# QuickRide – A Serverless Mobile App

Sai Palutla

A Project Submitted to

GRAND VALLEY STATE UNIVERSITY

In

Partial Fulfillment of the Requirements

For the Degree of

Master of Science in Applied Computer Science

School of Computing and Information Systems

August 2023


The signatures of the individuals below indicate that they have read and approved the project of Naga Sai Pramod Palutla in partial fulfillment of the requirements for the degree of Master of Science in Applied Computer Science.

_____

Robert Adams, Project Advisor Date

_____

Robert Adams, Graduate Program Director Date

_____

Paul Leidig, Unit head Date

# Abstract

The aim of this project is to develop a serverless ride hailing mobile application catering to the iOS platform. The application facilitates user registration as either a driver or a passenger. Passengers can solicit rides from drivers by specifying their desired destination location. The app dynamically computes the cost of the ride based on factors such as distance and chosen ride category. Consequently, drivers are notified of ride requests generated by passengers and can either accept or decline such requests.

This application was developed using the MVVM architectural pattern and SwiftUI framework, with Firebase Firestore and authentication as the backend infrastructure. The primary focus of the app is on providing ride booking capabilities.

The app follows the Model-View-ViewModel (MVVM) architectural pattern, ensuring a clear separation of concerns and promoting testability and maintainability. SwiftUI, the declarative user interface framework from Apple, is leveraged for building the app's intuitive and interactive user interface, enabling smooth navigation and engaging user experiences.

Firebase Firestore is employed as the backend database, facilitating efficient data storage and retrieval for user information, ride details, and booking records. Firebase authentication is utilized for user authentication and ensuring secure access to the app's features and functionalities. Users can easily browse available ride options, specify their pickup and drop-off

locations, and select preferred ride types. The app provides real-time fare estimation for a convenient and hassle-free booking process.

Overall, this ride-hailing app showcases the power of MVVM architecture, SwiftUI, and Firebase Firestore and authentication as backend technologies. It delivers a robust and user-friendly solution for booking rides, emphasizing simplicity, efficiency, and security.

# Introduction

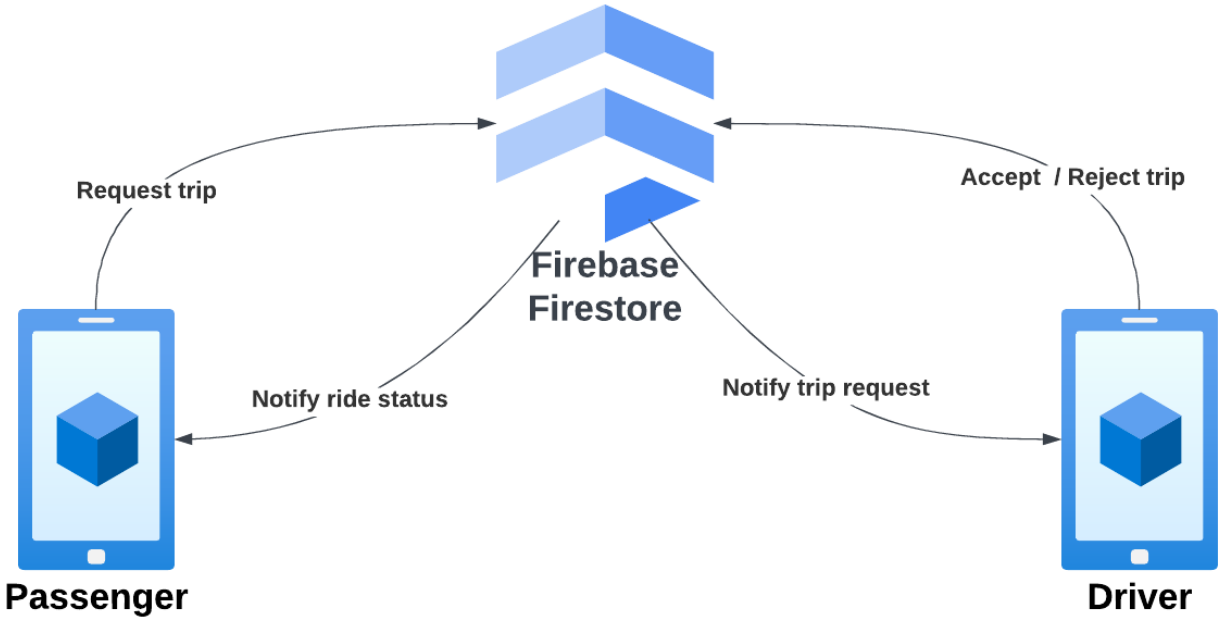
The application is designed to provide seamless user registration, enabling users to choose between becoming drivers or passengers. Passengers can effortlessly request rides from available drivers, specifying their desired destination locations. The app's dynamic trip price calculation algorithm ensures fair and transparent fare computation, considering factors such as distance and ride category. Drivers, on the other hand, receive prompt notifications of ride requests and have the flexibility to accept or decline them.

The foundation of the application's development lies in adopting the Model-View-ViewModel (MVVM) architectural pattern in conjunction with Apple's SwiftUI framework. By embracing MVVM, the app ensures a well-structured and maintainable codebase, facilitating efficient testing and reducing complexities associated with scalability. SwiftUI, as a declarative user interface framework, empowers the app to deliver an intuitive and interactive user experience, emphasizing ease of navigation and engaging visuals.

To support the app's functionality, Firebase Firestore is employed as the backend infrastructure, serving as a reliable database for storing and retrieving user information, ride details, and booking records. Furthermore, Firebase authentication ensures the security of the app, safeguarding access to its features and functionalities. As a result, users can confidently browse available ride options, specify pickup and drop-off locations, and select their preferred ride types. Real-time fare estimation adds a layer of convenience, streamlining the ride booking process for an enhanced user experience.

This project presents an in-depth exploration of a serverless application's development, highlighting the significance of MVVM architecture, SwiftUI, and Firebase Firestore and authentication as key technologies contributing to its robustness and user-friendliness.

## High level architecture




In the interactions between the passenger app, driver app, and Firebase, a seamless and efficient ride-hailing experience is facilitated while ensuring real-time data synchronization. The passenger app acts as the primary interface for users to request rides, while the driver app allows drivers to receive and respond to these ride requests. Firebase serves as the backend infrastructure, managing data storage, retrieval, and authentication.

When a passenger requests a ride through the app, the relevant ride details, such as pickup and drop-off locations, are sent to Firebase for storage. Firebase handles user authentication to ensure secure access and authorization for ride booking functionalities. Once the ride request is made, Firebase notifies nearby available drivers, enabling them to view and respond to the ride request.

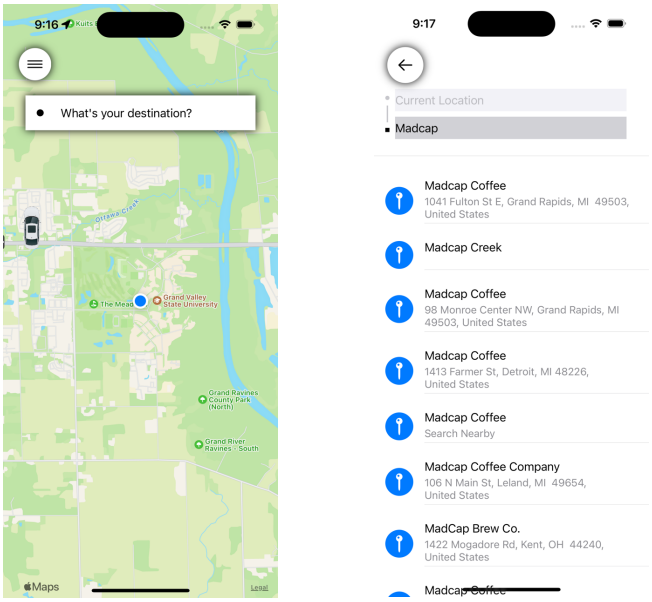
The driver app, connected to Firebase, receives these ride requests, and displays essential ride details like the passenger's pickup location. The driver can then decide to accept or decline the request. Upon acceptance, the app updates Firebase with the driver's response, allowing the passenger app to receive real-time updates on the status of their ride.

During the trip booking process, Firebase facilitates dynamic trip cost calculation and estimated time of arrival (ETA) computation based on factors such as distance and ride category. This ensures a fair and transparent fare calculation for both passengers and drivers.

In summary, the interactions between the passenger app, driver app, and Firebase create a well-coordinated system that efficiently manages ride requests, dynamic pricing, and real-time data synchronization. This approach optimizes the user experience while maintaining data security and privacy.

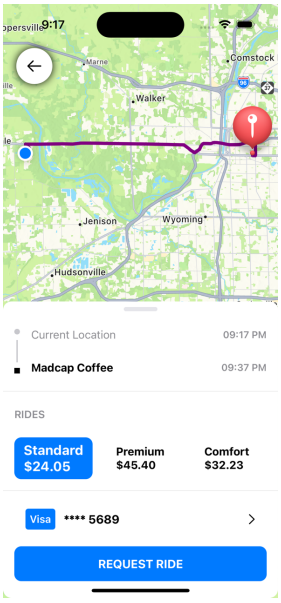
## SwiftUI

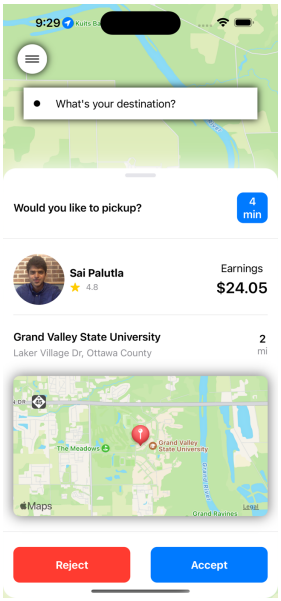
SwiftUI plays a pivotal role in the development of the ride-hailing app, offering a powerful and intuitive user interface framework for the iOS platform. With SwiftUI, developers can create a visually engaging and interactive app, enhancing the overall user experience. The declarative syntax simplifies the UI design process, enabling seamless navigation and smooth animations. SwiftUI's built-in components, such as buttons, lists, and forms, expedite the app's development and reduce boilerplate code. Moreover, SwiftUI's native support for dark mode and accessibility features ensures the app's adaptability and inclusivity. Leveraging SwiftUI empowers the app to deliver a modern and user-friendly interface, aligning with Apple's design guidelines.




An authenticated user is shown a landing screen with a map with their current location on the map. If the user is a passenger, they are shown locations of other drivers on the map. Apple Maps SDK is utilized for showing the map on the home screen.

The location search utilizes the places API of the Maps SDK for location search. User gets back a list of relevant locations based on their search criteria and can then choose the destination location for the trip.

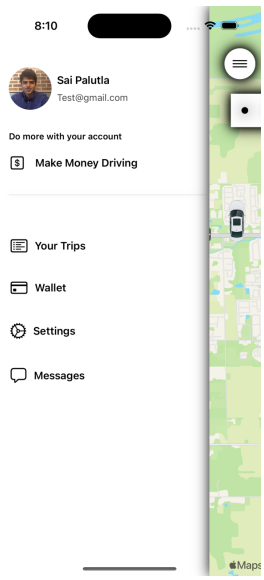


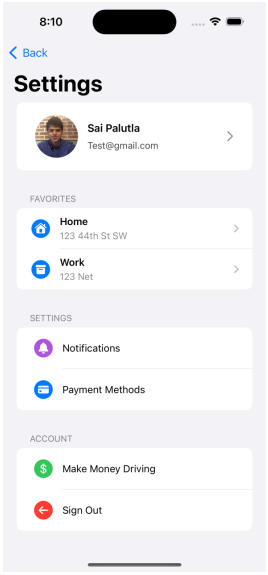



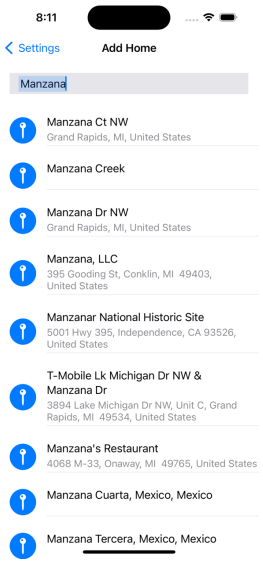

When the user selects a destination the map view updates to show the dynamic price calculated for each ride type based on the distance. ETA is also shown from the pickup location to the destination location. The polyline to the destination is drawn using the Maps SDK methods to give the user a rough idea about the route. The user can request the ride and the trip is saved with all the required information in the database.

The app on the driver's side gets the uploaded trip information with the pickup and drop-off locations. The app actively listens for new trips uploaded to the database with the driver's userID.

If the trip's driverID matches the drivers userID, the app shows a trip confirmation view for the driver. They can accept or decline the request. The passenger then gets notified of the trip status.



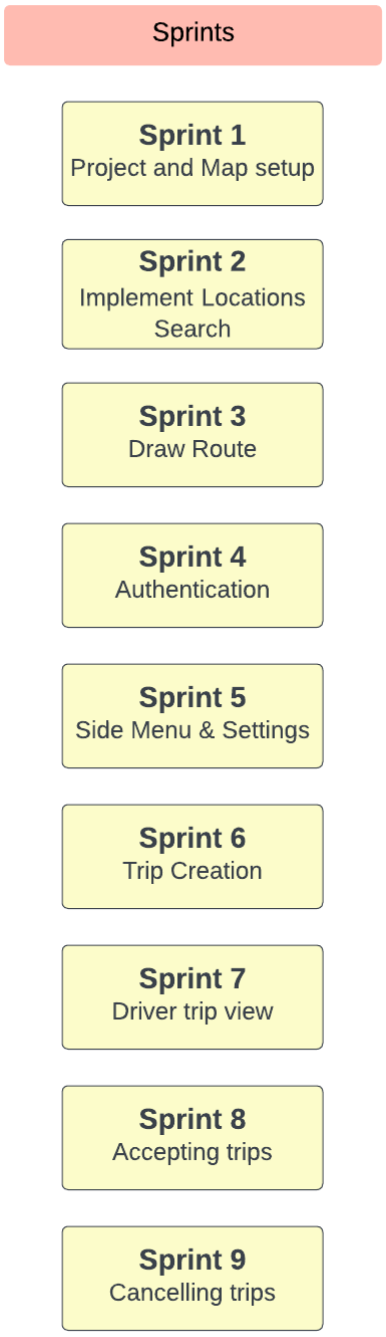




The app incorporates a side menu navigation system which is seamlessly integrated into the app's user interface, providing an intuitive and unobtrusive way to access additional functionalities. Notable functionality offered in the settings is the ability to save frequently used locations. Users can streamline their ride booking process without entering these addresses every time.

Users can access their ride history to review previous bookings, view and manage their payment methods for secure and effortless transactions. There is also an option to sign up for a driver account, which converts a passenger account to driver so that users can start earning.

# Project Management

The development of the application was divided into 9 sprints of 1 – 2 weeks each.




**Sprint 1**: Project and Map setup - Initiated project setup and integrated map functionality for location visualization.

**Sprint 2**: Locations Search - Implemented search functionality for finding specific destinations on the map.

**Sprint 3**: Adding locations and polylines to the Map - Enabled users to add locations and displayed routes using polylines.

**Sprint 4**: User authentication and requesting Rides - Incorporated user authentication and enabled ride requests by passengers.

**Sprint 5**: Side Menu and Settings Page - Developed a side menu and settings page for enhanced user navigation and customization.

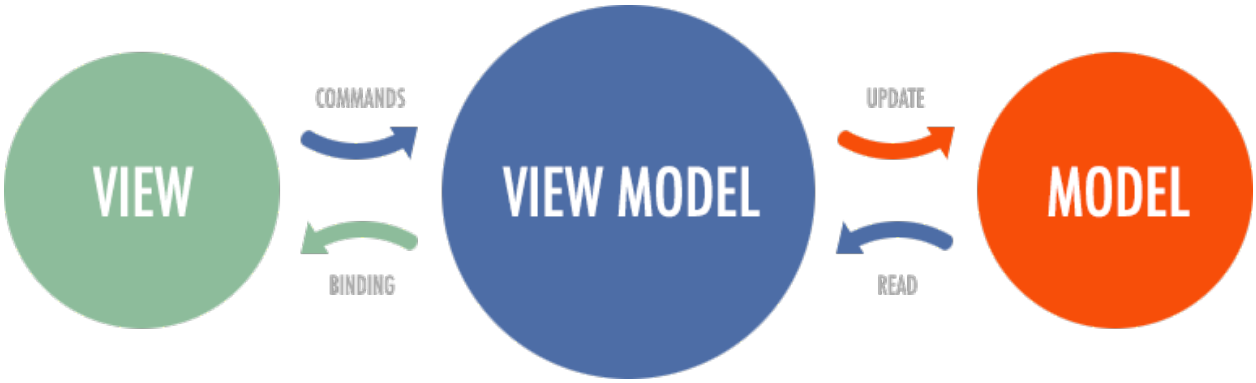
**Sprint 6**: Driver signup and Trip creation - Enabled driver signup and facilitated trip creation functionality for drivers.

**Sprint 7**: Driver Trip View - Implemented a dedicated view for drivers to manage their ongoing and upcoming trips.

**Sprint 8**: Accepting trips - Developed the mechanism for drivers to accept ride requests from passengers.

**Sprint 9**: Cancelling trips - Enabled users to cancel trips with appropriate handling of trip statuses and notifications.

## Organization




**MVVM Architecture**

MVVM (Model-View-ViewModel) is an architectural pattern for app development, promoting separation of concerns. Models represent data and logic, Views render the UI, and ViewModels mediate between them. This decoupling enhances code maintainability, scalability, and testability. Data binding ensures automatic UI updates when ViewModel data changes, streamlining development and responsiveness. MVVM's modularity fosters independent work on different parts of the app and facilitates code reusability. Adding new features is simplified as ViewModel logic is independent of the UI, reducing disruptions to the existing codebase. This pattern's clear division of responsibilities allows for easier integration of new functionality without extensive UI changes, resulting in a more efficient and scalable app development process. MVVM's loose coupling enables easy swapping of mapping services (e.g., Apple Maps to Google Maps) without major modifications.

## Reflection and Conclusion

The development of a serverless ride-hailing mobile application catering to the iOS platform has been a rewarding and insightful experience. By adopting the MVVM architectural pattern and leveraging SwiftUI and Firebase Firestore, we achieved a well-organized and user-friendly app, ensuring clear separation of concerns and promoting code maintainability and testability. The app's focus on ride booking capabilities has delivered an intuitive and engaging user interface, enhancing the overall user experience. The utilization of Firebase Firestore and authentication as the backend infrastructure allowed for efficient data storage, retrieval, and secure user authentication.

In conclusion, the development of this serverless ride-hailing app highlights the success of implementing MVVM, SwiftUI, and Firebase Firestore and authentication as backend technologies. The app effectively delivers ride booking services to users, providing seamless registration, real-time fare estimation, and dynamic trip price calculation. The MVVM architecture's modular approach and SwiftUI's declarative framework have significantly contributed to the app's robustness and user-friendliness.

## Future Work

Looking ahead, several enhancements can further elevate the app's functionality. One notable improvement would be the incorporation of live tracking, enabling passengers to monitor their driver's real-time location during the ride. This addition would enhance transparency and build trust between users and drivers. Additionally, the integration of payment gateways for cashless transactions and in-app messaging for improved communication between passengers and drivers would add convenience and efficiency to the app. Moreover, exploring multilingual support and accessibility features would extend the app's inclusivity and cater to a broader audience.