**Software Design**

**Document**

**for**

**<CSULA No Ticket 4 Me>**

**Version <1.1> approved**

**Prepared by <Brian Andrade, John Lopez, Gabriel Galindo, Braedon Edison, Haonan Ma, Anthony Sanchez, Angel Trujillo, Chiemela Eziechile>**

**<California State University, LA>**

**<11/22/24>**

Table of Contents................................................................................................................... <pg #>

Revision History..................................................................................................................... <pg #>

1. Introduction................................................................................................................ <pg #>
   1. Purpose........................................................................................................... <pg #>
   2. Document Conventions……………………................................................. <pg #>
   3. Intended Audience and Reading Suggestions................................................ <pg #>
   4. System Overview........................................................................................... <pg #>
2. Design Considerations............................................................................................... <pg #>
   1. Assumptions and dependencies..................................................................... <pg #>
   2. General Constraints........................................................................................<pg #>
   3. Goals and Guidelines.....................................................................................<pg #>
   4. Development Methods................................................................................... <pg #>
3. Architectural Strategies.............................................................................................. <pg #>
4. System Architecture................................................................................................... <pg #>
   1. ....................................................................................................................... <pg #>
   2. ....................................................................................................................... <pg #>
5. Policies and Tactics.................................................................................................... <pg #>
   1. Specific Products Used.................................................................................. <pg #>
   2. Requirements traceability............................................................................... <pg #>
   3. Testing the software....................................................................................... <pg #>
   4. Engineering trade-offs................................................................................... <pg #>
   5. Guidelines and conventions........................................................................... <pg #>
   6. Protocols........................................................................................................ <pg #>
   7. Maintaining the software............................................................................... <pg #>
   8. Interfaces........................................................................................................ <pg #>
   9. System's deliverables..................................................................................... <pg #>
   10. Abstraction..................................................................................................... <pg #>
6. Detailed System Design............................................................................................. <pg #>

6.x Name of Module........................................................................................... <pg #>

6.x.1 Responsibilities...................................................................................... <pg #>

6.x.2 Constraints............................................................................................. <pg #>

6.x.3 Composition........................................................................................... <pg #>

6.x.4 Uses/Interactions.................................................................................... <pg #>

6.x.5 Resources............................................................................................... <pg #>

6.x.6 Interface/Exports.................................................................................... <pg #>

1. Detailed Lower level Component Design

7.x Name of Class or File..................................................................................... <pg #>

7.x.1 Classification.......................................................................................... <pg #>

7.x.2 Processing Narrative(PSPEC)................................................................ <pg #>

7.x.3 Interface Description.............................................................................. <pg #>

7.x.4 Processing Detail.................................................................................... <pg #>

7.x.4.1 Design Class Hierarchy................................................................... <pg #>

7.x.4.2 Restrictions/Limitations.................................................................. <pg #>

7.x.4.3 Performance Issues........................................................................ <pg #>

7.x.4.4 Design Constraints......................................................................... <pg #>

7.x.4.5 Processing Detail For Each Operation........................................... <pg #>

1. User Interface
   1. Overview of User Interface.......................................................................... <pg #>
   2. Screen Frameworks or Images..................................................................... <pg #>
   3. User Interface Flow Model........................................................................... <pg #>
2. Database Design
3. Requirements Validation and Verification................................................................. <pg #>
4. Glossary..................................................................................................................... <pg #>
5. References................................................................................................................. <pg #>

**Revision History**

| Name | Date | Reason For Changes | Version |
| --- | --- | --- | --- |
| Brian | 10/10/24 | Adding 1.0 - 1.4 | 1.0 |
| Gabriel | 11/22/24 | Updating to meet new criteria. | 1.1 |
|  |  |  |  |
|  |  |  |  |

<Add rows as necessary when the document is revised. This document should be consistently updated and maintained throughout your project. If ANY requirements are changed, added, removed, etc., immediately revise your document.>

**1. Introduction**

**1.1 Purpose**

This Software Requirements Specification (SRS) defines the software requirements for the **"CSULA No Ticket 4 Me" Parking Registration System**, specifically for **Release 1.1**. The purpose of this document is to describe **what** the system must do in order to meet the needs of its users and stakeholders.

This SRS outlines the complete scope of the **"CSULA No Ticket 4 Me"** system, covering all aspects of its functionality, including user registration, permit management, notifications, and violation tracking. It details the system’s capabilities, but does not describe how the system will be implemented technically; that will be covered in the Software Design Document (SDD).

This document fully describes the **entire system** and does not focus on a specific subsystem or partial release. All major features, user interactions, and system behavior for Release 1.1 are included in this SRS, providing a comprehensive view of the system’s intended functionality.

**1.2 Document Conventions**

This Software Requirements Specification (SRS) follows specific conventions and standards to ensure clarity, consistency, and ease of interpretation. These conventions are applied throughout the document and are outlined below:

### **Typographical Conventions:**

* **Bold Text**: Used to emphasize section titles, key terms, and important concepts.
* *Italic Text*: Used to highlight examples, notes, or supplementary information.
* **ALL CAPS**: Used to denote constants or system-specific terms (e.g., "ADMIN", "USER").
* **[R#]**: Used to indicate unique requirement IDs (e.g., **[R1]**) for easy referencing and tracking of requirements.

### **Requirement Prioritization:**

* **Priority Levels**: Each requirement has its own priority level, indicated by the following tags:
  + **[HIGH]**: Essential features or functionalities that must be included in the release for the system to operate properly.
  + **[MEDIUM]**: Important features that add value but may be deferred if necessary.
  + **[LOW]**: Optional features that can enhance the system but are not critical to its core functionality.

### **Standards:**

* **IEEE 830-1998 Standard**: This SRS document is structured in accordance with the IEEE Standard for Software Requirements Specifications (IEEE 830-1998).
* **Numbering**: Requirements are numbered sequentially using a hierarchical numbering scheme (e.g., 1.1, 1.2, etc.) to provide clear organization and traceability.

### **Inheritance of Priority:**

* Unless otherwise stated, higher-level requirements’ priorities are **not** automatically inherited by detailed requirements. Each detailed requirement is assigned its own priority level, ensuring clarity on the importance of individual features.

These conventions aim to make the document clear, easy to navigate, and consistent, while ensuring that requirements are properly organized and prioritized for all readers.

**1.3 Intended Audience and Reading Suggestions**

This Software Requirements Specification (SRS) document is intended for multiple types of readers involved in the development, management, and use of the **"No Ticket 4 Me"** Parking Registration System. Each type of reader will find different sections of this document more relevant to their role. Below is a breakdown of the intended audience, along with recommendations on how to navigate the document.

### **Intended Audience**

* **Developers**: Developers are responsible for implementing the system based on the requirements outlined in this SRS. They will need to understand the functional and non-functional requirements, data models, and system constraints.
* **Project Managers**: Project managers will oversee the development process, ensuring that the project stays on schedule and meets all defined requirements. They will be most interested in the overall scope, goals, and priorities of the system to manage resources and timelines effectively.
* **Marketing Staff**: Marketing professionals will use this document to gain a high-level understanding of the system’s purpose, features, and target users. This information will help them create messaging that aligns with the system's capabilities.
* **End-Users**: End-users may review this document to understand the system’s functionality, particularly regarding how they will interact with it through user interfaces and what benefits they can expect.
* **Testers**: Testers will use this document to ensure that all system functionalities are thoroughly tested against the detailed requirements. They will focus on both functional and non-functional requirements to design comprehensive test cases.
* **Documentation Writers**: Writers responsible for creating user manuals and support materials will use this document to understand the system’s features and workflows. They will focus on the user interactions and user interface details.

### **Document Organization**

This SRS is organized into the following sections:

* **Introduction**: Provides an overview of the system, its purpose, scope, and key goals.
* **System Features**: Describes the high-level features of the system.
* **Functional Requirements**: Details what the system must do, focusing on user actions, system processes, and data management.
* **Non-functional Requirements**: Defines system qualities, such as security, performance, and scalability.
* **Use Cases and User Interfaces**: Describes how users will interact with the system, including specific use case scenarios and interface designs.
* **System Constraints**: Details any limitations or restrictions imposed on the system design and implementation.
* **Appendices**: Includes references and definitions of key terms, acronyms, and any additional documentation.

### **Suggested Reading Sequence**

* **For Developers**: Start with the **Introduction** for context, then focus on **System Features**, **Functional Requirements**, **Data Management**, and **System Constraints** to understand what needs to be built. Non-functional requirements may also be of interest for performance and security considerations.
* **For Project Managers**: Begin with the **Introduction** and **System Features** sections to understand the overall goals and scope. Follow this with the **Functional Requirements** and **Non-functional Requirements** sections to manage project scope and priorities.
* **For Marketing Staff**: Focus on the **Introduction**, **System Overview**, and **System Features** sections to gain a high-level understanding of the system's purpose, target users, and benefits.
* **For End-Users**: Read the **Use Cases and User Interfaces** section to understand how you will interact with the system and what tasks it enables you to perform.
* **For Testers**: Concentrate on the **Functional Requirements**, **Non-functional Requirements**, and **Use Cases** sections to design test cases that ensure the system performs as specified.
* **For Documentation Writers**: Review the **Functional Requirements** and **Use Cases** sections to create clear user documentation, and refer to the **User Interface** section for details on system interaction.

This document is structured to progressively dive deeper into technical details, making it accessible for various readers based on their specific needs and roles.

**1.4 System Overview**

### **Overview**

The **"CSULA No Ticket 4 Me" Parking Registration System** is a web-based application designed to streamline the process of vehicle registration for parking permits, as well add increased functionality in the form of ticket disputing and other miscellaneous parking needs. This system aims to reduce parking violations by providing users with an easy and efficient way to manage their parking permits while enabling parking authorities to monitor compliance effectively. The software facilitates real-time interaction between users and the parking authority, enhancing the overall parking management experience.

### **Functionality**

The **"CSULA No Ticket 4 Me"** system encompasses several core functionalities, including:

1. **[HIGH] User Registration and Login**: Users can create an account to register their vehicles and access their profiles securely. The system employs secure authentication methods to protect user data.
2. **[HIGH] Vehicle Management**: Users can add, update, and remove their vehicles from their accounts, ensuring that the parking permits reflect current vehicle information.
3. **[HIGH] Permit Application and Renewal**: Users can apply for new parking permits or renew existing ones through an intuitive online process. The system will notify users about permit expiration dates and upcoming renewals.
4. **[LOW] Notifications**: Users receive automated notifications regarding their registration status, permit expirations, and any parking violations through email and SMS alerts.
5. **[Medium] Violation Management**: Parking authorities can issue electronic citations for unregistered vehicles or parking violations. Users can view their citations and any associated penalties within their accounts.
6. **[LOW] Real-Time Parking Availability**: If integrated with parking facilities, the system can provide real-time data on available parking spaces, helping users find suitable parking locations quickly.
7. **[LOW]** **Data Reporting and Analytics**: The system allows parking authorities to generate reports on parking usage, permit registrations, and violations, aiding in data-driven decision-making and resource allocation.

### **Overall System Design**

The design of the **"CSULA No Ticket 4 Me"** system follows a modular architecture approach, ensuring flexibility, scalability, and ease of maintenance. The system comprises the following key components:

* **Frontend Interface**: A user-friendly web and mobile interface designed using modern UI/UX principles, ensuring an intuitive experience for both users and administrators. Technologies such as HTML, CSS, and JavaScript frameworks (e.g., React or Angular) are utilized to create a responsive and accessible interface.
* **Backend Services**: A robust backend system developed using server-side technologies (e.g., Node.js, Python, or Java) that handle business logic, user authentication, and data processing. This layer interacts with the database and manages API requests.
* **Database Management**: A relational database (e.g., MySQL, PostgreSQL) is employed to securely store user information, vehicle records, permit details, and violation data. Proper database design ensures data integrity and optimized queries for quick data retrieval.
* **Integration Layer**: The system may integrate with external services for functionalities such as SMS notifications and payment processing. APIs will be utilized to facilitate communication between the system and external services securely.
* **Security Measures**: The system incorporates security best practices, including encryption of sensitive data (e.g., SSL for data transmission), user authentication protocols, and regular security audits to protect user data and maintain compliance with data protection regulations (e.g., GDPR).

### **Basic Design Approach**

The basic design approach of the **"No Ticket 4 Me"** system emphasizes:

* **User-Centric Design**: Prioritizing user experience and usability to ensure that users can navigate the system easily and complete tasks with minimal effort.
* **Scalability**: Designing the system to accommodate future growth, such as increased user load or additional features, without significant redesign.
* **Modularity**: Employing a modular architecture to allow independent development, testing, and deployment of different components of the system.
* **Maintainability**: Writing clean, well-documented code and utilizing design patterns that facilitate easy updates and modifications.

### **Conclusion**

The **"CSULA No Ticket 4 Me" Parking Registration System** is designed to provide an efficient solution for vehicle registration and parking management. With its comprehensive set of functionalities, user-friendly interface, and robust architecture, the system aims to enhance the parking experience for users while enabling authorities to manage parking effectively. Through careful attention to design principles and user needs, the system seeks to minimize parking violations and improve overall compliance.

**2. Design Considerations**

* Possible Integration constraints with DMV repository.
* Ticket payment and dispute integration challenges.

**2.1 Assumptions and Dependencies**

Assumptions and dependencies of software in use include but are not limited to:

* Swift Payment API dependency
* MySQL dependency
* Google Sign-in API dependency
* Windows 10 Operating System dependency

End-User characteristics expected for use include but are not limited to:

* Young adults ages 18-26
* Enrolled at California State University Los Angeles
* Employed at California State University Los Angeles
* Drivers Licensed
* Average individual with no relative expertise or experience

**2.2 General Constraints**

Global limitations and constraints expected to impact system development include but are not limited to:

* Hardware requirement accessibility: Not all developers may have reliable access to the required hardware to develop.
* End-user environment: Not all developers may have consistent access to a proper work environment.
* Interoperability requirements: The software is expected to work properly on both mobile and desktop viewing environments.
* Security requirements: The software is expected to pass a standard expected security validations such as SQL injections.
* Software requirements: Development is expected to conclude smoothly, however general disruptions due to required software usability is to be expected

**2.3 Goals and Guidelines**

Our software is expected to maintain standards in the sense of goals, guidelines, principles, and priorities where applicable for an improved user experience as well as development standards:

* The K.I.S.S principle:
* The software is being developed in Python to maintain simplicity and speed in development, as performance is not a required factor in development.
* User accessibility:
* The software is expected to feel and function the same as the previous website it is replacing to maintain user familiarity, as well as add increased user-friendliness to improve user experience.
* Development goals;
* The software is expected to be delivered in full by the end of the semester for the Software Engineering CS 3337 class.

**2.4 Development Methods**

The approach used for the design of this software is strictly the Agile Development approach. The key to our use of Agile Development is frequent software testing, short development sprints, and an overall focus of developing quick simple updates.

Specifically, we are using the Scrum Agile framework in order to develop our software, we are functioning as a single development team of 8 members. We maintain a product backlog of features needed to be created and maintained with a product owner assigned to manage the backlog. The scrum values of respect, courage, focus, openness, trust, and commitment are held, along with scrum pillars of transparency, adaptation, and inspection which are maintained.

* What is Agile Development?
* **Author:** OpenText
* **Date:** 2024
* **Source:** OpenText

<https://www.opentext.com/what-is/agile-development>

**3. Architectural Strategies**

* **Architectural Pattern**
* The parking system follows a Client-Server Architecture, utilizing a rest api on the server side and a Single-Page Applicationapproach on the client side.
* **Reasoning**: This architecture promotes separation of concerns and scalability, making it easy to manage and extend components independently. The REST API handles all data interactions, ensuring a clean and secure interface between the client and server.
* **Alternatives Considered**: A monolithic idea was discussed but it seemed too hard to update or expand depending on what we needed in the future.
* **Programming Language and Frameworks**
* **Backend**: The server is developed using python because it is the easiest language to use for our goal. It also includes Flask which we will most likely be using.
* **Frontend**: Jinja2 Framework for Python is the tool being used for front end development.
* **Reasoning**: Python being everyone's most proficient language and best use for what we’re trying to make with a plethora of documentation, as well as being one of if not the fastest languages to work in.
* **Alternatives Considered**: **Node.js** for the backend was considered, but Flask was chosen due to our familiarity.
* **Reuse of Existing Software Components**
* **Reasoning**: To speed up development and reduce costs, existing libraries and APIs are utilized where possible:

Stripe/Swift Payments API for payment processing.

Google Sign-In API for quick and secure user authentication.

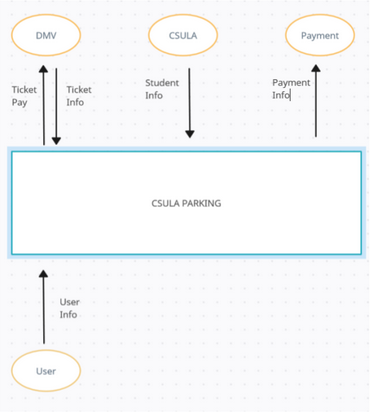
MySQL for managing user and vehicle data.

* **Error Detection and Recovery**
* . The client application displays user-friendly messages based on these error responses.
* **Reasoning**: Centralized error handling simplifies debugging and ensures a consistent user experience.
* **Alternatives Considered**: Distributed error handling was considered but could complicate tracking and resolving issues across multiple components.
* **External Database Management and Data Persistence**
* The system uses MySQL for managing all user data with retrieving inputting and any functions not listed here.
* **Reasoning**: MySQL is one of the most popular database management softwares to date that we all have experience using and can manage without much needed required learning or knowledge.
* **Alternatives Considered**: None.
* **Concurrency and Synchronization**
* The system uses Asynchronous Processing for handling any payment requests to ensure the system remains online/stable. Flask will support multiple connections at one time.
* **Reasoning**: Asynchronous processing improves performance and ensures the system remains up.
* **Alternatives Considered**: Synchronous processing was considered but not implemented because it could lead to bottlenecks
* **Communication Mechanisms**
* Rest will be the primary tool for the system.
* Websocket will be used for real time events like notifications and similar things.
* **Reasoning**: RESTful APIs are widely supported and scalable, while websocket is easy to implement and low latency.
* **Alternatives Considered**: None.

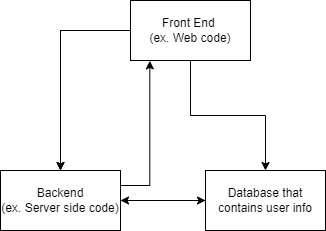
Each significant strategy employed should probably be discussed in its own subsection. Make sure that when describing a design decision that you also discuss any other significant alternatives that were considered, and your reasons for rejecting them (as well as your reasons for accepting the alternative you finally chose).

**4. System Architecture**

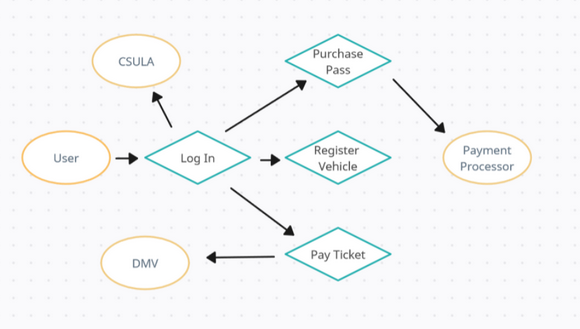
## 4.1 Logical View



## 4.2 Development View

Organized into 3 main modules being the Frontend code, backend, and finally the database. The front end displays all the info the user will need. The backend contains any API connections and code needed to make the system work. The database contains any existing information needed that has been previously entered.

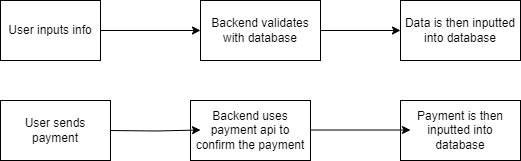


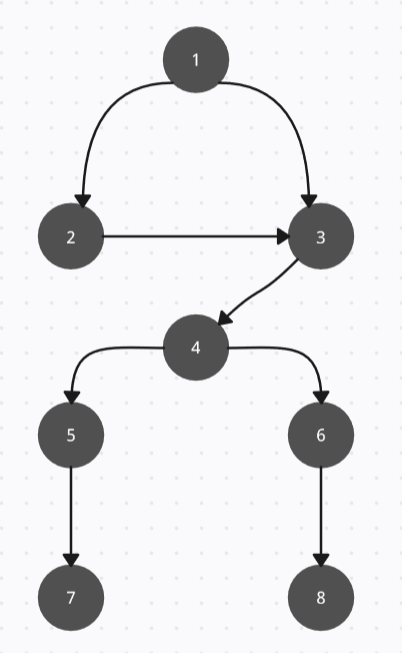
Level 1 Data Flow Diagrams (DFD) 

## 4.3 Process View:

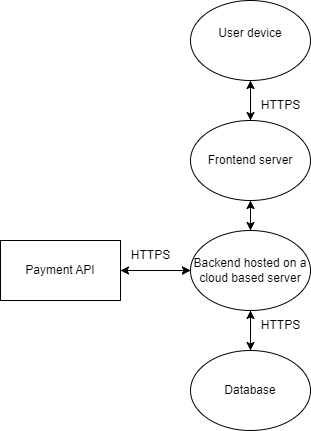
User Registration Process: User inputs information → data is sent to backend API → API validates and stores data in the database.

Payment Process: User initiates payment → frontend sends data to backend → backend calls Payment API → backend updates database with transaction info.





## 4.4 Physical view



**5. Policies and Tactics**

### 5.1 Choice of Specific Products Used

* IDE: By using VS Code which can help out with the web based parking system. This would help out with the startup.
* Database: By using SQLAlchemy as the ORM with a relational database which would align with the backend structure of the project.
* Libraries: By using Flask for routing, bcrypt for security, JavaScript for frontend validation, and HTML or CSS for the interface. These correctly represent the technologies and frameworks used for developing the software.
* Interpreter: By using Python which would ensure consistency with the project structure, because the back end uses Flask.

### 5.2 Plans for Ensuring Requirements Traceability

* This ensures traceability through the use of JIRA or GitHub Issues for managing requirements as user stories or tickets. It works hand in hand with the Agile workflow by following sprints and incremental development throughout the time. That way each feature can be traced back to a functional requirement ensuring everything was implemented according to the initial specification.

### 5.3 Plans for Testing the Software

* Unit Testing: These would include user authentication login and registration and form validation. Since these are considered the major features you have implemented on the project.
* Integration Testing: Interaction between the front end (HTML/JS forms) and the back end (Flask routes and database) are an important key structure critical for the project for proper form of submission and retrieval of user information.
* UAT: Average end users can use the system to sign up, log in, submit vehicle details, and buy parking permits.

**6. Detailed System Design**

**6.x Name of Component (Module)**

**7. Detailed Lower level Component Design**

**7. x Name of Class or File**

**8. Database Design**

### 8.1 Introduction to Database Design

The "No Ticket 4 Me" Parking Registration System relies on a structured database to manage critical data related to users, vehicles, parking permits, and notifications. The database ensures the integrity, security, and scalability of the system by efficiently storing and retrieving data as needed for various features such as permit applications, renewals, and notifications. This section describes the database schema, relationships between entities, and design considerations.

### 8.2 Database Design Overview

The system employs a relational database, using MySQL, to store and manage data. It follows standard relational database principles to ensure data consistency and supports CRUD (Create, Read, Update, Delete) operations. Key entities in the system include Users, Vehicles, Permits, and Notifications. Each entity has specific attributes relevant to the management of parking permits, and relationships between these entities are designed to ensure optimal data access and integrity.

### 8.3 Entity-Relationship Diagram (ERD)

The following is an ERD showing the relationships between the key entities in the database:

* **Users** (one-to-many) -> **Vehicles**
* **Vehicles** (one-to-many) -> **Permits**
* **Users** (one-to-many) -> **Notifications**

(Include an ERD diagram here, or if you can’t provide one, mention that it will show the tables and relationships.)

### 8.4 Tables and Descriptions

#### 8.4.1 Table: Users

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| UserID | INT (PK) | Unique identifier for each user |
| Username | VARCHAR(50) | Username for login |
| Email | VARCHAR(100) | Email address for notifications |
| PhoneNumber | VARCHAR(15) | Phone number for SMS notifications |
| Password | VARCHAR(255) | Encrypted password for user login |
| IsAdmin | BOOLEAN | Flag to determine if user is an admin |

#### 8.4.2 Table: Vehicles

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| VehicleID | INT (PK) | Unique identifier for each vehicle |
| UserID | INT (FK) | Links to the user who owns the vehicle |
| LicensePlate | VARCHAR(20) | License plate number of the vehicle |
| Make | VARCHAR(50) | Vehicle manufacturer (e.g., Toyota, Honda) |
| Model | VARCHAR(50) | Model of the vehicle (e.g., Camry, Civic) |

#### 8.4.3 Table: Permits

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| PermitID | INT (PK) | Unique identifier for each parking permit |
| VehicleID | INT (FK) | Links to the vehicle associated with the permit |
| PermitType | VARCHAR(50) | Type of permit (e.g., Daily, Monthly, Annual) |
| IssueDate | DATE | The date the permit was issued |
| ExpirationDate | DATE | The expiration date of the permit |

#### 8.4.4 Table: Notifications

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| NotificationID | INT (PK) | Unique identifier for each notification |
| UserID | INT (FK) | Links to the user receiving the notification |
| NotificationType | VARCHAR(50) | Type of notification (e.g., Email, SMS) |
| Message | TEXT | The content of the notification message |
| SentDate | DATETIME | The date and time the notification was sent |

### 8.5 Relationships Between Tables

The relationships between the tables ensure data consistency and allow for efficient data retrieval:

* **Users to Vehicles**: A single user can register multiple vehicles (one-to-many relationship).
* **Vehicles to Permits**: A vehicle can have multiple parking permits associated with it over time (one-to-many relationship).
* **Users to Notifications**: A user can receive multiple notifications, including emails and SMS messages (one-to-many relationship).

### 8.6 Constraints and Indexes

To ensure data integrity, the following constraints are applied:

* **Primary Keys**: UserID, VehicleID, PermitID, NotificationID are primary keys, ensuring each record is unique.
* **Foreign Keys**: UserID in the Vehicles and Notifications tables links to the Users table. VehicleID in the Permits table links to the Vehicles table. These foreign keys maintain relational integrity.
* **Indexes**: Indexes are applied on frequently searched columns like LicensePlate in the Vehicles table and Email in the Users table to optimize query performance.

### 8.7 Data Access and Security

* **Data Encryption**: Sensitive data like passwords is stored in encrypted format using strong hashing algorithms (e.g., bcrypt) to protect user information.
* **Access Control**: User access to the database is restricted based on roles (admin vs. regular users). Admins can perform all CRUD operations, while regular users are limited to actions related to their own data.
* **Audit Trail**: The system maintains logs of all actions performed in the database for accountability and security purposes.

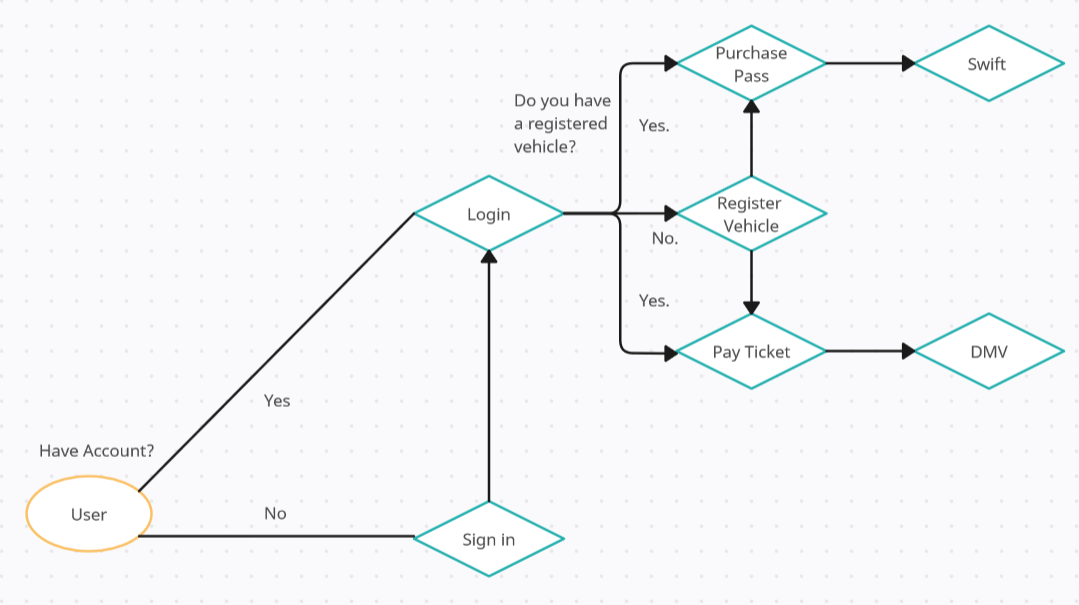
### 8.8 Backup and Recovery

The database is configured to perform automated daily backups to prevent data loss. In case of a failure, the system will restore the most recent backup to minimize data loss. Additionally, transaction logs are maintained to ensure that the system can recover from any incomplete transactions or errors.

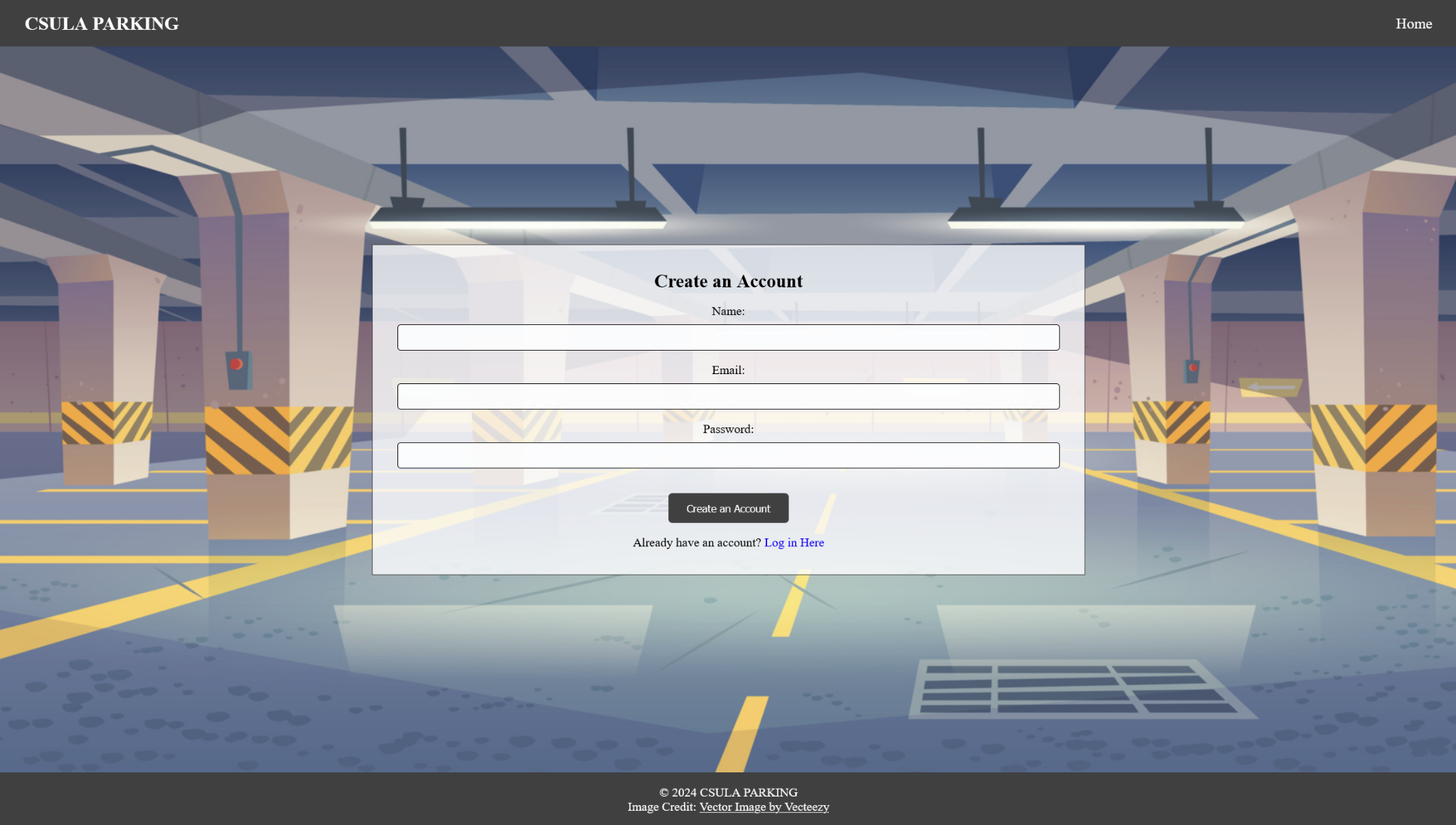
**9. User Interface**

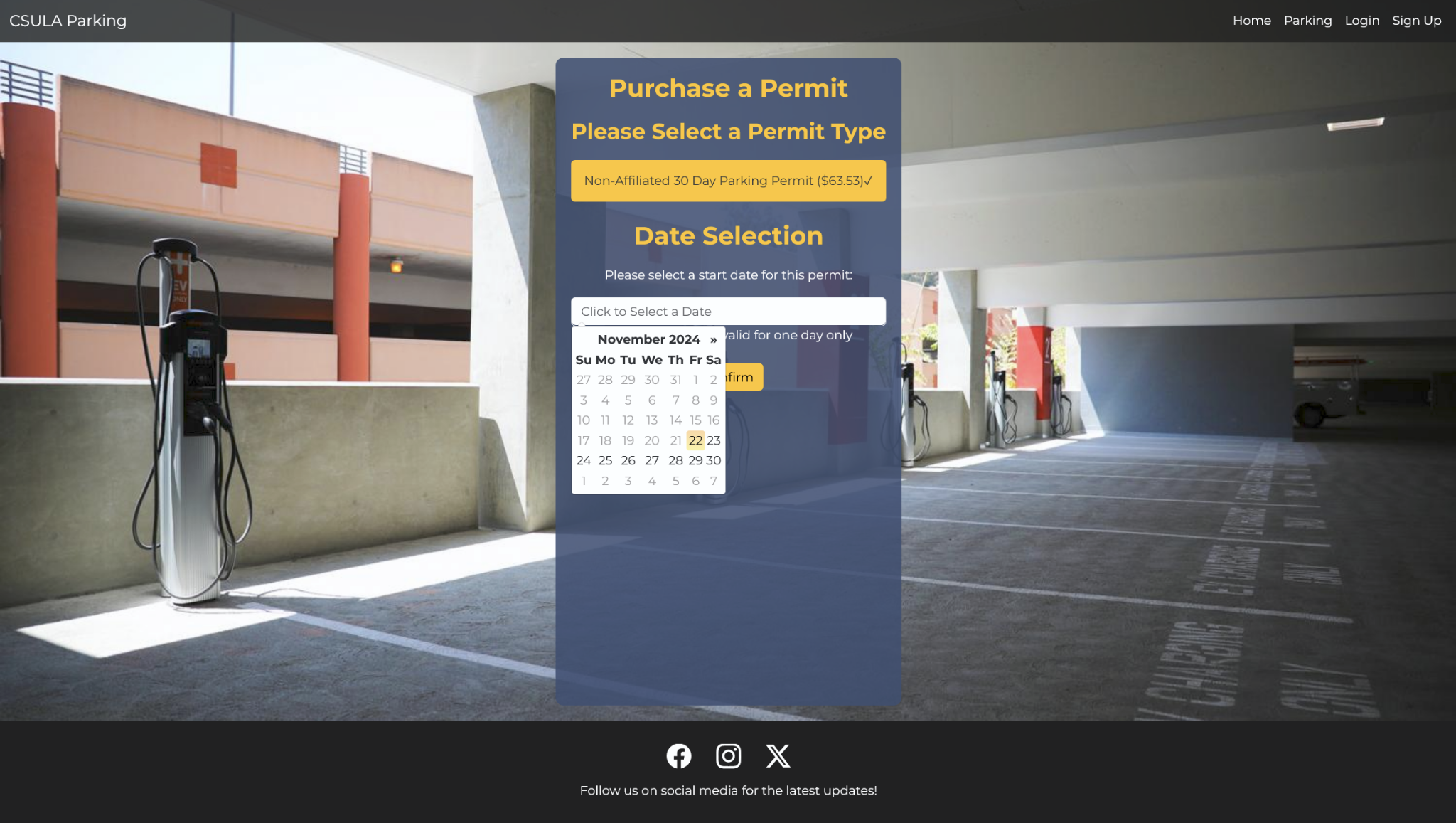
Our UI does have restrictions built in to increase simplicity, specifically in the sense of drop down menus and the like being standard with little room for free user will except in registration

forms.



The sign-in and login form allow the user to type into a form to submit their information in order to the website, with user actions being limited to typing.

Similar to the user sign-up and login page, the vehicle registration form allows the user to input their vehicle information to link to their account in order to purchase a pass or pay an outstanding ticket. The user does however have another action in the form of interacting with a drop down menu to select the time frame of which they would purchase for their parking pass.



**9.1 Overview of User Interface**

Describe the functionality of the system from the user’s perspective. Explain how the user

will be able to use your system to complete all the expected features and the feedback

Information that will be displayed for the user. This is an overview of the UI and its use. The user manual will contain extensive detail about the actual use of the software.

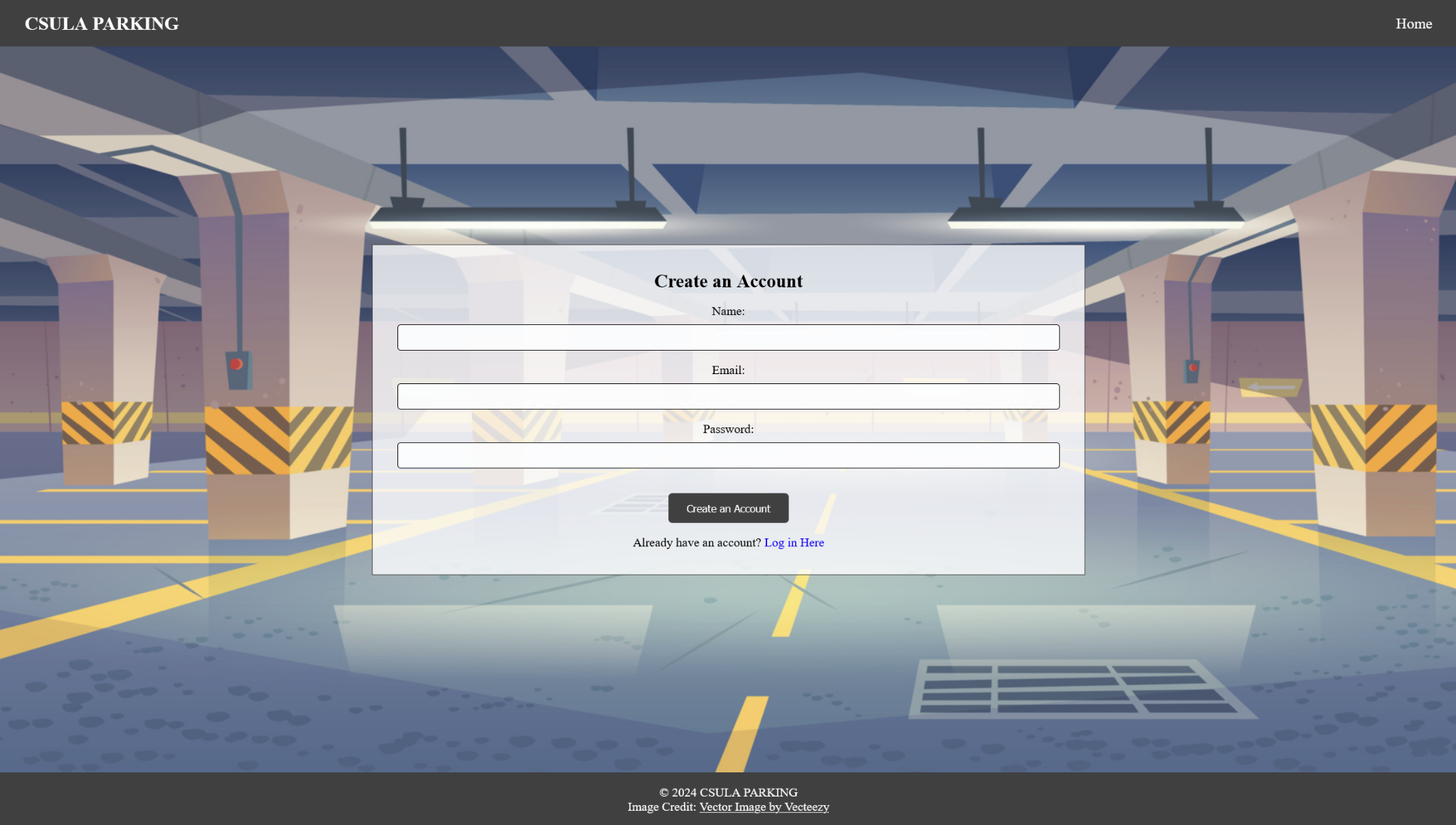
From the user's perspective, the home page has a navigation bar that allows users to click on Home, Parking, Login, or Sign-Up. The user must then decide their option.

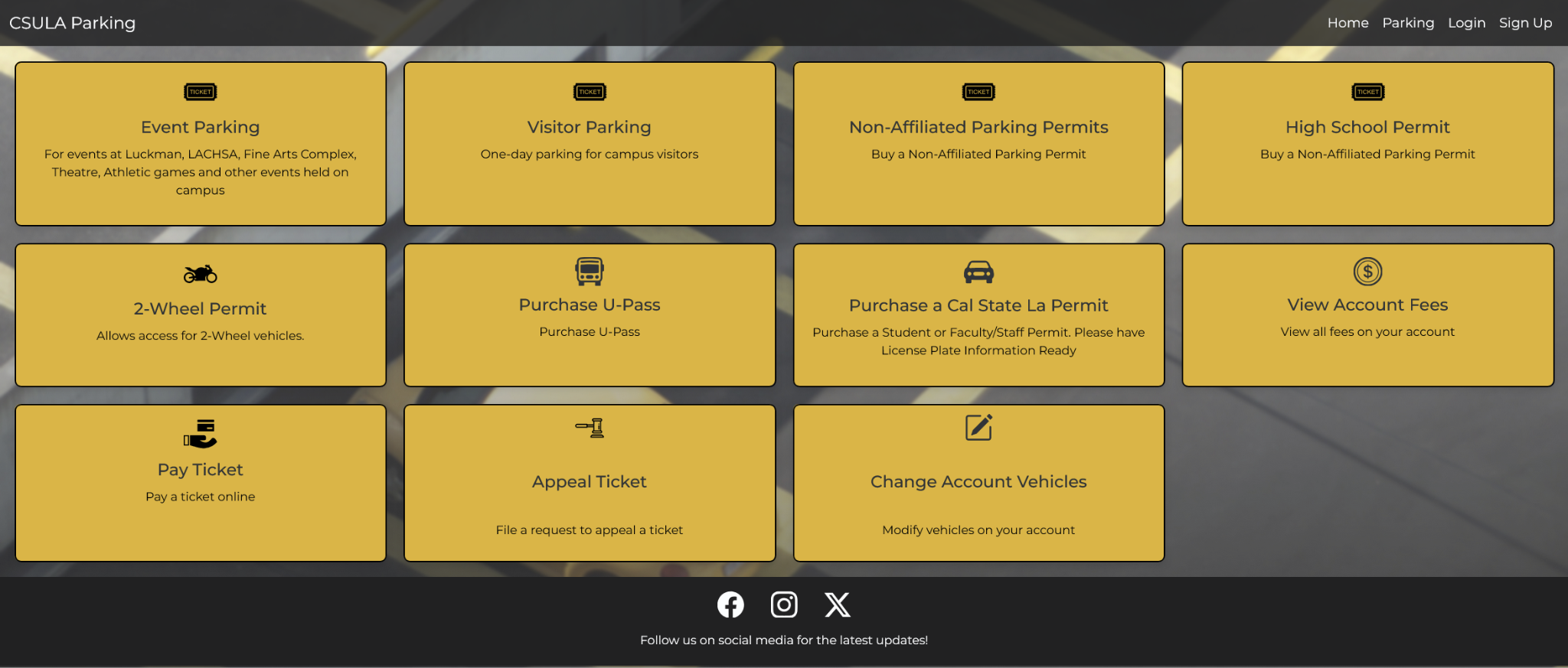
* Home Page:
  + Get started button that redirects to sign up
  + Navigation bar which includes options
* Parking:
  + only accessible if User creates an account
  + Allows user to register for parking
  + Vehicle Information:
    - Model
    - License Plate
    - Color
    - Duration of pass
* Login:
  + validates that the user email/name is a valid one
  + Feedback for user such as email is not registered or incorrect password
* Sign-up:
  + Allows user to sign up
  + Sign-up with google option
  + Options to remember User

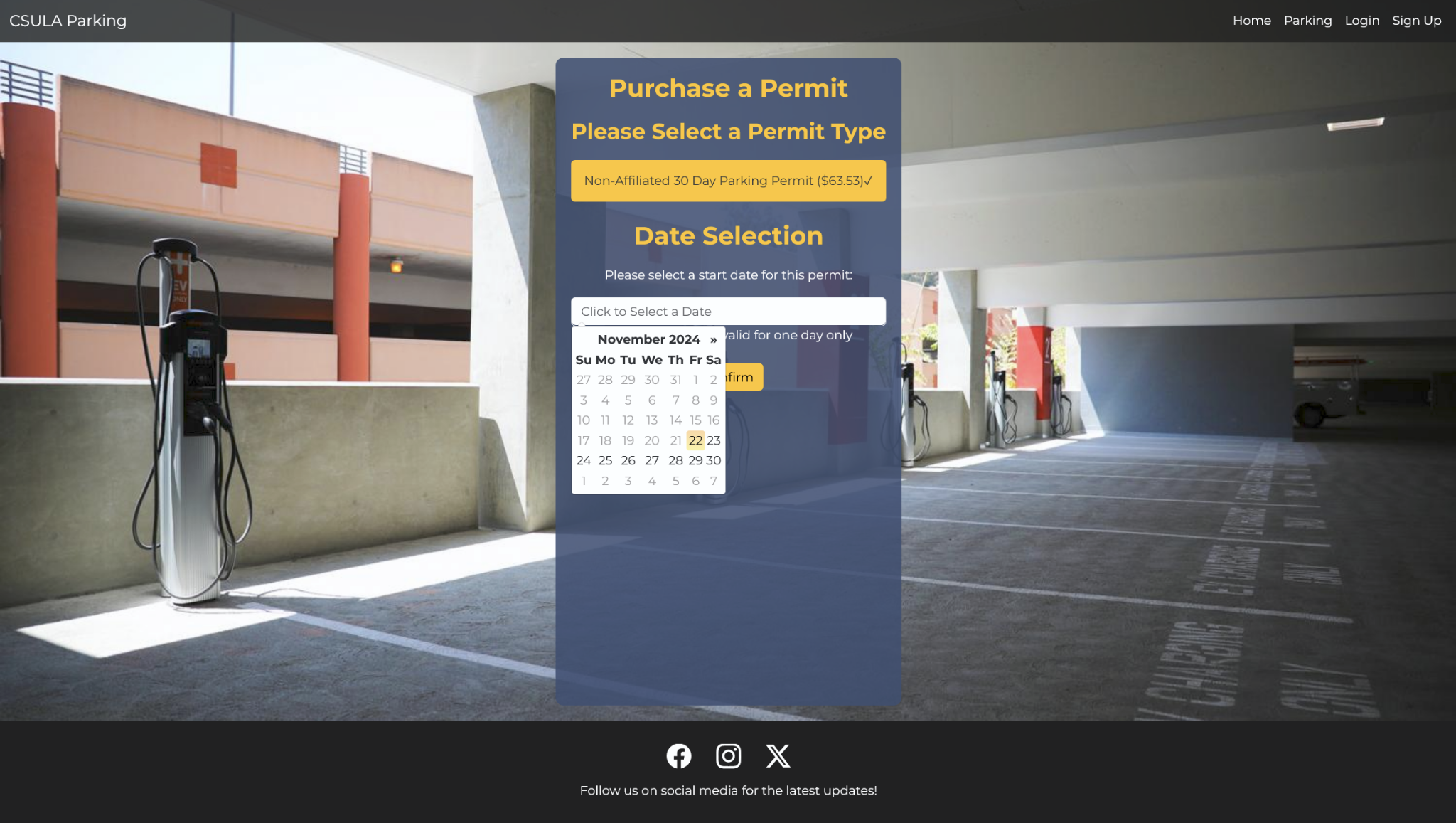
**9.2 UX Standards**

* Responsive Design:
  + User has some visual indication that they are on a section of the website or that it is loading
  + Error messages for things such as email/passwords
* Design:
  + Simple design that is easy to follow
  + Has alt text for users with disabilities

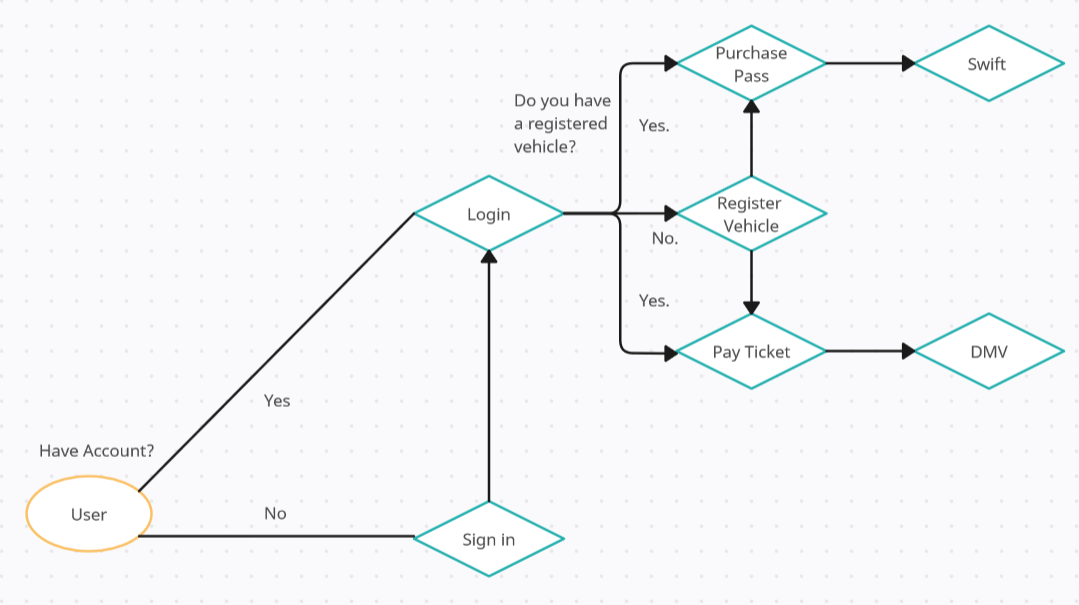
**9.3 Screen Frameworks or Images**







**9.4 User Interface Flow Model**



**10. Requirements Validation and Verification**

| Requirement | Component Modules |  | Testing Method |
| --- | --- | --- | --- |
| **User Registration**: Users must be able to create an account with an email and password. | UserController, SQLAlchemy | sign-in.html | **Unit Testing**: Test the form submission and verify the database stores user data. **Integration Testing**: Ensure the frontend and backend communicate correctly, and new users are added to the database. |
| **User Login**: Users must log in with their email and password. | AuthController, SQLAlchemy, bcrypt | login.html | **Unit Testing**: Test that passwords are hashed and verified correctly using bcrypt. **Integration Testing**: Ensure the session starts and redirects to the user dashboard after a successful login. |
| **Vehicle Registration**: Users must input vehicle details | VehicleController, SQLAlchemy | | parking-details.html | | --- |  |  | | --- | | **Unit Testing**: Test the form validation for required fields. **Integration Testing**: Verify vehicle details are stored in the database after submission. |
| **Parking Permit Purchase**: Users must be able to purchase a parking permit. | PaymentController, SQLAlchemy | parking-details.html, Stripe API | **Unit Testing**: Test the payment form submission and interaction with the payment gateway. **Integration Testing**: Ensure that a successful transaction leads to a parking permit being issued and stored in the database. |
| **Session Management**: Users should be logged out after 15 minutes of inactivity. | SessionController, Flask Session | |  | | --- |  | No UI element | | --- | | **Unit Testing**: Test the session timeout logic. **Integration Testing**: Verify that users are logged out and redirected to the login page after inactivity. |
| **Password Hashing**: User passwords must be securely hashed before storage. | | AuthController, bcrypt, SQLAlchemy | | --- |  |  | | --- | | |  | | --- |  | No UI element | | --- | | **Unit Testing**: Verify that plain text passwords are hashed before storage in the database and tested correctly upon login. |
| **Data Encryption**: Sensitive user data must be encrypted during transmission. | Flask, SSL/TLS | |  | | --- |  | No UI element | | --- | | **Security Testing**: Ensure all user data is encrypted during transmission by using HTTPS. Conduct vulnerability scans for insecure transmissions. |
| **Error Handling**: The system must provide clear error messages for invalid input. | ErrorController, Flask | Forms in sign-in.html, login.html | | **Unit Testing**: Test that the correct error messages are shown when required fields are left blank or input is invalid. | | --- |  |  | | --- | |
| **Role-Based Access Control**: Admin users must have access to user and vehicle management tools, while regular users should not. | RBACController, SQLAlchemy | Admin Dashboard | **Unit Testing**: Verify that only admin users have access to the relevant features. **Security Testing**: Ensure regular users cannot bypass the access control to reach admin functions. |

**11. Glossary**

**API (Application Programming Interface):** A set of functions and protocols that allows different software applications to communicate with each other. In this project, APIs are used for external services such as Swift Payments and Google Sign-In.

**CSS (Cascading Style Sheets):** A stylesheet language used for describing the presentation of a web page. CSS is used to style the user interface of the parking registration system.

**AWSS (Amazon Postgres):**

**HTML (HyperText Markup Language):** The standard markup language for creating web pages. HTML is used to structure the content on the parking system's web interface.

**HTTPS (HyperText Transfer Protocol Secure):** A protocol used to securely transfer data over the internet. The system uses HTTPS to encrypt communications between the client and server.

**MySQL:** A popular open-source relational database management system used to store and manage user, vehicle, and parking data.

**ORM (Object Relational Mapping):** A technique that allows developers to interact with the database using object-oriented programming languages. SQLAlchemy is used as an ORM in this system.

**PostgreSQL:** A powerful, open-source relational database system. It may be used in future iterations of the project for better scalability.

**SQL (Structured Query Language):** A standard language for managing and querying databases. SQL is used to interact with the MySQL database.

**SQLAlchemy:** A Python SQL toolkit and ORM used in this project to manage interactions with the database.

**SSL/TLS (Secure Sockets Layer / Transport Layer Security):** Cryptographic protocols designed to provide secure communication over a computer network. SSL/TLS is used to secure communications in the parking system.

**Swift Payments API:** An external API used for handling payment transactions in the parking system.

**UI (User Interface):** The visual part of a software application through which users interact with the system. The parking system's UI includes web pages for registering vehicles, paying for parking, and managing permits.

**User Session:** A period during which a user interacts with the system. Sessions are used to maintain state and track authenticated users.

**Vehicle Registration:** The process by which users register their vehicle details, such as license plate and make, within the system.

**12. References**

<List any other documents or Web addresses to which this SDD refers. These may include other SDD or SRS documents, user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

Brad Appleton <brad@bradapp.net>  <http://www.bradapp.net>

<https://www.cs.purdue.edu/homes/cs307/ExampleDocs/DesignTemplate_Fall08.doc>