the system architecture diagram of where our project currently stands. The application starts in the backend with the database. In our database, we store data such as parking lot, parking spot, user, owner, vehicle, and reservation information. All of this data can be accessed through Python, our choice of backend language. Also in

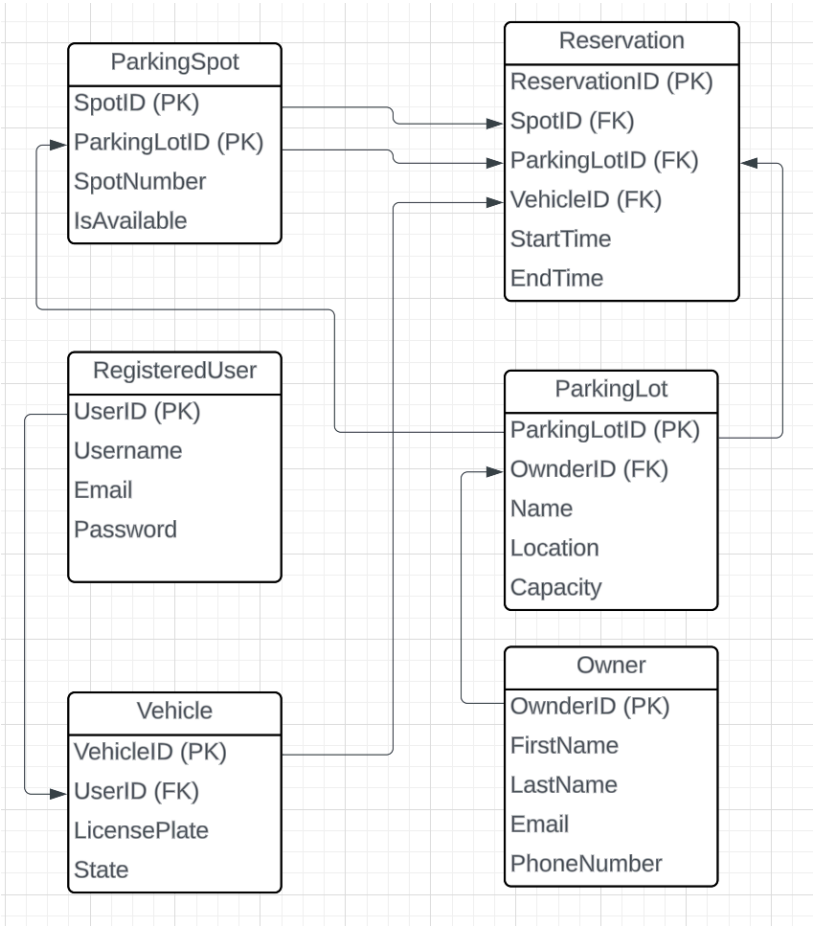
Python, we are using Flask to create our web application. For our front end, we chose to work with HTML, W3 CSS, and JavaScript to create a seamless and smooth experience for our users. In our current application, we only accommodate the driver as a user. The driver can either have an account or use the application without signing up. The benefit of having an account is that once users go to reserve a spot the logged-in user does not need to input their vehicle information, as it is already linked to their account.



This is the system architecture that we strive for ParkSmart to have. The backend and front-end languages stay the same, but eventually would like to add a camera scanning module to our application that can scan and detect parking to show the user available parking spots. In our future system architecture diagram we will also accommodate a user with owner privileges, which the application will provide a much more detailed view of each parking lot and spot.

4.2 Design Level Diagrams

Database Schema:



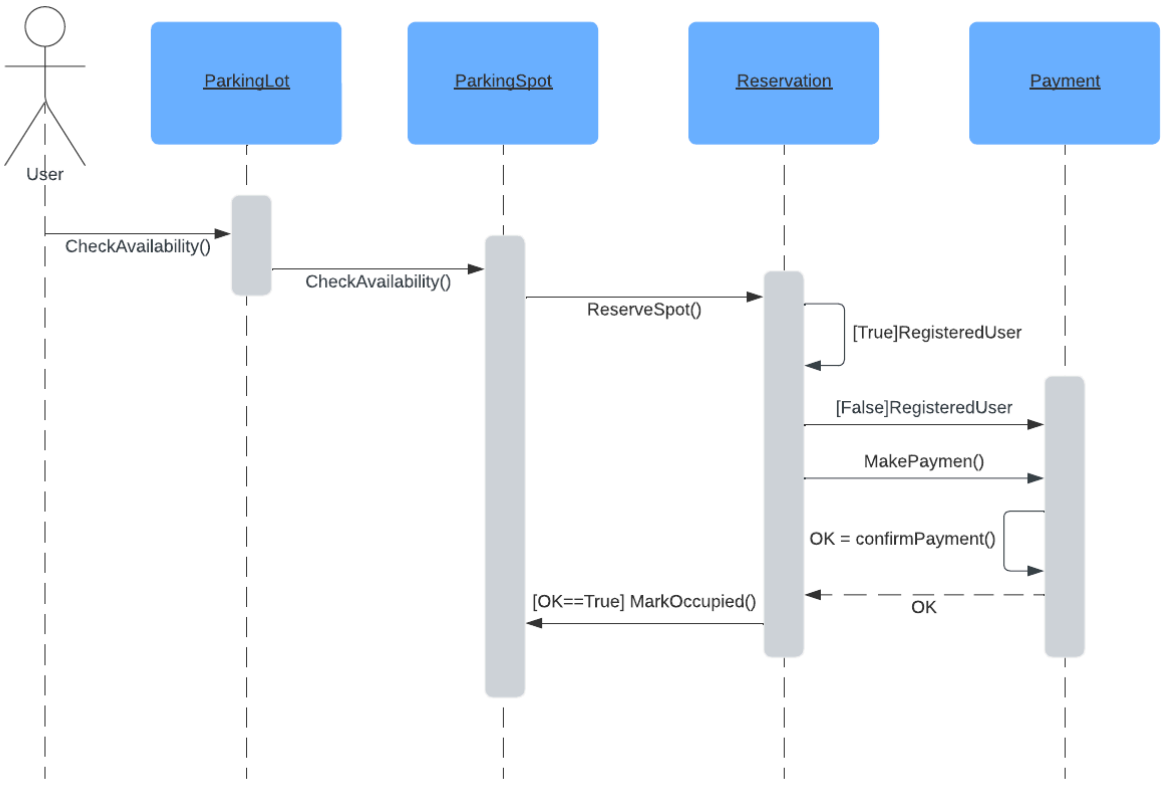
This is a diagram of our database schema. At this current stage of the project, we have only been able to utilize the Reservation, RegisteredUser, and Vehicle table, but we have implemented the rest of the table for future use.

- The RegisteredUser table will store the username, email and password of the user.
- The Vehicle table will store the UserID of the vehicle owner, license plate, vehicle type, and vehicle color.
- The Owner table will store the first and last name of the owner, as well as their email and phone number.
- In the ParkingLot table, the OwnerID is a foreign key, meaning a parking lot can have an owner. The ParkingLot table also has a parking lot name, location, and spot capacity.

- The ParkingSpot table has a primary key from the ParkingLot, as well as a spotID. This allows us to create unique spots for each parking lot. The ParkingSpot table also holds if the spot is available or not.
- In the Reservations table, all of the elements combine together. It holds a SpotID, ParkingLotID, and VehicleID. This table also contains the start and end times of when the reservation begins and ends.

4.3 User Interface Diagrams

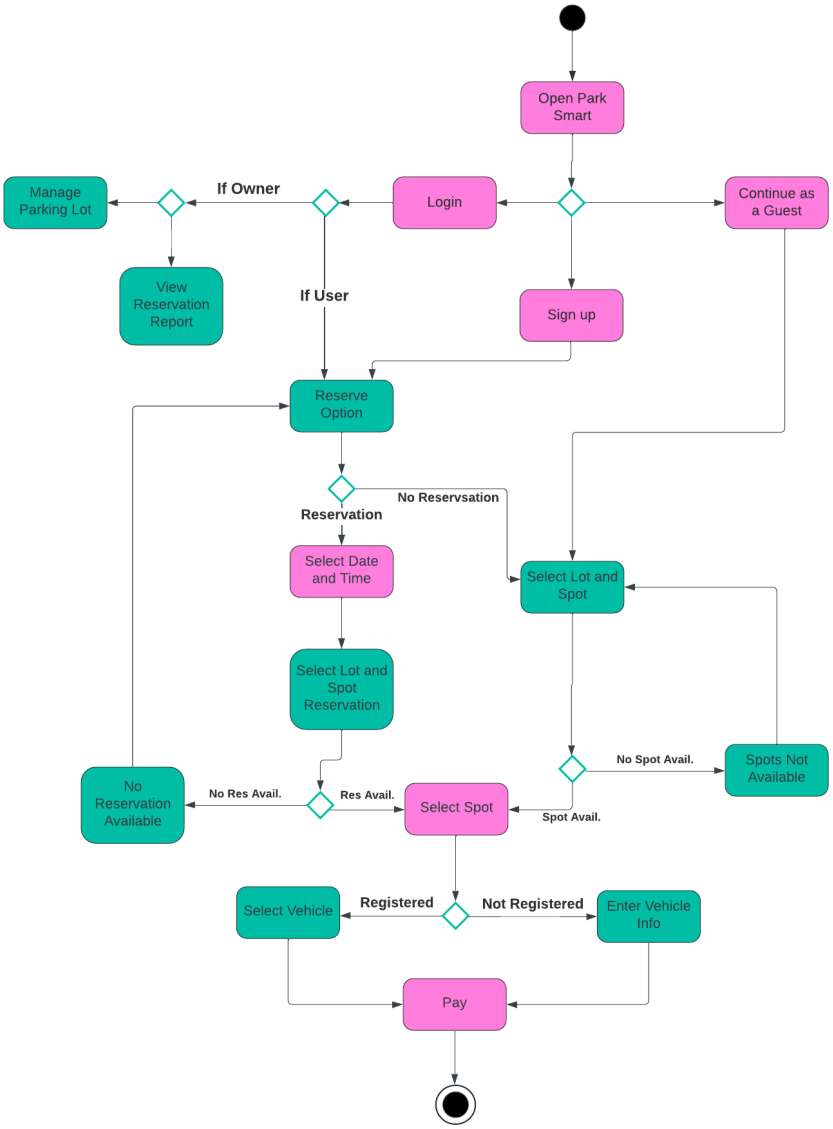
Sequence Diagram:



This sequence diagram shows the flow at which a user will use the ParkSmart application. First, a user will check the availability of a ParkingLot and ParkingSpot. Once the user finds an unoccupied spot they would like to reserve our system, our system checks if the user is logged in. If the user is logged in then they will only need to input their payment information, but if the user is not logged in then they will need to input their payment information and vehicle information. Once the payment is done and

checked then our system will go back and mark the reserved spot occupied, so future users can see that someone is currently parked in that spot.

Activity Diagram:



This is our activity diagram. This diagram shows what a user would go through when opening ParkSmart.

5. Implementation and Experimental Results

5.1 Prototype Description

The parking management system prototype provides a user-friendly interface that allows registered users and guests to select and reserve parking spots. Built using HTML, W3 CSS, and JavaScript for the front end, Python for backend processing, and a local database for data storage, the prototype includes functionalities such as account creation, vehicle information storage, and time-based parking reservations.

5.2 Working of the project

Upon accessing the system, users can either log in to their accounts or continue as guests. Registered users have their vehicle information (type, color, and license plate number) saved for quicker reservations in future sessions. Guests need to enter their vehicle information during each visit. Users select available parking spots displayed on the lot, input their parking duration, and proceed to payment. The system updates the parking lot status in real time to reflect the new reservation.

5.3 Procedural Workflow

1. **User Authentication**: Users log in or proceed as guests. Authentication checks are performed via Python scripts interfacing with the local database.
2. **Spot Selection**: Users are presented with an interactive parking map where they can choose from available spots. Spot statuses are dynamically updated using JavaScript.
3. **Vehicle Information Input**: Registered users have their vehicle details pre-filled, while guests input theirs manually. This data is captured or retrieved using HTML forms.
4. **Duration and Payment**: Users specify how long they intend to park and are directed to the payment interface where the cost is calculated.

5. **Reservation Confirmation**: Once payment is processed, the system confirms the reservation, updates the spot status to 'reserved', and the database records the transaction.

5.4 Algorithmic Approaches Used

JavaScript is used to dynamically render the parking spots based on their current availability, which is fetched from the database. On selection, the algorithm checks for conflicts and updates the database in real time.

5.5 System/Code Screenshots

```
import ...

app = Flask(__name__)
app.secret_key = "Boolean_Bros_Rule"

#import images
IMAGES_FOLDER = os.path.join('static', 'images')
app.config['UPLOAD_FOLDER'] = IMAGES_FOLDER

# Initialize SQLite database
 Caleb Stewart
def init_db():
    conn = sqlite3.connect('parking.db')
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS Reservations (
                      ReservationID INTEGER PRIMARY KEY AUTOINCREMENT,
                      SpotID INTEGER NOT NULL,
                      ParkingLotID INTEGER,
                      VehicleID INTEGER,
                      StartTime DATETIME,
                      EndTime DATETIME,
                      FOREIGN KEY (SpotID) REFERENCES ParkingSpot(SpotID),
                      FOREIGN KEY (ParkingLotID) REFERENCES ParkingLot(ParkingLotID),
                      FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID));
                   ''')
```

This snippet of code shows the start of the initialization of the database.

```python
@app.route( rule: '/reserve', methods=['POST'])
def reserve():
    data = request.json
    spot_id = data.get('spot_id')
    parking_lot_id = data.get('parking_lot_id', 1)  # Assuming a default parking lot ID
    start_time = data.get('start_time', '2024-01-01 00:00:00')  # Default start time
    end_time = data.get('end_time', '2024-12-31 00:00:00')  # Default end time
    licenseNum = data.get('license_number', '123456')
    vehicleType = data.get('vehicle_type', 'Car')
    vehicleColor = data.get('car_color', 'Black')

    conn = sqlite3.connect('parking.db')
    cursor = conn.cursor()

    if('username' in session):
        cursor.execute( _sql: "SELECT UserID FROM RegisteredUser WHERE Username = ?",  _parameters: (session['username'],))
        userID = cursor.fetchone()[0]
        cursor.execute( _sql: "SELECT VehicleID FROM Vehicle WHERE UserID = ?",  _parameters: (userID,))
        vehicle_id = cursor.fetchone()[0]
    else:
        cursor.execute( _sql: "INSERT OR IGNORE INTO Vehicle (UserID, LicensePlate, VehicleType, VehicleColor) VALUES (?, ?, ?, ?)",
                        _parameters: (None, licenseNum, vehicleType, vehicleColor))
        cursor.execute( _sql: "SELECT VehicleID FROM Vehicle WHERE LicensePlate = ?",  _parameters: (licenseNum,))
        vehicle_id = cursor.fetchone()[0]

    cursor.execute( _sql: '''INSERT INTO reservations (SpotID, ParkingLotID, VehicleID, StartTime, EndTime)
                      VALUES (?, ?, ?, ?, ?)''',
                    _parameters: (spot_id, parking_lot_id, vehicle_id, start_time, end_time))
    conn.commit()
    conn.close()

    return jsonify({"message": "Reservation successful"}), 200
```

This code snippet shows what the program does when reserving a parking spot.

```javascript
// Generate parking spots
fetch( input: '/reserved_spots') // Fetch reserved spots
    .then(response : Response  => response.json())
    .then(reservedSpots => {
        for (let i : number  = 1; i <= 50; i++) {
            const spot : HTMLDivElement  = document.createElement( tagName: 'div');
            spot.classList.add('spot');
            spot.textContent = `Spot ${i}`;
            spot.dataset.spotId = i;

            if (isReserved(i, reservedSpots)) { // Check if spot is reserved
                spot.classList.add('reserved');
            } else {
                spot.classList.add('available');
                spot.addEventListener( type: 'click',  listener: function() : void  {
                    if (spot.classList.contains('reserved')) return;
                    if (selectedSpot) {
                        selectedSpot.classList.remove( tokens: 'selected');
                    }
                    selectedSpot = spot;
                    spot.classList.add('selected');
                });
            }

            parkingGrid.appendChild(spot);
        }
    });
```

This snippet of code generates the parking spots on the availabilities page for the user to see.

5.6 Testing Process

We made sure to do a thorough job when testing the ParkSmart application. We focused on ensuring the functionality and usability of the application across many modules. These included the home page, parking availability, checkout, registration, and the database interface. We did automated testing using the PyTest library to simulate user interactions and validate both the front-end and back-end components of the application.

5.7 Test Plan

The test plan was structured so that we covered all of the most important features of the ParkSmart application. It included the testing of the loading of pages, the presence and functionality of navigation links, dynamic content rendering, form validation, and ensuring that the databases were being set up correctly. The tests prioritized making sure that the typical user interaction was functioning correctly.

5.8 Features To Be Tested

Key features tested included:

- Home Page: Navigation, main sections, dynamic content, and JavaScript functionality.
- Parking Availability: Page loading, search functionality, and reservation process.
- Checkout Process: Form elements presence and submission logic.
- Registration Process: Form validations and submission.
- Database Interface: Correct display of data tables and content.

5.9 Test Techniques

We used a combination of black-box techniques to ensure functionality, and white-box testing techniques for internal operations. This included tests for input fields, form validation, and visual inspection for UI elements.

5.10 Test Cases

The test cases were designed to verify each element's expected behavior under normal conditions as well as boundary conditions. To give a couple of examples, some tests checked to make sure the navigation links were present and directed to the correct pages, forms were present, and dynamic content like images were properly rendered.

5.11 Test Results

The test results were overwhelmingly positive, with all of the major functionalities working as expected. Minor issues were identified in dynamic content reading and form error handling, which were then addressed and handled. One thing to note is that I could not figure out how to test the pages where it required the user to be logged in. This will require more research on our part to figure out how to handle this. Overall, it showed that our application was ready to be presented and in good shape.

5.12 Validation of Requirements

The testing process effectively validated the functional and non-functional requirements of the ParkSmart application. Requirements related to system performance, user accessibility, and real-time responsiveness were all passing and looking good. This validation was crucial in moving the project from development to presenting it to the class.

6. Conclusions and Future Directions

6.1 Conclusions

Creating and launching ParkSmart has shown how technology can tackle the common needs of the people. especially with parking. The app combines modern web technologies like Flask, W3.CSS, HTML, Python3, and JavaScript to make reserving parking spots easy and efficient. By cutting down the time spent looking for parking, ParkSmart makes life easier for users and helps improve urban traffic flow. This project highlights how important it is to have a strong backend and a user-friendly interface to solve real-life issues effectively.

6.2 Environmental, Economic, and Societal Benefits

ParkSmart offers multiple benefits across environmental, economic, and societal dimensions. Environmentally, by reducing the time vehicles spend searching for parking, ParkSmart lowers fuel consumption and decreases greenhouse gas emissions, contributing to improved air quality and reduced urban pollution. Economically, the application helps optimize the use of available parking spaces, leading to increased efficiency and potential revenue growth for parking providers. It also saves users time and fuel costs, enhancing overall economic productivity. Societally, ParkSmart alleviates the stress and frustration associated with finding parking, improving the quality of life for urban residents. It also contributes to reducing traffic congestion, making city streets safer and more navigable.

6.3 Reflections

Reflecting on the development of ParkSmart, several key insights emerge. Initially, we planned to use an Azure database, but due to unforeseen technical challenges, this proved unfeasible. As a result, we pivoted to using a local database with Flask, which, although not initially planned, ultimately offered a more straightforward and reliable solution. This experience underscored the importance of adaptability and the ability to pivot when necessary. Being flexible allowed us to overcome obstacles and find effective alternatives, ensuring the project's success.

Additionally, team communication was crucial throughout the development process. Regular meetings, clear communication channels, and collaborative problem-solving allowed the team to address issues promptly and efficiently. This strong teamwork fostered a productive development environment and ensured that everyone was on the same page with the project's goals and progress.

One key takeaway for future projects is the importance of focusing on back-end development before tackling the front end. This approach ensures that the core functionalities and data management systems are solid, providing a strong foundation

for the user interface to build upon. Developing in this order can lead to a more cohesive and efficient process, ultimately resulting in a more polished final product.

6.4 Future Work

Future enhancements for ParkSmart include implementing the Stripe Payment API to facilitate secure and efficient payment processing. Establishing a fully functional Flask database will enhance data management and scalability. Adding QR scanner functionality and vehicle recognition systems will ensure that only authorized vehicles are parked and improve overall security. Displaying vehicle visualization for parked cars and showing the time remaining for reserved parking spots will provide users with a better experience. Additionally, developing owner profiles will allow for personalized features and improved user management. These enhancements aim to make ParkSmart a more comprehensive and user-friendly solution for smart urban parking.

References

Nicholas Burlakov, Austin Harrison, Dillon Pikulik, Caleb Stewart, and Tyler Woody. "Project Proposal for CSCD 350 Spring 2024 - ParkSmart." Project Proposal - ParkSmart, 2024. Accessed June 11, 2024. https://github.com/Sanmeet-EWU/github-teams-project-bid-boolean-bros/blob/main/Resources/Group%202-Boolean%20Bros-Project%20Proposal.pdf.

Nicholas Burlakov, Austin Harrison, Dillon Pikulik, Caleb Stewart, and Tyler Woody. User Stories for CSCD 350 Spring 2024 - ParkSmart." User Stories - ParkSmart, 2024. Accessed June 11, 2024. https://github.com/Sanmeet-EWU/github-teams-project-bid-boolean-bros/blob/main/Resources/Team2-Boolean%20Bros-User%20Stories.pdf.

Nicholas Burlakov, Austin Harrison, Dillon Pikulik, Caleb Stewart, and Tyler Woody. Structural Modeling for CSCD 350 Spring 2024 - ParkSmart." Structural Modeling - ParkSmart, 2024. Accessed June 11, 2024. https://github.com/Sanmeet-EWU/github-teams-project-bid-boolean-bros/blob/main/Resources/Team2-Boolean%20Bros-Structural%20Modeling.pdf.

Nicholas Burlakov, Austin Harrison, Dillon Pikulik, Caleb Stewart, and Tyler Woody. Behavioral and Flow-Based Modeling for CSCD 350 Spring 2024 - ParkSmart." Behavioral and Flow-Based Modeling - ParkSmart, 2024. Accessed June 11, 2024. https://github.com/Sanmeet-EWU/github-teams-project-bid-boolean-bros/blob/main/Resources/Team2-Boolean%20Bros-Behavioral%20and%20Flow%20Modeling.pdf.

Nicholas Burlakov, Austin Harrison, Dillon Pikulik, Caleb Stewart, and Tyler Woody. Presentation 2 for CSCD 350 Spring 2024 - ParkSmart." Presentation 2- ParkSmart, 2024. Accessed June 11, 2024. https://github.com/Sanmeet-EWU/github-teams-project-bid-boolean-bros/blob/ma

in/Resources/Team%202%20-%20Boolean%20Bros%20-%20Presentation%202.pdf.

Nicholas Burlakov, Austin Harrison, Dillon Pikulik, Caleb Stewart, and Tyler Woody. Final Presentation for CSCD 350 Spring 2024 - ParkSmart." Final Presentation - ParkSmart, 2024. Accessed June 11, 2024. https://github.com/Sanmeet-EWU/github-teams-project-bid-boolean-bros/blob/main/Resources/Team%202%20-%20Boolean%20Bros%20-%20Final%20Presentation.pdf.