Oregon State University

College of Engineering
Electrical Engineering and Computer Science

Software Innovation Lab

Master's Project Report

Shreya Dhume

SpotFinder

Dedicated parking app for OSU

25th August, 2023

Commencement June, 2024

Abstract

SpotFinder is the mobile frontend of a parking system that helps drivers find a parking spot on campus. (The backend piece of the parking system was developed by others in the lab as part of a previous project.) Finding parking can be viewed as both a search-and-filter problem and a coordination problem. The search-and-filter problem is finding all available parking lots with available parking spots and filtering the most suitable one for the user. The coordination problem is avoiding having two drivers try to park in the same spot. For example, there are 5 drivers and 3 spots available in parking lot A and 2 spots in parking lot B, so all 5 people should not be directed towards the same parking lot. SpotFinder helps solve the search-and-filter problem by computing the most suitable parking lot based on real-time availability and the user's estimated time to reach the parking lot. The coordination problem is solved using a soft reservation concept. Rather than a firm commitment, this approach temporarily holds a parking spot to ensure that 2 people are not directed to the same spot, improving the odds of getting a parking spot. SpotFinder can't control the actual parking lot reservations, but the soft reservation helps to streamline online users in order to solve the coordination problem.

Acknowledgments

I would like to take this opportunity to express my gratitude to all those who have contributed to the successful completion of my project. Firstly, I would like to thank Dr. Will Braynen, for his guidance and constant support throughout.

I would also like to thank my committee members, Dr. Mike Bailey and Dr. Arash Termehchy, for making time in their busy schedules and providing valuable feedback.

Thanks to Arun Sasi, Upanshu Chaudhary, and Kedar Dhere for their continuous support and patience during the technical review process. Their critical feedback and suggestions have been invaluable in improving the quality of my work.

Finally, I would like to thank my parents, sister, and Taher Mulla for their unconditional love and support throughout.

Table of Contents

List of Figures

1. Introduction

Oregon State University (OSU) has over 28,000 students, and around 2,000 faculty and staff.[1] With such a large population, there is a constant demand for parking spaces. Despite having over 45 zonal parking lots on campus[2], finding an available parking space is often a frustrating and time-consuming task. This often results in vehicles being parked in unauthorized areas or on the streets, leading to safety concerns and traffic disruptions. The current parking system is unable to provide real-time updates on the availability of parking spots. Additionally, certain parking zones might close without prior notice, leaving drivers to find out only when they arrive. On the event day, traditional signboards or volunteers indicate whether the parking lot is full or not. The lack of real-time information also makes it challenging for users to plan their parking, leading to longer wait times and increased traffic jams on campus.

To address this issue, the SpotFinder app has been developed. SpotFinder is an Android application that utilizes real-time data from CCTV cameras installed on campus to provide users with information about available parking spots in real-time. SpotFinder search-and-filters the most suitable parking lot for the user based on the user's current location or destination location and spot availability in the parking lot. This enables drivers to easily navigate to the available parking lot. Campus parking is similar to street or public parking, with no reservation system in place. The lack of a reservation system often introduces coordination problems. The coordination problem is avoiding having two drivers try to park in the same spot. For example, there are 5 drivers and 3 spots available in parking lot A and 2 spots in parking lot B, so all 5 people should not be directed towards the same parking lot. SpotFinder uses a “soft reservation” concept, which holds the spot for online users without firm commitment and helps to streamline online users in order to solve the coordination problem.

A user study was carried out to figure out the most driver-friendly interface. With a focus on making the experience as easy as possible for users, the SpotFinder app was created with a straightforward and simple user interface that can be conveniently used while driving. The scope of the project includes the development and testing of the Android app, ensuring that it provides real-time data to users. This report also includes future plans to improve the application and suggestions for the university in order to promote users to use the application and migrate the parking system to online mode making it more reliable and accurate.

2. Existing Solutions

There are a few existing and possible solutions that have been implemented to improve the parking experience. However, all these applications are more generic and aren't specifically tailored to the needs of OSU.

2.1 Google Maps

Google Maps platform[3] is more than a navigation aid. It allows users to search for locations, get driving directions, explore businesses and landmarks, and even view street-level imagery. It also allows users to filter for local amenities, such as parking. It's become an essential tool for drivers, particularly when visiting new areas where parking spaces might be scarce. However, there are some shortcomings in using google maps for finding parking spots:

Lack of real-time updates: Google Maps just shows the parking lot location, it does not give real-time updates regarding parking availability. It might guide you to a parking lot, but there's no guarantee you'll find a spot when you arrive.

Filter limited to current location: Google Maps does not allow you to find parking garages around your upcoming destination. You may have to search for the destination and look for a parking marker in the map fragment.

Top searches show paid garages: Even after applying the filter by default top searches show parking garages. Most of the cases google does not show street parking.

2.2 Parkopedia

Parkopedia[4] app, this is a dedicated platform designed with one key mission in mind - to help you find parking spaces. This application allows users to discover parking options around their upcoming destinations, as well as near their current location. Unlike Google Maps, Parkopedia not only highlights parking lots but also flags potential street parking opportunities nearby.

Lack of real-time updates: Similar to Google Maps - it doesn't tell you real-time parking lot availability. So, it could send you to a parking lot, but it might happen that the parking lot is full.

Reserved parking spaces: Parkoedia also involves listing reserved parking spaces. For instance, it might show the "Hilton Garden" as an available option. However, this lot is specifically reserved for hotel customers, which could mislead users and once reaching there users will realize that the parking space is unavailable to them.

In highly congested urban areas or during peak hours, such an occurrence can cause significant inconvenience and frustration. These continued issues highlight the ongoing need for an innovative parking solution that not only identifies parking locations but also provides real-time updates on their availability.

3. Proposed Solution

SpotFinder is designed to overcome gaps in the existing solutions like lack of real-time parking availability, and aims to provide a more reliable and OSU-centric parking solution parking application.

SpotFinder platform is an application developed using an API designed for the SmartPark project[5] which provides a list of parking lots with real-time parking availability. Based on the real-time availability and the user's location, SpotFinder shows the most suitable parking lot for the user. SpotFinder currently provides only soft reservations, which are tentative reservations without a formal commitment. Soft reservations are done on the server side to solve the coordination problem.

The 'confirmed reservation' system with integrated online payment options will be prioritized in later phases. Future features such as 'reserve online, pay online' will be introduced in subsequent phases to further enhance reliability and user experience

Figure 3.1: Application Logo

3.1 Weighted Scoring

Weighted Scoring[6] is a methodology used to rank or prioritize items according to their distinct value or significance. It involves assigning a unique weight to each item and then generating a weighted score by integrating this weight with other pertinent factors or attributes. Items with higher weighted scores are prioritized over others. This technique is used in decision-making and problem-solving scenarios where optimization is required.

Application of Weighted Algorithm in SpotFinder:

SpotFinder gets parking lot availability data from smart park API and then uses a weighted scoring algorithm to find the best suitable spot for the user. In the case of SpotFinder, we use this Weighted Algorithm to assign weights to each parking lot. These weights are calculated considering two primary factors - the number of available spots in a parking lot and its duration from the user's current location. The algorithm crunches these data points and assigns a weighted score to each parking lot. The parking lot with the highest weighted score is the most suitable based on availability and proximity, which is then suggested to the user.

3.2 Soft Reservation

Parking lots on campus don't currently use a reservation system. Implementing such a system with kiosks or QR code scanners would be costly, considering the number of parking lots across the campus. Due to the lack of a reservation system, coordination problem come into the picture. The coordination problem is avoiding having two drivers try to park in the same spot. For example, there are 5 drivers and 3 spots available in parking lot A and 2 spots in parking lot B, so all 5 people should not be directed towards the same parking lot. The coordination problem is solved using a soft reservation concept. The soft reservation approach temporarily holds a parking spot to ensure that 2 people are not directed to the same spot, improving the odds of getting a parking spot.

Here’s how it works: If a user is predicted to arrive at the parking lot within the next 15 minutes SpotFinder initiates a soft reservation on the server side for that user once the user starts driving toward the parking lot. This is used by the weighted algorithm to calculate the most suitable parking spot for the user. This 15-minute window was chosen because it’s the average travel time within the campus, thus providing a reasonable expectation of arrival. This ensures that reservations are not made too prematurely, taking into consideration the average speed of travel in a city environment. Thus, any user within this 15 mins travel range is reasonably likely to find a parking spot upon arrival. Further, we have accounted for users who stay close to campus in off-campus housing. With these factors, any user heading toward that parking lot is assumed to have a reasonable chance of finding a spot.

In conclusion, the "soft reservation" system represents a strategic and cost-effective solution that improves the odds of users getting parking spots despite of lack of hardware-based reservation systems like kiosks used in parking garages. This results in an improved parking experience for all users at Oregon State University.

4. UI design

4.1 Market Survey - Design and Usability Study

The way an app looks and feels is very important for how people feel about using it. This is especially true for this app, where the design needs to be easy to use while driving. Surveys were conducted to analyze how other parking applications work and interviews were conducted to understand user experience with these apps.

The evaluation was done for the following UI components:

- Landing page
- Search bar
- Floating action buttons

Surveyed applications:

- Google maps [3]
- Uber [7]
- Lyft [8]
- ParkWhiz [9]
- SpotHero [10]

Insights:

- Most apps have a layout similar to Google Maps. This design is familiar to most people and makes the app easy to use.
- The home screens of Google Maps and Lyft automatically show a map of the user's current location or other relevant features for the current location.
- All the apps keep their home screens simple, usually featuring just a search bar where users can type in the location they want.
- Users can swipe up for more details about the location they searched for.
- ParkWhiz and SpotHero have similar designs, but they require users to go through 2-3 steps before they can see parking results. We think this could be a problem. Since users will be driving when they use the app, the design needs to be simple and easy to navigate

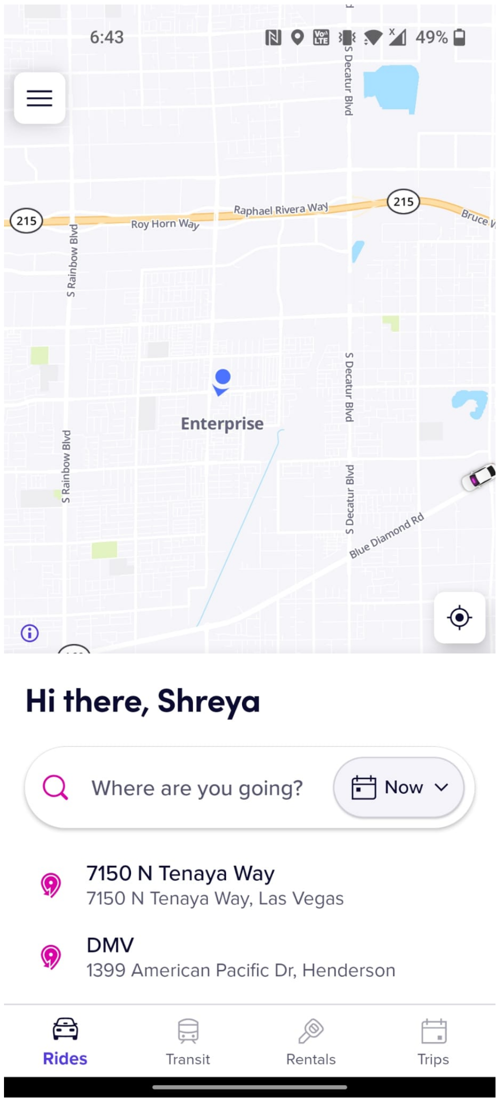


Figure 4.1: Landing page - Lyft

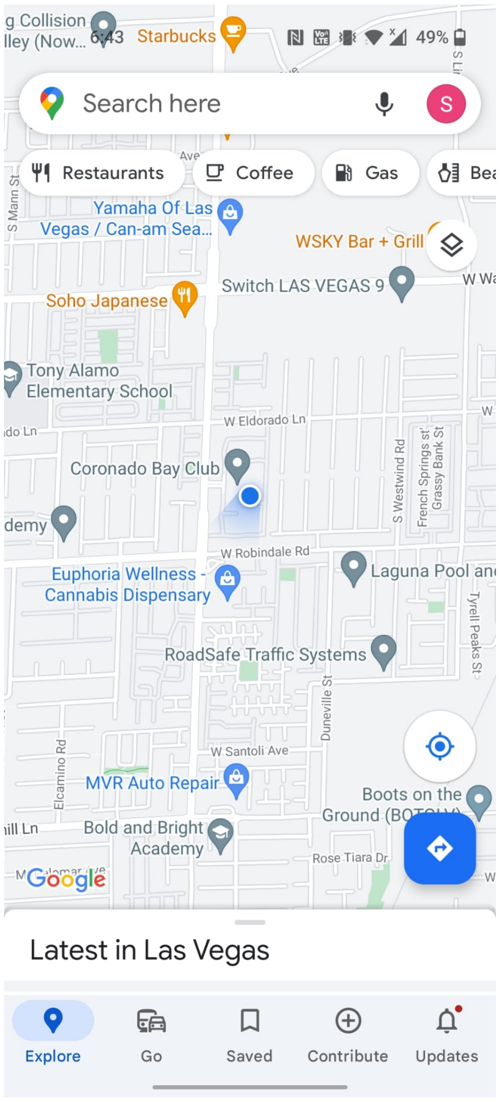
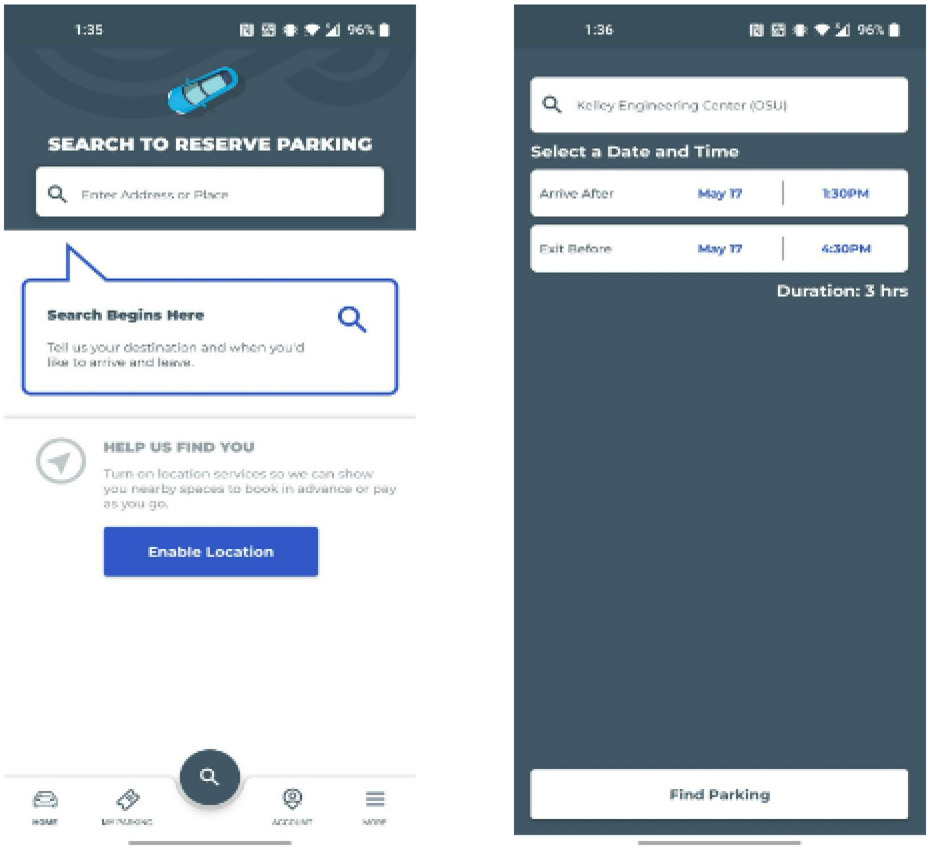


Figure 4.2: Landing page - Google Maps


1:35

SEARCH TO RESERVE PARKING

Enter Address or Place

Search Begins Here

Tell us your destination and when you'd like to arrive and leave.

HELP US FIND YOU

Turn on location services so we can show you nearby spaces to book in advance or pay as you go.

Enable Location

1:36

Kelley Engineering Center (OSU)

Select a Date and Time

Arrive After May 17 1:30PM

Exit Before May 17 4:30PM

Duration: 3 hrs

Find Parking

Figure 4.3: Parkwiz Application Flow

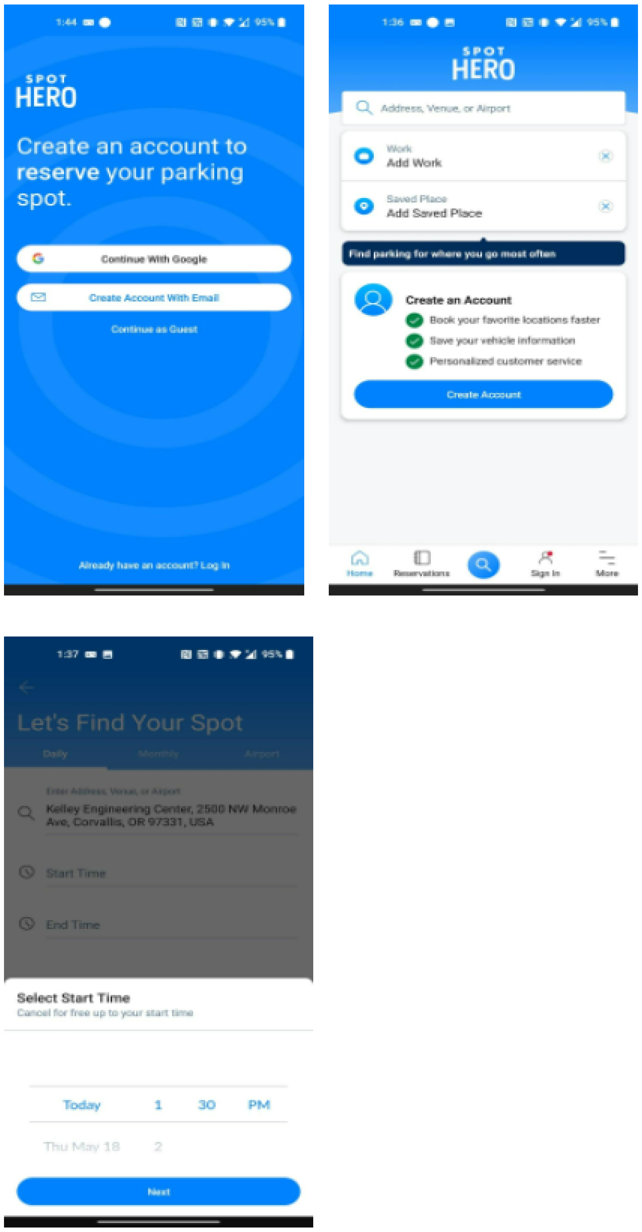

SPOT HERO
Create an account to reserve your parking spot.
Continue With Google
Create Account With Email
Continue as Guest
Already have an account? Log In
Address, Venue, or Airport
Work
Add Work
Saved Place
Add Saved Place
Find parking for where you go most often
Create an Account
Book your favorite locations faster
Save your vehicle information
Personalized customer service
Create Account
Home
Reservations
Sign In
More
Let's Find Your Spot
Daily
Monthly
Airport
Enter Address, Venue, or Airport
Kelley Engineering Center, 2500 NW Monroe Ave, Corvallis, OR 97331, USA
Start Time
End Time
Select Start Time
Cancel for free up to your start time
Today 1 30 PM
Thu May 18 2
Next

Figure 4.4: SpotHero Application flow

Seeing how well these designs worked in other apps, we decided to use a similar look and feel for SpotFinder.

4.2 Figma Prototypes

Based on the market survey and user interviews, an initial paper prototype and Figma mockups were designed. The landing page of SpotFinder is similar to Google Maps, with a map of the best suitable parking lot based on the user's current location, a search bar at the top where users can enter their desired location, and floating action buttons on the right. Unlike other apps, SpotFinder would provide the best parking spot availability results without unnecessary steps or UI, minimizing the UI trap.

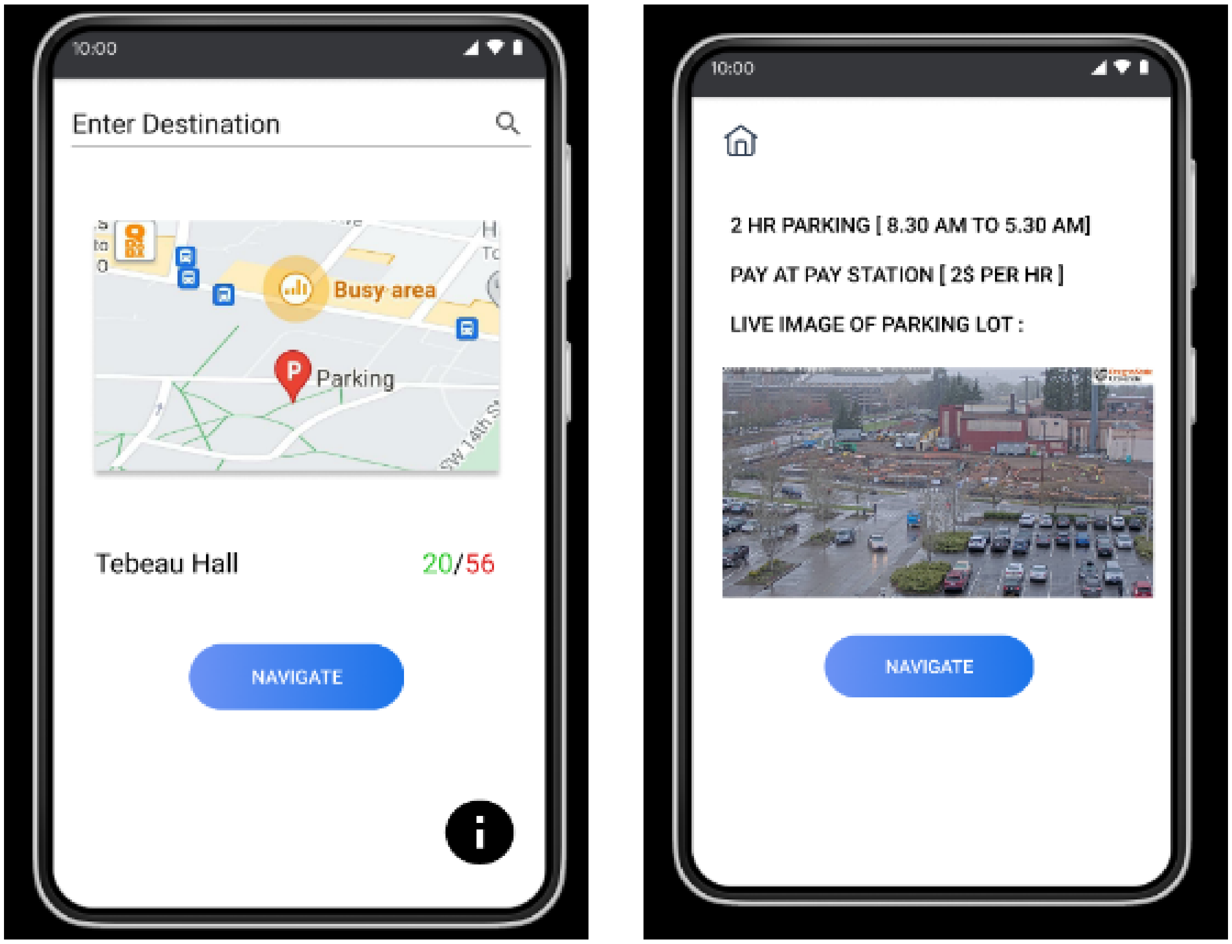

10:00
Enter Destination
Busy area
Parking
Tebeau Hall
20/56
NAVIGATE
10:00
2 HR PARKING [8.30 AM TO 5.30 AM]
PAY AT PAY STATION [2$ PER HR]
LIVE IMAGE OF PARKING LOT :
NAVIGATE

Figure 4.5: Application Home Page

5. Technology Stack

SpotFinder is developed using Android Studio with Kotlin as the primary programming language. Android Studio provides a powerful development environment for building Android applications, while Kotlin offers a modern and concise syntax that enhances developer productivity. The application is powered by a serverless backend service hosted using Amazon Web Services (AWS) and utilizes various AWS service offerings. Additionally, SpotFinder integrates with the Google Maps platform, utilizing several of its services such as the Place API, Google Maps SDK, Translate API, and Distance Matrix API.

5.1 Kotlin

For the same task, Kotlin has much simpler and shorter code as compared to Java. One notable feature is null safety, which helps detect null-related errors at compile time, reducing the likelihood of crashes during runtime. In 2017, Google announced Kotlin as an officially supported language for Android development. This endorsement from Google provided developers with the confidence to adopt Kotlin as their primary language for Android projects. It also ensured that Kotlin would receive ongoing support, updates, and integration with Android development tools and libraries.

5.2 Google Maps Platform

The integration with the Google Maps platform adds powerful mapping and location-based services to SpotFinder.SpotFinder integrates with the Google Maps platform, utilizing several of its services such as the Place API, Google Maps SDK, Translate API, and Distance Matrix API. These services provide features like location-based search, map visualization, translation support, and distance calculations, enhancing the functionality and user experience of the app.

5.3 Node.js

SpotFinder's server is backed by the Node Express stack. Node.js is an open-source server environment that allows developers to run JavaScript on the server. Manually managing all aspects of server management, especially as the application grows, can be challenging. Express, built on top of Node.js. This framework provides a simple API to define routes, middlewares, and handlers with a minimal amount of code. While Node.js gives the capability to manage HTTP requests, Express makes it simpler, more organized, and more maintainable. In this application we use Node.js to efficiently develop our server in Node, ensuring a clean and efficient backend infrastructure.

5.4 MongoDB Atlas

MongoDB is a flexible, document-oriented NoSQL database. MongoDB is a perfect fit for SpotFinder as it allows the application to store, retrieve, and manipulate user data in a fast and efficient manner. MongoDB Atlas is a fully-managed cloud database that handles all the complexity of deploying, managing, and healing the deployments on the cloud service provider like AWS, or Azure. MongoDB Atlas[11] is the best way to deploy, run, and scale MongoDB in the cloud.[10] Moreover, MongoDB was also used in the SmartPark project, this makes it easy for the integration process when we containerize the backend to migrate it to the OSU server.

6. Frontend architecture

6.1 Model-View-ViewModel

Model-View-ViewModel (MVVM) is the separation of concerns. MVVM separates UI or views from the core business logic part of the application. MVVM is fully supported and encouraged by Google with its libraries.[12] MVVM divides an application's structure into three key parts: Model, ViewModel, and View.

Model is responsible for handling the data-related operations of the application and provides data to ViewModel. This can include interacting with databases, performing network requests, handling local storage, or anything else related to the retrieval, manipulation, or storage of data [13].

ViewModel is a bridge between Model and View. The ViewModel is not aware of the specific details of the View; it simply provides data in a format that is ready for display. This abstraction makes it easier to test and change the UI without affecting the underlying business logic. It binds to the UI components through DataBinding, allowing the UI to automatically update when the data changes.

View layers, providing data and functionality to the UI. The View layer observes data changes in the ViewModel and updates the UI accordingly. It does not directly interact with the Model but rather displays data and notifies the ViewModel of user interactions.

The View observes data in the ViewModel, and ViewModel observes data in the model i.e. data coming from the local database and from the network calls. In this hierarchy, each layer has a reference to the layer below it, but not the one above. For instance, the View has a reference to the ViewModel, but the ViewModel does not know anything about the View. ViewModel can expose some data by allowing it to be observed through LiveData [14].

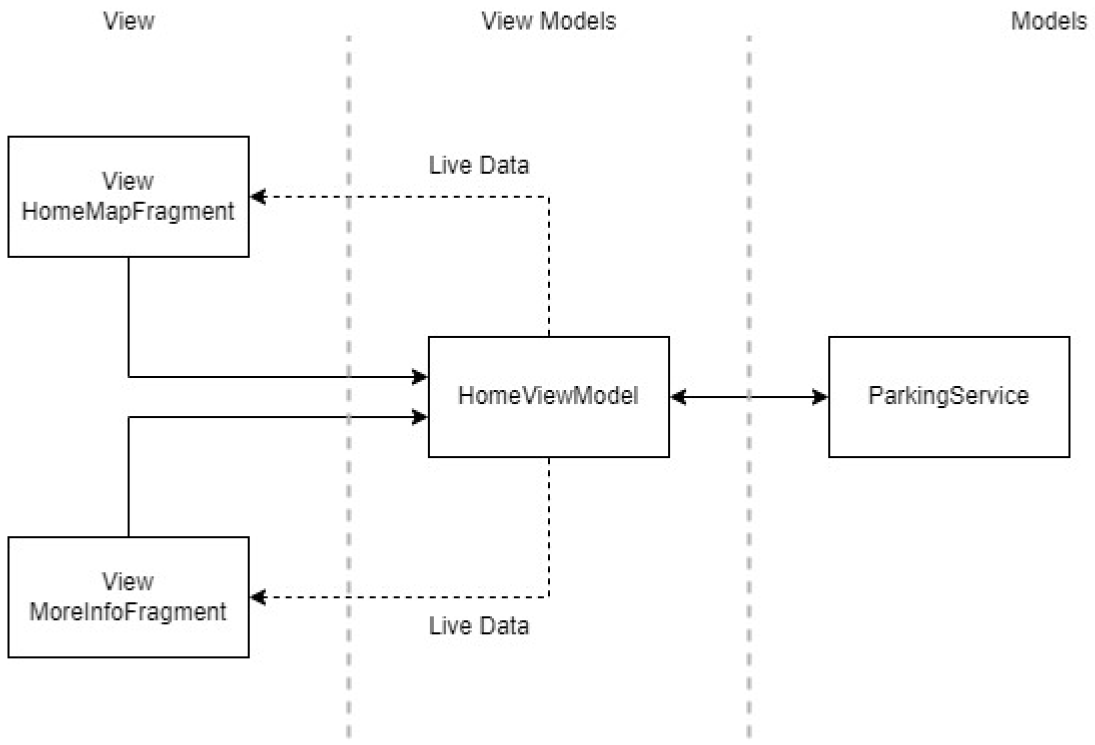


Figure 6.1: MVVM architecture diagram

6.2 LiveData

LiveData is a data holder class. It allows components to subscribe to changes and updates in the data that it holds. LiveData automatically manages subscriptions based on the lifecycle state of the observers [15].

LiveData only updates components that are in an active lifecycle state, which typically means that they're in the foreground and the user can interact with them. When a component is destroyed, for instance when the user navigates away from an activity, the component automatically stops observing the LiveData. If a component becomes inactive, it won't receive updates until it becomes active again.

The ViewModel holds LiveData objects and the View components (activities or fragments) observe these LiveData objects. When the data in the ViewModel changes, the UI updates automatically.

7. API Documentation

7.1 RESTful APIs

7.1.1 Create API

Reserve parking spot:

- [POST] /reserveParkingSpot
- Content-type: application/json
- Soft Reserves parking spot in the parking lot.

7.1.2 Search API

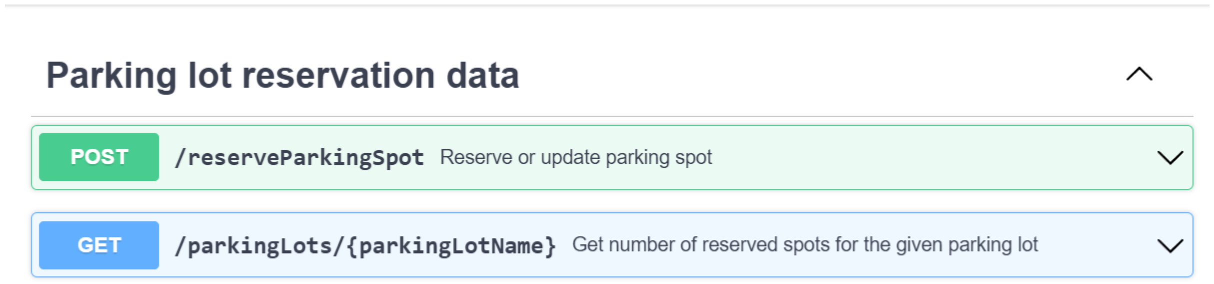
Get the number of reserved parking spots:

- [GET]/parkingLots/{parkingLotName}
- Returns a JSON of the specified parking lot.

7.2 Swagger Document

Swagger is an open-source set of rules, specifications, and tools for developing and describing RESTful APIs [16]. With the Swagger framework, developers can create interactive and comprehensible API documentation for both machines and humans.API details include supported operations, input, and output parameters, and available endpoints. Based on the API code, Swagger can generate this information automatically. Swagger is also used for API testing.

The swagger YAML file for the SpotFinder application can be found here (GitHub).


Parking lot reservation data

POST /reserveParkingSpot Reserve or update parking spot

GET /parkingLots/{parkingLotName} Get number of reserved spots for the given parking lot

Figure 7.1: Swagger document

8. Unit Testing

A unit test is a type of software testing in which the smallest testable parts of an application, called units are tested in isolation from the rest of the application. The main objective is to validate that each unit of the software performs as designed. Unit tests act as documentation by demonstrating how a piece of functionality works and how it can be used. Unit tests help developers to catch and fix bugs at an early stage, which reduces the cost of bug fixes.

JUnit is a widely used testing framework that is included in dependencies by default in Android. It provides annotations to identify test methods and contains assert statements to test expected results.

When unit testing mocking dependencies is important. The purpose of mocking is to isolate and focus on the code being tested and not on the behavior or state of external dependencies. [17] Mocking allows developers to isolate the testing unit from external dependencies such as API calls, databases, and other external services or methods. Mocking replaces real objects with mocked versions that simulate real-world behaviors. SpotFinder uses MockK for mocking external dependencies. MockK is a new open-source library focused on mocking in Kotlin.

Coroutines, introduced in Kotlin, are a way to handle asynchronous tasks smoothly and are a lightweight alternative to threads. Testing coroutines can be challenging because of their asynchronous nature. Kotlin provides TestCoroutineDispatcher and TestCoroutineScope to make testing coroutines easier. TestCoroutineDispatcher allows for the execution of coroutines synchronously in a testing environment, making tests predictable and easy to debug.

9. Scope & Current Limitations

The current scope of SpotFinder includes the following considerations and constraints:

- Offline Drivers:
  In terms of the ’soft reservation’ feature, the current system doesn’t account for drivers who aren’t using the SpotFinder app. This means that some parking spots might be occupied by drivers not represented in our system, which could impact the chances of online use getting a spot.
- User API:
  At present, the API to calculate the number of users for each parking lot is running on a local server. As part of our development plan, we aim to host this API on OSU server together with the SmartPark API.

While these limitations define the current boundaries of SpotFinder, they also provide a roadmap for future improvements. In the "Future Development" section at the end of this report, we’ll discuss potential new features and the possibility of transitioning the OSU parking system to an online platform. This would provide more accurate and reliable data, enhancing the overall functionality and usefulness of SpotFinder.

10. Application Screenshots

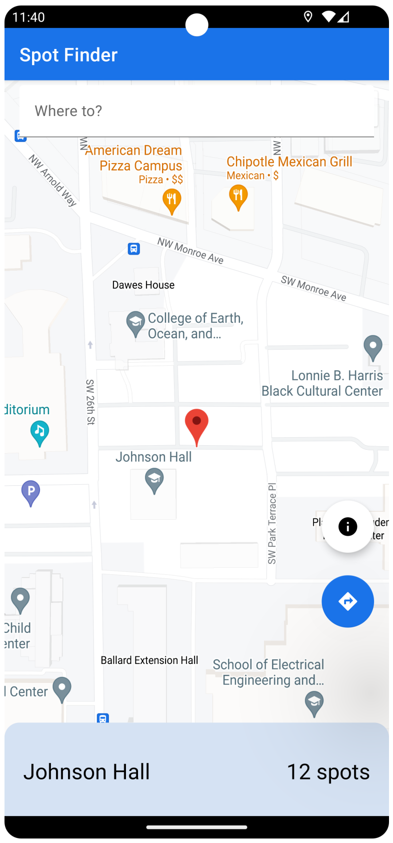
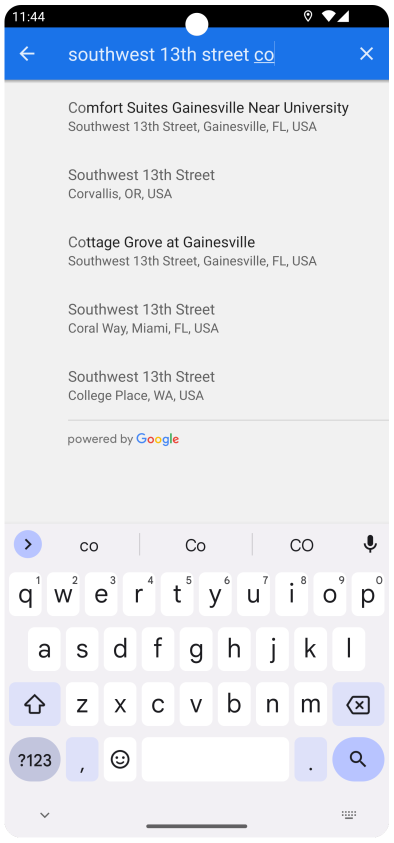


Figure 10.1: Landing page: The first screen presented to the user contains the search bar, the most suitable parking lot based on the current location, a more information button, and a navigation button



Comfort Suites Gainesville Near University
Southwest 13th Street, Gainesville, FL, USA

Southwest 13th Street
Corvallis, OR, USA

Cottage Grove at Gainesville
Southwest 13th Street, Gainesville, FL, USA

Southwest 13th Street
Coral Way, Miami, FL, USA

Southwest 13th Street
College Place, WA, USA

Figure 10.2: Search page: The user can search for a destination to find nearby parking lots. An autocomplete API provides address suggestions.

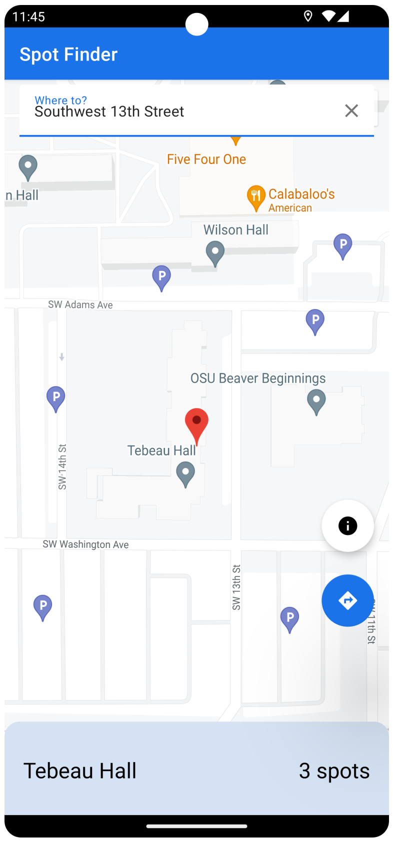
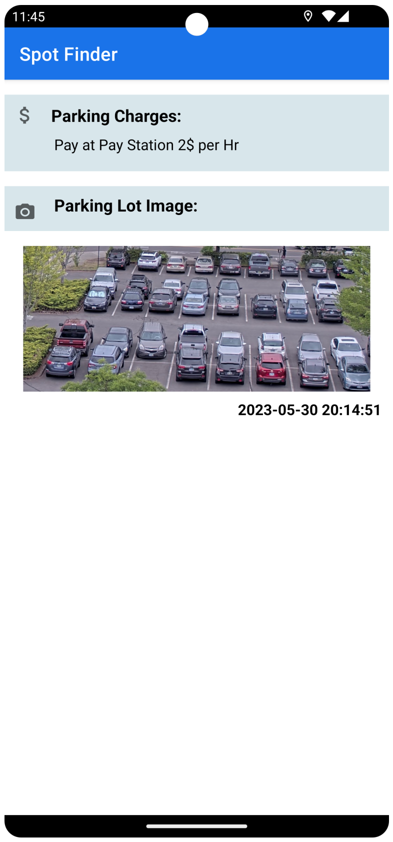


Figure 10.3: Searched result page: Most suitable parking lot is displayed based on the searched location.



Parking Charges:

Pay at Pay Station 2$ per Hr

Parking Lot Image:

2023-05-30 20:14:51

Figure 10.4: More Information Page: Parking charges and the live image is displayed for the most suitable parking lot

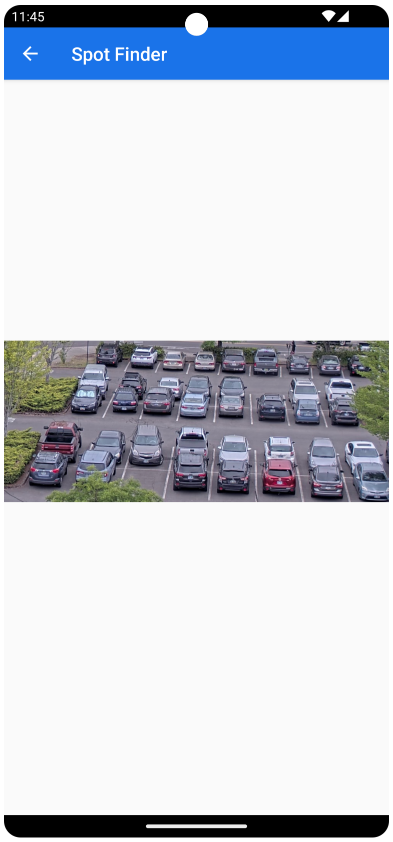
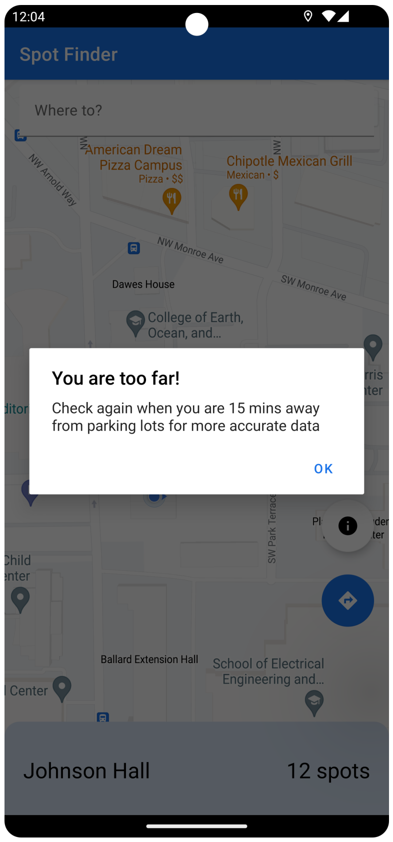


Figure 10.5: Full-screen Image View: A full-screen live image is displayed, with zoom-in and zoom-out capabilities.


12:04

Spot Finder

Where to?

You are too far!

Check again when you are 15 mins away from parking lots for more accurate data

OK

Johnson Hall 12 spots

Figure 10.6: Alert Message: When the user is far from all the parking lots, a popup is displayed

11. Future Development

11.1 Application Enhancements

OSU currently operates a virtual permit system [18] for its parking facilities, which allows users to register their license plates via an online portal. However, this system lacks real-time updates regarding parking spot availability, leading to instances where users may purchase permits but fail to find an available spot due to lot saturation. In response to this, we plan to incorporate the OSU virtual permit system into our SpotFinder application, thereby providing a more dynamic, reliable, and streamlined parking experience. This integration necessitates the addition of several key features to the application:

1. User Authentication: To ensure a seamless user interface, the introduction of ONID or guest sign-in/sign-up options for user authentication is needed. With this feature, users will stay logged in until they opt to log out manually.

2. Payment System: As we shift towards online reservations for parking spots, a secure and intuitive payment feature is vital. To minimize user interaction with the app while driving, we also propose a system to store payment data securely. This will also be integrated with orange cash for a better user experience.

3. Algorithm-Based Parking Selection: By leveraging the specialized weighted algorithm, this will recommend the most convenient parking spots to users. Upon booking, we store crucial information like the user's ID, assigned parking lot, reserved duration, and license plate number in our database.

4. SpotFinder Exclusive Spots: To encourage app usage and streamline the reservation process, we suggest reserving certain spots in the parking lot specifically for SpotFinder users. Users who don't have the app can scan a QR code to download it, sign in, and reserve the spot. This approach promotes the application and assists the university's transition towards a more digital, reliable, and efficient parking process. Gradually with more user

adoption, we can increase the footprint of reserved spots to eventually take the campus towards a 100% online model.

5. Parking Zones: We also will have a database to store the users who have purchased long term passes. Using this we can direct users to spots in which they are able to park.

To improve the accuracy of SpotFinder, we intend to corroborate data from two sources: our computer vision model and the reservation database. This comparison between the number of reserved spots and the number of spots actually occupied will help us flag potential instances of illegal parking to the campus enforcement team.

Campus enforcement vehicles are outfitted with systems that recognize license plates and check them against valid virtual permits [19] If a vehicle is found parked in an incorrect zone or if a license plate does not match a valid permit, the enforcement officer can issue a citation.

If the reservation database indicates more reserved spots than the computer vision model shows occupied, we can engage with users whose permits are about to expire. By verifying if they've vacated their spot prematurely and whether they consent to their spot being reassigned, we can maximize the usage of parking spaces. The university can provide incentives for such actions, especially during peak hours or when parking lots are fully occupied.

In summary, the comprehensive integration of SpotFinder will redefine the parking experience at OSU. By addressing the current system's shortcomings and introducing new features to streamline the reservation process, we aim to position SpotFinder as the go-to parking solution even in areas with free parking spaces that currently employ a soft reservation system.

11.2 To improve the Computer vision model accuracy

Upgrade to the Latest YOLO Version: The current model utilizes YOLO4 and we could improve accuracy by upgrading to the latest version, YOLO8 [20].

Continuous model training: Regularly feeding the model with additional and diverse data - such as images of parking lots in different lighting conditions, angles, car sizes and colors, and weather conditions - can help improve its car detection accuracy.

Optimizing Camera Positions: The accuracy of a computer vision model is heavily dependent on the quality of the images it has to analyze. We need to ensure the cameras are strategically positioned to cover all parking spots without any blind spots, with the right angles and height.

Employing Ensemble Techniques: Using multiple models and aggregating their predictions would help to better results compared to one model. This approach, known as an ensemble method, can boost the robustness and stability of the computer vision model.

12. Conclusion

In conclusion, SpotFinder has been innovatively designed to address the existing gaps in parking solutions at Oregon State University (OSU). By offering a soft reservation system, SpotFinder eases the task of parking by prioritizing spots based on proximity and availability, thus eliminating the risk of unnecessary overlap. The soft reservation system, based on well-thought-out 15-minute radius criteria, ensures an efficient allocation of parking spaces and significantly enhances the user experience on campus.

Recognizing the challenges posed by offline drivers and limitations of the current User API, I have clearly identified areas for future growth. As SpotFinder continues to evolve, enhancements such as the confirmed reservation system with integrated online payment options, and transitioning the OSU parking system to an online platform, are planned to increase its usability and reliability.

The future integration of OSU's virtual permit system into the SpotFinder application will lead to a dynamic, reliable, and streamlined parking experience. This includes introducing user authentication, a secure payment system, an algorithm-based parking selection, and SpotFinder exclusive spots. The expansion of reserved spots with increasing user adoption will eventually lead the university towards a 100% online model.

We also plan to improve SpotFinder's accuracy by corroborating data from our computer vision model and the reservation database, which will help in identifying illegal parking and ensure optimal utilization of parking spaces.

Overall, SpotFinder is not just a solution for the present, but a strategic plan for the future. It represents a long-term commitment to making parking at OSU as smooth, efficient, and user-friendly as possible. SpotFinder aims to redefine the parking experience at OSU and set a new standard for campus parking systems.

References

[1] "Oregon state university." https://en.wikipedia.org/wiki/Oregon_State_University. Online; accessed 13-August-2023.

[2] "Osu corvallis parking map." https://transportation.oregonstate.edu/sites/transportation.oregonstate.edu/files//parking-map.pdf. Online; accessed 13-August-2023.

[3] "Google maps - app on google playstore." https://play.google.com/store/apps/details?id=com.google.android.apps.maps. Online; accessed 13-August-2023.

[4] "Parkopedia - app on google play store." https://play.google.com/store/apps/details?id=com.parkopedia. Online; accessed 13-August-2023.

[5] "Smartpark project." https://research.engr.oregonstate.edu/si-lab/archive/2022_chaitanya.pdf. Online; accessed 13-August-2023.

[6] "What is weighted scoring?." https://chisellabs.com/glossary/what-is-weighted-scoring. Online; accessed 13-August-2023.

[7] "Uber - app on google play store." https://play.google.com/store/apps/details?id=com.ubercab. Online; accessed 13-August-2023.

[8] "Lyft - app on google play store." https://play.google.com/store/apps/details?id=me.lyft.android. Online; accessed 13-August-2023.

[9] "Parkwhiz - app on google play store." https://play.google.com/store/apps/details?id=com.parkwhiz.driverApp. Online; accessed 13-August-2023.

[10] "Spothero - app on google play store." https://play.google.com/store/apps/details?id=com.spothero.spothero. Online; accessed 13-August-2023.

[11] "Mongodb." https://www.mongodb.com/atlas/database. Online; accessed 13-August-2023.

[12] "Introduction to mvvm on android." https://resocoder.com/2018/08/31/introduction-to-mvvm-on-android/. Online; accessed 13-August-2023.

[13] “Android – build a movie app using retrofit and mvvm architecture with kotlin.” https://www.geeksforgeeks.org/android-build-a-movie-app-using-retrofit-and-mvvm-architecture-with-kotlin/#. Online; accessed 13-August-2023.

[14] “Build mvvm application with kotlin—part1getting started.” https://medium.com/android-news/build-mvvm-application-with-kotlin-part1-getting-started-972852e61193. Online; accessed 13-August-2023.

[15] “Livedata.” https://developer.android.com/reference/android/arch/lifecycle/LiveData. Online; accessed 13-August-2023.

[16] “Swagger.” https://www.techtarget.com/searchapparchitecture/definition/Swagger. Online; accessed 13-August-2023.

[17] “Understanding mockk kotlin.” https://sharmaricha7724.medium.com/understanding-mockk-kotlin-86db80db07e6. [Online; accessed 13-August-2023].

[18] “Osu - virtual permits.” https://transportation.oregonstate.edu/parking/parking-rates/virtual-permits. Online; accessed 13-August-2023.

[19] “Osu - virtual permits faq.” https://transportation.oregonstate.edu/parking/parking-rates/virtual-permits. Online; accessed 13-August-2023.

[20] “From yolo to yolov8: Tracing the evolution of object detection algorithms.” https://medium.com/nerd-for-tech/from-yolo-to-yolov8-tracing-the-evolution-of-object-detection-algorithms-eaed9a982e [Online; accessed 13-August-2023].