Software Design and Development

Design Architecture

October 25th, 2023

Valet Buddy

Airport Valet Car Locator Mobile App

Project Team

Our team is Team 1 comprised of the following students:

- **Jordan Treutel**
- **Richard Hoehn**
- **Ian Hurd**
- **Patrick Burnett**

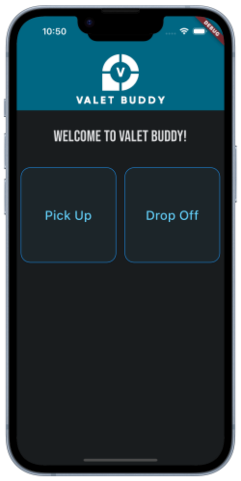
Introduction

This document serves as the basis for the MVC (Model-View-Controller), which is a widely used architectural pattern in software development, designed to enhance the organization and maintainability of code in applications. In our case of the Valet Buddy, a mobile app for iOS and Android that allows a Traveler (car owner) to drop off their car with a Valet (driver) that parks the car and sets a GPS location for the parked car based on their phone's current GPS location.

This MVC design architecture document helps in designing the Models, Vies, and Controllers for our project. It also lists the intended audiences, the risk and their mitigation and provides details on the interactions flows of our Valet Buddy application from both the Server and Client sides.

Our approach is standard in respect to MVC: the Model, represents the data and business logic; the View, responsible for presenting information to users; and the Controller, which manages user interactions and orchestrates communication between the Model and View. This separation of concerns in MVC promotes modularity and easier maintenance, making it a solid process in the engineering of the Valet Buddy for our car owners, drivers, and the mobile that they use.

Intro to App / Project



The project is a mobile app for iOS and Android that allows a Traveler (car owner) to drop off their car with a Valet (driver) that parks the car and sets a GPS location for the parked car based on their phone's current GPS location. In addition to the car's GPS location the associated license plate number, name of the car owner, and picture of the car are also captured during the "Drop-Off" phase.

Another user (Pick-Up-User) can then retrieve the location of the car by entering the license plate number. The storing and retrieval of the car is facilitated by a server that accesses a database for getting and setting GPS coordinates, the license plate and image for later retrieval.

With this app (Valet Buddy), GPS can be used to "Set" and "Get" the car's location and displayed on a user's mobile phone.

The business case of this app is simple... Using technology (specifically GPS) the Airport Car parking owner will not have to "update" and paint numbers on the parking spaces anymore, saving time, and money. Instead, they can rely on the "Valet Buddy" app and backend server to keep track of all the cars that are parked on their lot and quickly and efficiently retrieve them when a car owner returns from the travels at the airport.

Intended Audience

The intended audience of a document refers to the specific group or individuals for whom the document is created or intended.

For this Design Architecture document, the intended audience would be the development company responsible for creating Valet Buddy. This includes:

- **Software Developers:**
    - Who initially implement the system
    - Who maintain the over Valet Buddy application
    - Of neighboring systems who need to know about external interfaces and their technical details
- **Software Architects:**
    - Who need to prepare, shepherd, or implement architectural decisions
    - Who do the design reviews for SOX3 and Security Compliance
- **Quality Assurance Testers:**
    - Who need to perform unit and integration testing within the system
    - Re-Check after features are extended on the Valet Buddy

Summary

Overview

MVC, or Model-View-Controller, architecture is a type of software design architecture that involves breaking the program down into three parts. Models consist of components that have their own logic and interact with some database. Views are components that handle presenting data to the user. Controllers link Models and Views together by managing communication between Models and Views.

Benefits

The benefits of using MVC (Model-View-Controller) are plentiful, however three main points are crucial to a good development strategy using MVC:

- The MVC architecture promotes a clear separation of functional areas in software development, enhancing code organization and maintainability by dividing an application into three distinct components: the Model (data and business logic), the View (user interface), and the Controller (user input and application flow).
- In addition, MVC also facilitates collaborative development by allowing developers to work on different aspects of the application independently, making it easier to manage and scale complex projects.
- And finally, MVC also improves code reusability and testability, as each component can be tested separately, leading to more robust and efficient software development.

MVC Architecture

Models

- **Tickets:** This Model contains data attributes such as license plate number, car owner, valet driver user id, ticket status, gps location, and image file name (ID (uuid) of the ticket)
- **Vehicles:** This Model contains data attributes such as traveler first name, traveler last name, make, model, and license plate number.
- **Valet Drivers (users):** This Model contains data attributes such as a valet driver user id, a first name, a last name, and a password.
- **Owners:** This is a sub-object of the tickets model. It helps with the reporting of the usage of the Valet Buddy system in that we can query how many times an owner might have used the system.
- **Car Images:** This is the image itself that is stored in a “/images/” folder on the server. Each image’s filename is the “ID” of the ticket. Which is a one-to-one relationship. All images are in “*.png” format.

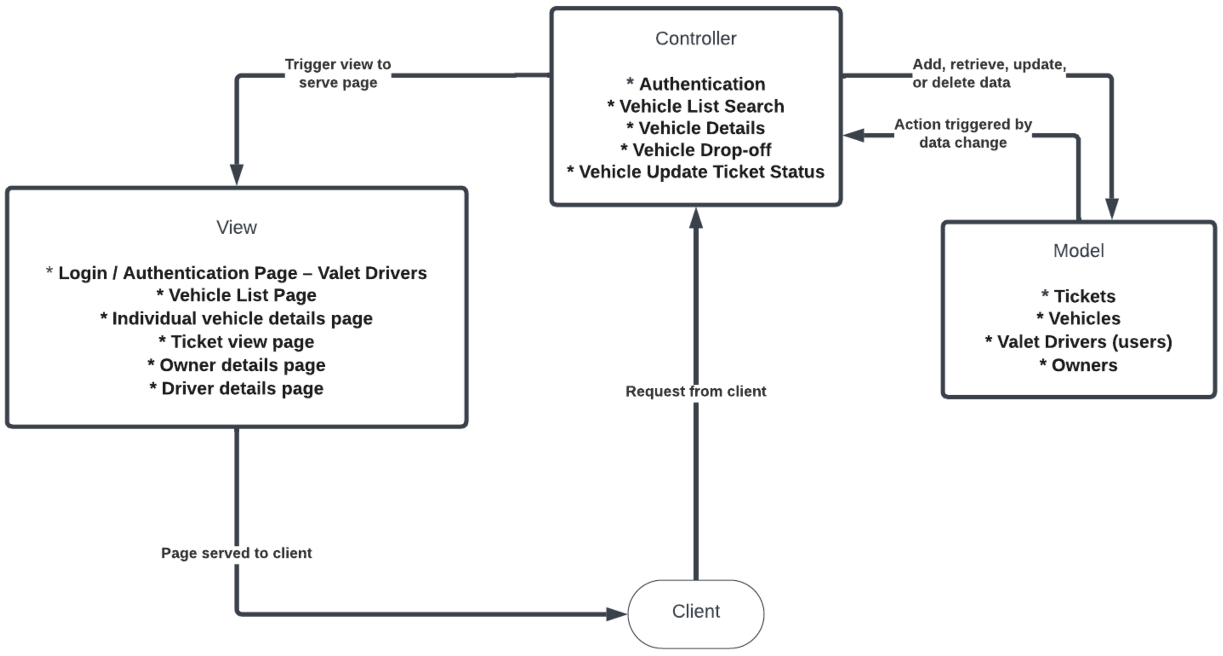
Views

- **Login / Authentication Page – Valet Drivers:** This View contains form fields that allow a guest user to login to their account. The view displays form fields for username and password, but the creation of new accounts will be considered outside the scope of this project due to time constraints of the semester. The page will also display a “Login” button that submits the form data to the Authentication Controller.
- **Vehicle List Page:** This View contains a search bar at the top that allows a user to enter a license plate number or traveler name. More importantly, it also contains a list of tickets from the models. This list is filtered down by the Vehicle List Search Controller as the user types in the search bar to match the text input that the user typed in the search bar.
- **Individual Vehicle Details Page:** This View contains all details relevant to a ticket that the user clicked on from the Vehicle List Page. This structure and the way this view appears depends on the status of the ticket of the vehicle that the user clicked on. For example, if the ticket status is “Parked,” then this view will need to display the image and GPS map location of the vehicle. This functionality will be provided by the Vehicle Details Controller.
- **Drop-Off Vehicle Page:** This View contains form fields for Traveler Name and License Plate Number. It also contains a button to “Begin Parking”. After the valet driver clicks this button, the Vehicle Drop-Off Controller will then update the Tickets model to contain a new ticket with a status of “Parking”.

Controllers

- **Authentication Controller:** Processes user input by collecting username/password from a View form field and comparing it against the models in the system. For the scope of this project, this controller will only be responsible for *checking* the models to see if a user exists, rather than creating a new one. If the user does exist, this controller will log the user in and update the view. More complex responsibilities are outside of the scope of this project.
- **Vehicle List Search Controller:** Processes user input by using it as a filter to match vehicles on the Vehicle List view. As the user types, the view is changed to reflect a new list only consisting of vehicles that match the user input by license plate or traveler name.
- **Vehicle Details Controller:** Processes user input by displaying the correct details and buttons based on the vehicle/ticket that the user selected. If the selected ticket has the status of "Parking", then the only information displayed on the view should be the traveler's name, the license plate number, the status of the vehicle, and an "Add Image & GPS Tag" button. If the selected ticket has the status of "Parked", then this means an image and GPS tag have already been saved for this vehicle, so it would display the traveler's name, the license plate number, the status of the vehicle, an image of the vehicle, a map with a location marker, and a "Retrieve Car" button. If the selected ticket has the status of "Retrieving", then the details page would display only the traveler's name, the license plate number, the status of the vehicle, an image of the vehicle, and a "Close Ticket" button (no longer contains a map of the location since the car is in transport).
- **Vehicle Drop-Off Controller:** Processes user input by collecting form data (Traveler Name and License Plate Number) from the Drop-Off view and updating the Tickets model to contain a new vehicle. This change can later be viewed through the "Vehicle List" page of the app as a new ticket with a status of "Parking".
- **Vehicle Update Ticket Status Controller:** Processes user input (button-click when viewing a specific vehicle ticket details page) as well as information from the Tickets model to determine which status the vehicle should be updated to. If the ticket is currently "Parking" and the driver has just submitted an image and GPS tag, the ticket is updated to "Parked". If the ticket is already "Parked" and a driver needs to retrieve it, then clicking the "Retrieve Car" button will mark the ticket as "Retrieving". If the ticket was already "Retrieving" and the driver has just returned it to the traveler, then clicking the "Close Ticket" button will mark the ticket as "Closed".

Diagram:


Controller
* Authentication
* Vehicle List Search
* Vehicle Details
* Vehicle Drop-off
* Vehicle Update Ticket Status
Trigger view to serve page
Add, retrieve, update, or delete data
Action triggered by data change
View
* Login / Authentication Page – Valet Drivers
* Vehicle List Page
* Individual vehicle details page
* Ticket view page
* Owner details page
* Driver details page
Model
* Tickets
* Vehicles
* Valet Drivers (users)
* Owners
Request from client
Page served to client
Client

Interaction Flow

Login – Interaction

- User enters username and password.
    - If password accepted:
        - User is logged in as their user
    - If password does not match user, username unrecognized, or either field blank:
        - User is informed of issue / password mismatch
        - No login occurs
        - Stays on same login page

Vehicle Search / Filter

- User enters car information into search field
    - List of cars updates to only match search
- User can change filter to only search open tickets
- User can change filter to only search closed tickets

Vehicle Update Status

- User clicks 'begin parking' button
    - User is prompted to enter car identifying information
        - User enters information and clicks submit
            - Ticket created with status 'parking' and vehicle information is added to database.
        - User clicks cancel
            - User returned to main ticket screen with nothing changed
- User clicks 'drop off' button on ticket
    - User prompted to photograph vehicle and confirm GPS tag
        - User enters information and clicks update
            - Ticket updated and photograph and GPS saved
        - User clicks cancel
            - Ticket unchanged and user returned to ticket
    -
- User clicks 'begin retrieval'
    - Ticket status updated to 'retrieving'
- User clicks 'retrieved'
    - Ticket status updated to 'picked-up'
- User clicks 'delivered' button
    - Ticket closed with status 'delivered
- User clicks update-status dropdown menu

- User selects new status and clicks confirm
    - Status updates to what was selected
- User clicks out of dropdown menu
    - Dropdown menu closes with ticket unchanged

Vehicle Details View

The Vehicles details page can be accessed from different sections of the app. There are two main ways to get it.

- First – when a Valet Driver starts a ticket the tickets are created which is also the vehicle details screens. In essence there is a POST to the server that creates the entry in the database. This is then returned with the new "ID" of the ticket and with it the app then makes another call for the details of the vehicle via the returned ID.
- Second – When an owner comes to pick up a vehicle, the Valet driver enters the owner's name or license plate number in the ticket / vehicle listing page. The system gets the ID either from the filter or search and with that ID we can then feed to the details page to pull the vehicle / ticket information.

Risks & Mitigation

Based on the MVC design – what are the risks on such an architecture.

Risk – One Component Affecting Another Component – Modularity

Mitigation for the components affecting each other can be achieved by the following:

- Design based on configuration files over Hardcoding by externalizing configurations like database connection strings, API endpoints, etc., so that they can be changed without modifying the code.
- Another option would be to ensure that modules don't depend on each other in a circular manner. This can be checked using tools that analyze dependency graphs from third-party providers.

Risk – As Project Grows models are tightly coupled

Mitigation for tightly coupled modules that make the system less flexible and harder to maintain or extend. Can be mitigated with following these development strategies:

- **Interface-Based Design** – An option would be to use interfaces to define the interactions between different modules. This ensures that modules interact based on interfaces rather than concrete implementations, promoting loose coupling.
- **Avoid Static Methods and Singletons** - These can create hidden dependencies between classes. Instead, rely on instance methods and dependency injection.
- **Unit Testing** – We can write unit tests for each component that is very isolated. This will ensure that components / modules are decoupled enough to be tested individually.

Meeting Minutes & Notes

The below are simple meeting minutes of the discussions we as a group had during the development of this "Design Architecture" document.

2023-10-18 – In Class Meeting:

Meeting during class. We got the previous document submitted and are now working on the Design Architecture document. We spent time talking about the MVC and how it relates to our project. We worked on some of the Flow and the models, where we discussed how granular we want to go with the model's section. In addition, we also set up the next meeting time for Friday at 10:00am to continue working on this document.

2023-10-20 – Fri @ 10:00am Zoom Meeting:

Spent a good hour discussing the MVC and how to design it. We went back and forth on the MVC diagram and what or if we should use the one from class or one that we design ourselves. We decided to ask her about it and see if she had insight on Monday during or after class.

We also talked about how we should write up the mitigation and what it should entail, even though we might not implement it. We came up with three of them for this.

2023-10-23 – Mon @ 8:40am Zoom Class:

We spent most of the class working on this document. We started off by asking Prof. Ranganathan about the MVC diagram and how we should think about the graphical making of it. In addition, we also spent some time talking about how we should think about listing the Risks & mitigation of the document details... Overall, we decided to focus and higher-level Risk / Mitigation for the MVC approach in itself.

Lots of work was then started on the diagram itself, the introduction and summary were completed. More details are flowing for the completion of the Interaction Flows which still need to be completed.

Some time was spent working on more of the mitigation and we ended up with two of them that we defined. We also spoke about meeting 20min prior to class on Wednesday to review our issues on the MVC and how to present our thoughts to the class.

2023-10-25 – Wed @ 8:20am Before Class:

Decided to spend a bit of time before class working on our Design Document. In specific, we were also thinking about what areas that we had issues with and being able to articulate these in class at 8:40am today.

We also spent time working on the Interaction Flows section to gather our thoughts on it and how best to present this that was closely coupled with the MVC approach in general.

2023-10-25 – Wed @ 9:30am During Class:

We worked on submitting the MVC Challenges document and finalized the Interaction Flows. We ran into some long discussions on what to call the buttons. Overall looks good what we are getting done.