**Logo

Description automatically generated LEARN DRIVE**

**DHARMA TEJA DONEPUDI**

**VAMSI KRISHNA ANANTHANENI**

This project is submitted to the Gannon University graduate faculty in

partial fulfillment for the degree Master of Sence in Computer and Information Science.

**Approved By:**

**Dr. Sheheeda Marian Manakkadu Dr. Joshua Nwokeji**

**Advisor Graduate Program Director**

**Gannon University, Erie, Pennsylvania 16541**

**MAY 2023**

**INDEX**

|  |  |
| --- | --- |
| **S.No** | **Contents** |
| **1.** | Abstract |
| **2.** | Scope & Background |
| **3.** | Introduction |
| **4.** | Problem Statement & Existing Systems |
| **5.** | Research Questions |
| **6.** | Methodology |
| **7.** | Hardware/Software tools used |
| **8.** | UML Diagrams, Use Cases & Architecture Diagrams |
| **9.** | Code & Results |
| **10.** | Conclusion |

**ABSTRACT**

In today's fast-paced world, mobile applications have become an essential part of our daily routine. They provide us with convenience and accessibility, making our lives easier. Designing a mobile iOS application is an excellent way to reach multiple users, and if done correctly, it can be a powerful tool for growth. The key to designing a successful mobile application is to make it user-friendly and reliable. One area where mobile applications have made a significant impact is in the field of transportation. Booking a ride through a mobile application has become a preferred mode of transport for many people. It's quick, convenient, and affordable. With the help of a mobile application, booking and learning through an existing driver in a few hours has become much more accessible than spending days in a driving school.

The primary advantage of a mobile application for booking rides is ease of use. When a user sets up a time and pick-up location, the app instantly shows available drivers and their availability to match the user's requirements. This feature saves a considerable amount of time, as users don't have to wait for a long time to book a ride. It also reduces the chances of miscommunication between the user and the driver, as everything is clearly defined within the app. Another advantage of a mobile application is the payment plan and methods. This feature is essential as it provides users with transparency and security. The payment plan and methods are processed on the next page of the booking. Users can choose from various payment options like debit/credit cards, digital wallets, or even cash. This feature provides users with flexibility, making the booking process hassle-free.

The mobile application's design plays a crucial role in its success. The interface should be simple and easy to use. It should also be visually appealing, with bright colors and clear graphics. The app should be optimized for different screen sizes and have a fast response time. User feedback is an essential aspect of any mobile application. It helps the developers understand the users' needs and make necessary improvements to the app. In-app feedback forms are an excellent way to gather feedback from users. The feedback form should be easy to access and should ask relevant questions. The feedback should be taken seriously, and necessary improvements should be made promptly. Security is a crucial aspect of any mobile application. The app should be designed to protect user’s sensitive information, like their personal and financial details. The app should have encryption protocols in place to prevent data breaches. The developers should also provide regular updates to ensure the app's security.

Marketing is another crucial aspect of a successful mobile application. A well-planned marketing strategy can help the app reach a wider audience. The developers should identify the target audience and tailor the marketing strategy accordingly. Social media is an excellent platform for promoting mobile applications. It provides developers with an opportunity to engage with potential users and gather feedback. Designing a mobile iOS application for booking rides is an excellent way to reach multiple users. The app should be user-friendly and reliable, with a simple interface and fast response time. The payment plan and methods should be transparent and secure. User feedback should be taken seriously, and necessary improvements should be made promptly. Security should be a top priority, with encryption protocols in place to protect users' sensitive information. A well-planned marketing strategy can help the app reach a wider audience. With the right approach, designing a mobile application can be a powerful tool for growth.

Apart from booking and learning through existing drivers, a mobile application can also provide users with additional features like tracking the driver's location, estimated time of arrival, and fare estimates. These features provide users with more transparency, making the booking process much more comfortable and less stressful. Mobile applications can also provide users with loyalty programs and rewards, encouraging them to use the app regularly. Loyalty programs can include discounts, cashback offers, and free rides. These programs can help the app gain a loyal user base, leading to higher growth rates. Another crucial aspect of a successful mobile application is customer support. Users may face issues while using the app or may have queries regarding the app's features. A reliable customer support team can help resolve these issues promptly. The customer support team should be accessible through multiple channels like chat, email, or phone. Data analytics is another essential aspect of a mobile application. The app should be designed to collect user data, like booking patterns, frequently visited locations, and preferred payment methods. This data can be used to provide users with personalized offers and discounts. It can also be used to make necessary improvements to the app, based on user behavior and preferences.

Mobile applications can also be designed for different types of transportation, like cars, bikes, or scooters. This feature can provide users with more flexibility and variety, making the booking process more comfortable and more convenient. One potential challenge while designing a mobile application for booking rides is competition. There are numerous ride-hailing applications available in the market, and competition can be fierce. To stand out in the market, the app should have unique features and provide users with a seamless booking experience. A well-planned marketing strategy can also help the app gain a competitive edge. Hence, a mobile iOS application for booking rides can be a powerful tool for growth. The app should be user-friendly, reliable, and secure. It should have a simple interface, fast response time, and transparent payment plans. The app should also provide users with additional features like tracking the driver's location and fare estimates. Loyalty programs, customer support, data analytics, and personalized offers can also help the app gain a loyal user base. Competing with other ride-hailing applications can be a challenge, but a well-planned marketing strategy and unique features can help the app stand out. Overall, designing a mobile application for booking rides requires careful planning, research, and implementation to ensure its success.

**SCOPE**

The scope of the project for designing a mobile iOS application for booking rides is extensive, with a range of features and functionalities that can be implemented to enhance the user experience and promote growth. The following are some of the potential areas of scope for the project:

**User Registration:** The first step in the application is to allow users to register and create an account. The registration process can include basic information like name, phone number, email address, and payment details. The app can also provide the option to register using social media accounts like Facebook or Google.

**Ride Booking:** The primary feature of the application is to allow users to book rides from their location to their desired destination. The user should be able to select the type of ride they need, like a car, bike, or scooter, and set the pickup and drop-off locations. The app should show available drivers in the area and provide information like the driver's rating, estimated time of arrival, and fare estimates.

**Driver Tracking:** The application should allow users to track the driver's location in real time once the booking is confirmed. The user should be able to see the driver's location on a map and get updates on their estimated time of arrival.

**Payment Gateway:** The app should provide a secure and user-friendly payment gateway to enable users to make payments for their rides. The app should support multiple payment methods like credit/debit cards, net banking, or mobile wallets. The payment gateway should be integrated with the app to ensure a seamless payment experience for the users.

**Loyalty Programs:** The application can provide loyalty programs and rewards for regular users. This can include discounts, cashback offers, or free rides. The app can track user behavior and provide personalized offers based on their preferences and booking patterns.

**Customer Support:** The app should have a reliable customer support team accessible through multiple channels like chat, email, or phone. The team should be trained to resolve issues promptly and provide quality service to the users.

**Data Analytics**: The app should be designed to collect user data, like booking patterns, frequently visited locations, and preferred payment methods. This data can be used to provide users with personalized offers and discounts. It can also be used to make necessary improvements to the app, based on user behavior and preferences.

**Security and Privacy:** The app should be designed to ensure the safety and privacy of user data. The app should have a robust security system to prevent unauthorized access and ensure user data is stored securely.

**Marketing and Promotion:** A well-planned marketing strategy can help the app gain a competitive edge in the market. This can include digital marketing, social media marketing, influencer marketing, and other promotional activities to reach a broader audience.

**Expansion and Scaling:** Once the app gains traction in the market, there can be potential opportunities for expansion and scaling. The app can be expanded to other cities, and countries, or even to other modes of transportation like buses or trains.

The scope of the project for designing a mobile iOS application for booking rides is vast, with numerous features and functionalities that can enhance the user experience and promote growth. The success of the project will depend on careful planning, research, and implementation of these features to create a user-friendly, reliable, and secure app that can stand out in the competitive market. With a well-planned marketing strategy and a strong focus on user experience, the app can gain a loyal user base and become a significant player in the ride-hailing industry.

**INTRODUCTION**

Driving is an essential skill that has become increasingly important in today's world. It not only provides independence and freedom of mobility but is also an essential requirement for many jobs and career opportunities. However, learning to drive can be a challenging and expensive task, which often deters many people from obtaining their driver's license. In many cases, individuals either have to pay a considerable amount of money to attend driving schools or rely on friends and family to teach them how to drive. Moreover, attending a driving school can be time-consuming, and transportation to and from the school can be expensive.

To overcome these challenges and make learning to drive more accessible and convenient, we propose to develop a mobile application that connects learners with experienced drivers who are willing to teach. Our application aims to provide a low-cost solution for individuals who want to learn how to drive while also enabling experienced drivers to earn extra income by teaching. This is a unique application, and there is no existing mobile application that offers such services.

The primary objective of our mobile application is to provide a safe, convenient, and affordable solution for individuals who want to learn how to drive. The application will enable learners to connect with experienced drivers who are available to teach at their convenience. Learners will be able to browse through a list of available drivers and select the one that best fits their needs. The drivers will be able to provide driving lessons to the learners, focusing on the necessary skills and techniques needed to drive safely and confidently.

The application will also offer features that will make the learning process more convenient and user-friendly. For instance, learners will be able to schedule their driving lessons at their preferred time and location. The application will also provide real-time tracking of the learners' progress, enabling them to monitor their driving skills and identify areas that need improvement. In addition to providing a solution for learners, our mobile application will also benefit experienced drivers who are willing to teach. The application will offer a platform for these drivers to connect with potential learners and earn extra income by providing driving lessons. Drivers who are already working for ride-hailing services like Uber and Door Dash can leverage their driving experience and use our application to offer driving lessons in their free time.

Our mobile application will also be beneficial to individuals who are new to a country and want to learn the traffic rules and regulations. These individuals can connect with native drivers who can help them understand the local traffic laws and customs. The application will provide an opportunity for native drivers to earn extra income while helping newcomers adjust to their new environment. The development of our mobile application will require significant effort and resources. The application will require a team of experienced developers, designers, and testers who can work together to ensure the application's success. We will also need to engage in market research to identify potential users and understand their needs and preferences. We will also need to develop a marketing strategy that can help us reach a wider audience and promote the application.

So, the development of a mobile application that connects learners with experienced drivers who are willing to teach can be a significant breakthrough in the field of driver education. Our application aims to provide a low-cost solution that is convenient, user-friendly, and accessible to everyone. With the right team and resources, we are confident that we can develop an application that can make a significant impact on individuals' lives who want to learn how to drive. Learning how to drive is an essential skill in today's world, but it can be a daunting and expensive task. Traditionally, individuals have had to either pay for expensive driving lessons or rely on family and friends to teach them. However, a new mobile application can now make the learning process more accessible, convenient, and cost-effective.

Our mobile application connects learners with experienced drivers who are willing to teach, providing a low-cost solution that is convenient and user-friendly. The application offers several benefits, including:

1. **Cost-effective:** Attending a driving school can be costly, with fees ranging from hundreds to thousands of dollars. Our application provides a more affordable alternative that enables learners to take driving lessons at a fraction of the cost.
2. **Convenience:** Our application allows learners to schedule their driving lessons at their preferred time and location, eliminating the need for costly transportation to and from a driving school.
3. **User-friendly:** Our application provides an easy-to-use platform that enables learners to browse through a list of available drivers and select the one that best fits their needs. The application also provides real-time tracking of the learners' progress, making it easy for them to monitor their driving skills and identify areas that need improvement.
4. **Accessible:** Our application is accessible to anyone with a smartphone, making it easy for individuals in remote areas or those without easy access to driving schools to learn how to drive.
5. **Increased safety:** By providing learners with access to experienced drivers, our application helps to increase safety on the roads. Learners can benefit from the expertise of experienced drivers who can provide them with essential driving skills and techniques to drive safely and confidently.
6. **Additional income:** Experienced drivers who are willing to teach can use our application to earn extra income by providing driving lessons in their free time. This provides a new revenue stream for ride-hailing drivers who may be looking for additional ways to supplement their income.
7. **Social impact:** Our application has the potential to make a significant social impact by enabling individuals who may have previously been unable to afford driving lessons to learn how to drive. This can have a significant impact on their ability to find employment, access healthcare, and participate in other important activities.

In addition to these benefits, our mobile application can also be beneficial for individuals who are new to a country and want to learn the traffic rules and regulations. These individuals can connect with native drivers who can help them understand the local traffic laws and customs. The development of our mobile application will require significant effort and resources. We will need a team of experienced developers, designers, and testers who can work together to ensure the application's success. We will also need to engage in market research to identify potential users and understand their needs and preferences. Developing a marketing strategy will be critical to reaching a wide audience and promoting the application.

Hence, our mobile application has the potential to revolutionize the field of driver education by providing a low-cost, convenient, and user-friendly solution for individuals who want to learn how to drive. The application offers several benefits, including cost-effectiveness, convenience, user-friendliness, accessibility, increased safety, additional income, and social impact. By providing learners with access to experienced drivers, our application can help to increase safety on the roads and empower individuals to participate more fully in their communities.

**BACKGROUND**

The driving education industry has been predominantly dominated by traditional driving schools, which are often expensive and may not be accessible to everyone. In addition, traditional driving schools may not be flexible in terms of scheduling, location, and course duration, which can be a major inconvenience for learners. As a result, there is a growing demand for a more flexible, affordable, and accessible driving education system. Mobile applications offer an excellent platform to meet these demands. In today's fast-paced world, it has become essential to have access to reliable, affordable, and convenient transportation. Driving a car is one of the most preferred modes of transportation as it provides freedom, flexibility, and convenience. However, learning to drive can be a challenging task, especially for those who do not have access to a car or a licensed driver to teach them.

Traditionally, the most common way to learn to drive is by enrolling in a driving school or hiring a personal instructor. However, these options can be expensive and time-consuming. Additionally, many people may not have the luxury of taking time off work or school to attend driving classes. To address these challenges, online driving platforms and mobile driving applications have emerged as innovative and convenient solutions. These platforms offer learners the flexibility to learn at their own pace, schedule, and budget. They also provide licensed drivers with an opportunity to earn extra income by sharing their skills and knowledge.

The need for a driving education application has been fueled by the fact that many people require the flexibility to learn to drive on their own schedule and at their own pace. Traditional driving schools, while effective, often come with significant costs and inflexible schedules that do not cater to everyone. There have been previous attempts at creating similar applications, such as the “Driving School 2016” app, which teaches users how to drive different vehicles, including buses, trucks, and cars. However, this app is more of a simulation game rather than an actual driving education tool, and it lacks the ability to connect learners with qualified driving instructors.

Other online driving schools like Driving-Tests.org provide free practice tests, driver's education courses, and licensing guides for all 50 states. These online resources, while informative, do not provide hands-on training or connect learners with licensed instructors. As previously mentioned, there are also ride-sharing apps like Uber and Lyft that have already disrupted the traditional taxi industry. These apps could serve as a potential model for a driving education app, as they already have a well-established platform for matching drivers with riders based on location and availability.

Another potential source of inspiration for the project could be the various language learning apps like Duolingo, which uses a gamified approach to make learning a new language more engaging and fun. The driving education app could use a similar approach to make learning to drive more accessible and enjoyable for users. In summary, the need for a driving education app that offers flexible schedules, affordable pricing, and easy access to qualified instructors is apparent. There are currently few existing solutions that meet these criteria, which presents an opportunity for the development of a driving education app that meets the needs of learners and instructors alike.

**METHODOLOGY**

**Technical Requirements -**

**X-Code -** Xcode is an integrated development environment (IDE) for creating software applications that run on Apple's macOS and iOS operating systems. It includes a suite of software development tools that allow developers to design, code, test, and debug their applications all in one place.

Xcode includes a number of features and tools that make it easy to create apps for Apple devices. One of its key features is its user interface builder, which allows developers to create graphical user interfaces (GUIs) for their apps using a drag-and-drop interface. This makes it easy for developers to design their app's user interface without having to write any code. Another key feature of Xcode is its code editor, which includes features such as syntax highlighting, autocompletion, and code folding. This makes it easy for developers to write clean, readable code that is easy to understand and maintain. There are different types of services that X code provides -

**Project Navigator -** The project navigator shows all of the files and resources in your project. You can use it to navigate between files, create new files, and organize your project.

**Interface Builder** - The interface builder allows you to design the user interface of your app using a drag-and-drop interface. You can add buttons, labels, text fields, and other user interface elements to your app using this tool.

**Code Editor & Simulator** - The code editor is where you write the code for your app. It includes features such as syntax highlighting, autocompletion, and code folding to make it easier to write clean, readable code. The simulator allows you to test your app on different devices without having to physically own those devices. You can test your app's user interface and functionality using this tool.

**Swift Programming -**

Swift is a programming language developed by Apple Inc. for developing applications for iOS, macOS, watch OS, and type OS platforms. It was first introduced in 2014, and it has rapidly gained popularity among developers due to its modern syntax and robust features. It is designed to be easy to learn and use, with a clean and concise syntax that makes code easy to read and write. It is a compiled language, which means that code is translated into machine code before it is run, resulting in faster performance. Some of the key features of Swift include optional, which helps developers handle the absence of a value, and closures, which allow for code to be passed around and executed at a later time. Swift also has a powerful type of system, which helps developers catch errors at compile time.

Swift is an open-source language, which means that it is available for anyone to use and contribute to. This has led to a thriving community of developers who share their code and provide support to one another. It can be used with Apple's integrated development environment (IDE), Xcode, which provides tools for building, debugging, and testing applications. Xcode includes a visual editor for designing user interfaces, as well as a source code editor for writing code. Overall, Swift is a modern and powerful programming language that is well-suited for developing applications for Apple's platforms. Its ease of use and robust features make it a popular choice among developers, and its open-source nature ensures that it will continue to evolve and improve over time.

**Use Cases of Swift Programming -**

iOS and macOS app development: Swift is the primary programming language used for developing iOS and macOS applications. With its clean syntax, type safety, and high performance, Swift provides a great development experience for building native apps for these platforms.

**Server-side programming:** Swift can also be used for server-side programming. Swift on the server is particularly useful when you need to build scalable, fast, and secure web applications.

**Augmented Reality:** Swift programming language is widely used in augmented reality (AR) development. ARKit, Apple's framework for building AR experiences on iOS devices, is primarily built with Swift.

**Machine learning:** Swift is also used in machine learning applications. Apple has developed a powerful framework called Core ML that allows developers to build machine learning models that can be deployed on iOS and macOS platforms.

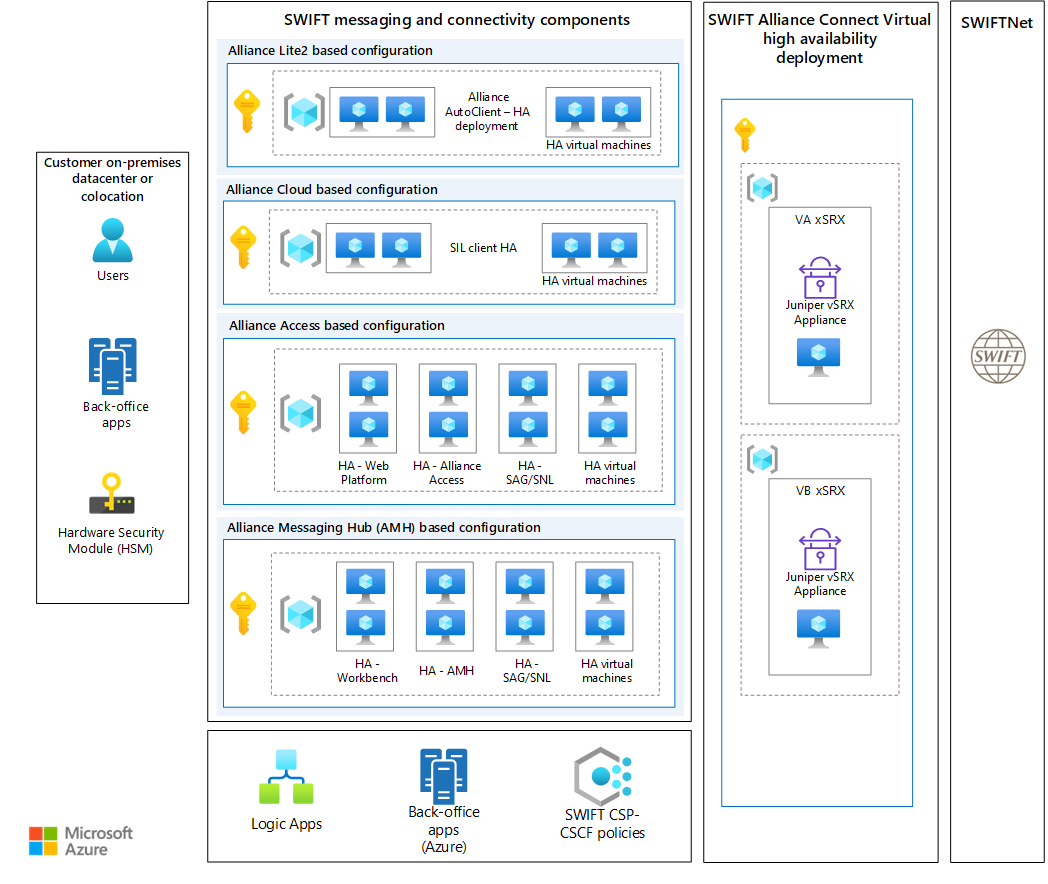
**Wearable technology:** Swift is also used for building apps for wearable technology such as Apple Watch. With its high performance and low resource usage, Swift provides a great development experience for building apps for wearable devices.

**Existing System**

The existing system for a driving education app could be developed using various programming languages and frameworks, including Swift, Java, or React Native. However, Swift is a popular programming language for developing iOS apps, and it offers a range of advantages for building a driving education app.

There are few applications that support the same methodology. But those applications provide only driving less and users should use other applications for riding. Also, these applications are based on high-level methodologies that are very difficult to deploy.

Also, the major disadvantages of existing systems are -



**High cost:** Traditional driving schools can be expensive, especially if a learner needs multiple lessons or extra practice time.

**Limited availability:** Driving schools may not be available in all areas, or may have limited hours of operation, which can make it difficult for some learners to access them.

**Limited customization:** Traditional driving schools often have a fixed curriculum and lesson plan, which may not be customizable to meet the specific needs of each learner.

**Limited feedback:** In a traditional driving school setting, learners may not receive detailed feedback or have the opportunity to review their progress in real-time.

**Limited flexibility:** Traditional driving schools may have rigid schedules and limited availability for lesson rescheduling, which can make it difficult for learners to accommodate other commitments.

**Potentially stressful:** Some learners may feel nervous or intimidated by the high-pressure environment of a traditional driving school, which can negatively affect their ability to learn and retain information.

**Limited access to resources:** Traditional driving schools may have limited access to technology or advanced resources, which can make it difficult to provide learners with the latest and most effective learning tools.

Overall, traditional driving education systems can be effective for some learners, but they may not be accessible, affordable, or customizable enough to meet the needs of all learners. A driving education app could potentially address some of these limitations by providing learners with more flexibility, customization, and access to resources. However, it is important to consider the potential limitations and disadvantages of any technology-based system as well, such as potential technical issues, reliance on internet connectivity, and the need for a certain level of digital literacy.

**RESEARCH QUESTIONS**

1. How can the mobile app be designed to provide a user-friendly and reliable experience for learners and drivers?
2. What features should be included in the mobile app to provide learners with a customized and effective learning experience?
3. How can the mobile app be designed to address the potential limitations and disadvantages of traditional driving education systems?
4. What factors influence learner satisfaction and success with the mobile app?

**User Interface Design:** Use Swift code to design and implement a user interface that is intuitive, easy to use, and visually appealing for both learners and drivers.

**Customization and Personalization:** Use Swift code to develop a system that allows learners to set their own goals, track their progress, and receive personalized feedback and recommendations.

**Machine Learning and Data Analytics:** Use Swift code to develop algorithms that can analyze data from learner interactions with the app to identify patterns, trends, and areas for improvement.

**Gamification:** Use Swift code to create gamified elements within the app, such as rewards and achievements, to motivate and engage learners.

**Payment and Transaction Processing:** Use Swift code to implement a secure and reliable system for processing payments and transactions within the app.

**Social Networking and Community Building:** Use Swift code to develop social networking and community-building features within the app, such as chat rooms and forums, to enable learners to connect with each other and share resources and advice.

**Location-Based Services:** Use Swift code to implement location-based services within the app, such as GPS tracking and mapping, to help learners find nearby driving instructors and practice locations.

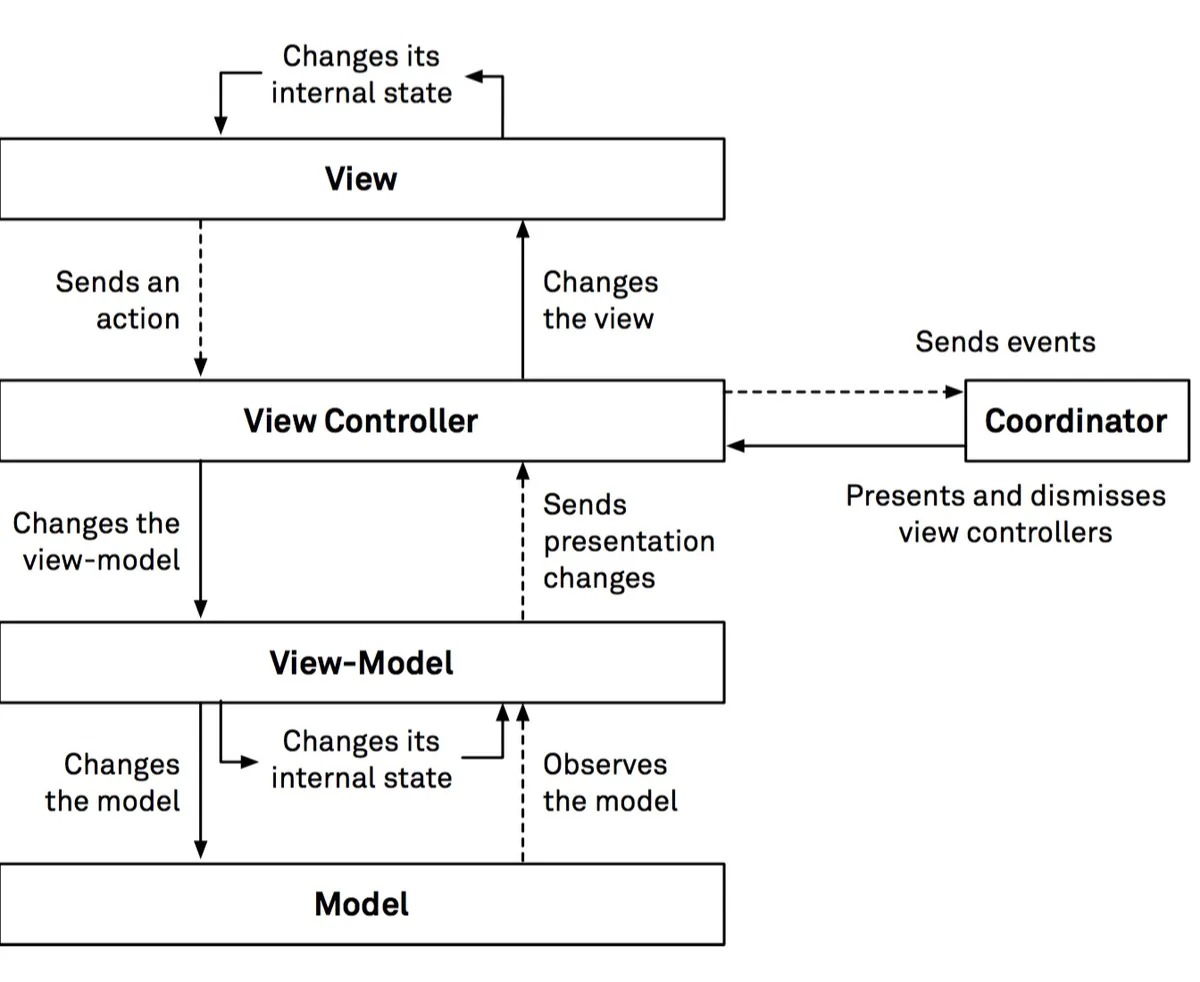
**METHODOLOGY**

**Technical Requirements**

**Swift Programming -**

Swift is a programming language developed by Apple Inc. for developing applications for iOS, macOS, watch OS and type OS platforms. It was first introduced in 2014, and it has rapidly gained popularity among developers due to its modern syntax and robust features. It is designed to be easy to learn and use, with a clean and concise syntax that makes code easy to read and write. It is a compiled language, which means that code is translated into machine code before it is run, resulting in faster performance. Some of the key features of Swift include optional, which helps developers handle the absence of a value, and closures, which allow for code to be passed around and executed at a later time. Swift also has a powerful type of system, which helps developers catch errors at compile time.

Swift is an open-source language, which means that it is available for anyone to use and contribute to. This has led to a thriving community of developers who share their code and provide support to one another. It can be used with Apple's integrated development environment (IDE), Xcode, which provides tools for building, debugging, and testing applications. Xcode includes a visual editor for designing user interfaces, as well as a source code editor for writing code. Overall, Swift is a modern and powerful programming language that is well-suited for developing applications for Apple's platforms. Its ease of use and robust features make it a popular choice among developers, and its open-source nature ensures that it will continue to evolve and improve over time.



**Use Cases of Swift Programming -**

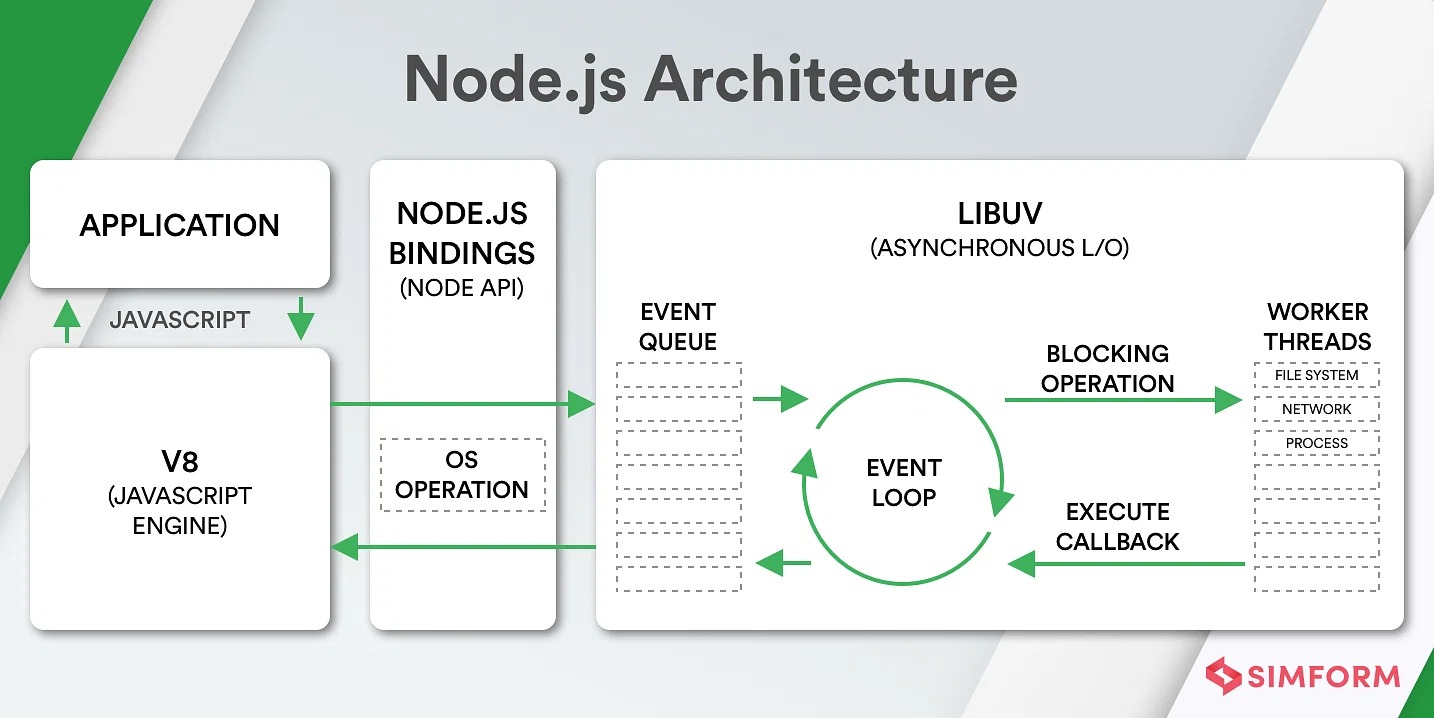
* **IOS and macOS app development:** Swift is the primary programming language used for developing iOS and macOS applications. With its clean syntax, type safety, and high performance, Swift provides a great development experience for building native apps for these platforms.
* **Server-side programming:** Swift can also be used for server-side programming. Swift on the server is particularly useful when you need to build scalable, fast, and secure web applications.
* **Augmented Reality:** Swift programming language is widely used in augmented reality (AR) development. ARKit, Apple's framework for building AR experiences on iOS devices, is primarily built with Swift.
* **Machine learning:** Swift is also used in machine learning applications. Apple has developed a powerful framework called Core ML that allows developers to build machine learning models that can be deployed on iOS and macOS platforms.
* **Wearable technology:** Swift is also used for building apps for wearable technology such as Apple Watch. With its high performance and low resource usage, Swift provides a great development experience for building apps for wearable devices.

**Node JS**

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that executes JavaScript code outside of a web browser. It allows developers to build scalable and high-performance applications by using an event-driven, non-blocking I/O model. It can be used in a variety of applications, including web servers, desktop applications, and mobile applications. In the context of iOS application development, Node.js can be used in several ways:

**Backend development:** Node.js can be used as a server-side platform to build the backend for iOS applications. It allows developers to create scalable and efficient web services that can handle a large number of requests from iOS clients. Real-time applications: Node.js is particularly useful for building real-time applications such as chat applications and multiplayer games. Its event-driven architecture allows developers to build applications that can handle a large number of real-time connections with low latency. Cross-platform development: With the help of Node.js, developers can write server-side code once and reuse it across multiple platforms, including iOS, Android, and web applications.

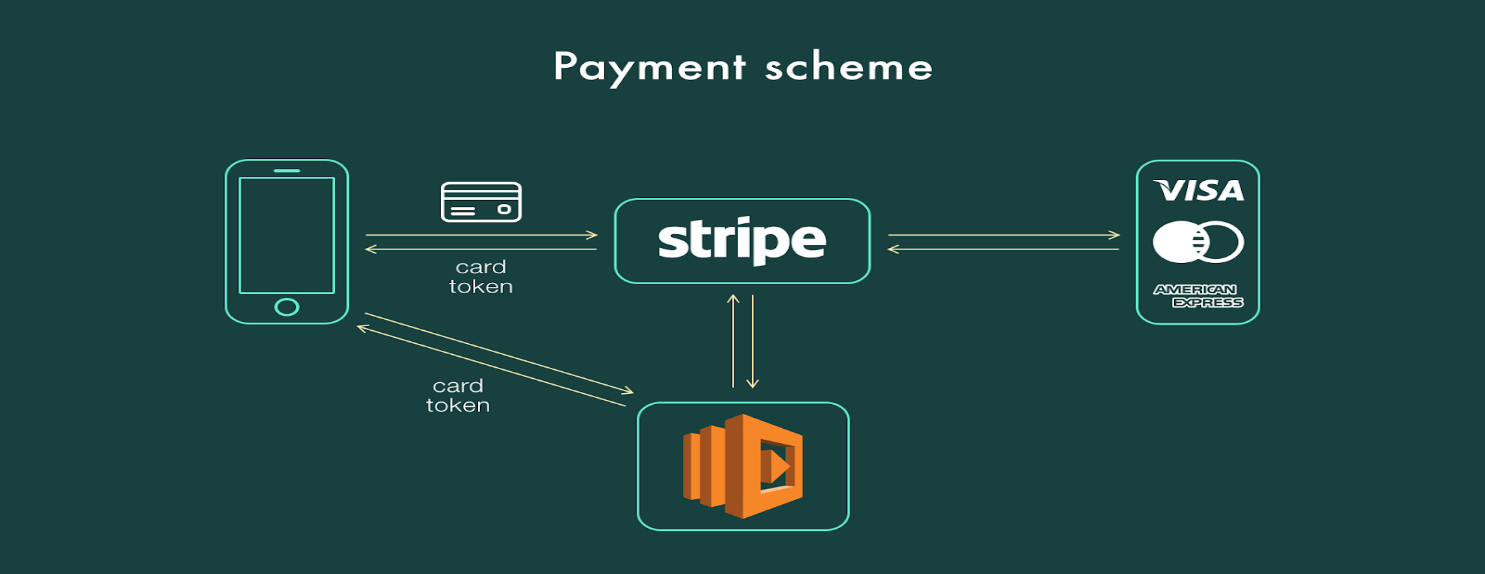
**JavaScript development:** Node.js is a great platform for developers who are already familiar with JavaScript. It allows developers to use the same language on both the front end and the back end of an application, which can lead to faster development cycles and easier maintenance. Package management: Node.js comes with a powerful package management system called NPM (Node Package Manager), which makes it easy for developers to find and install third-party packages and libraries.



**Stripe SDK**

Stripe is a popular payment gateway that provides a secure and reliable platform for businesses to accept and manage online payments. Stripe's SDK (Software Development Kit) is a set of tools that enables developers to integrate Stripe's payment processing services into their iOS applications. The Stripe SDK is designed to provide developers with a simple and efficient way to handle payment transactions within their mobile applications. One of the key advantages of the Stripe SDK is its ease of use. The SDK provides a simple and intuitive interface that allows developers to integrate Stripe’s payment processing services quickly and easily into their iOS applications. The SDK includes a wide range of features, including support for various payment methods, such as credit cards, debit cards, and Apple Pay. Additionally, the SDK provides a secure and reliable platform for processing payments, ensuring that customer data is protected at all times.

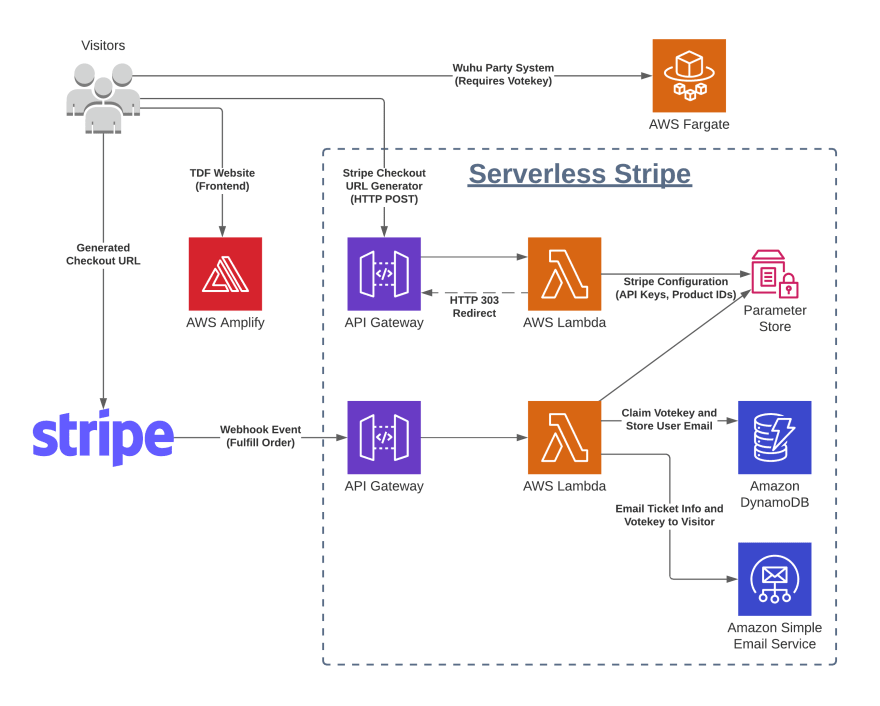
**Basic Architecture of How Stripe Works -**



Another key advantage of the Stripe SDK is its flexibility. The SDK is highly customizable, allowing developers to tailor the payment processing experience to meet the unique needs of their applications. Developers can choose from a wide range of customizable components, such as payment forms, checkout flows, and card entry fields. Additionally, the SDK provides support for multiple currencies, enabling developers to accept payments from customers around the world. The Stripe SDK also provides a range of tools and features that help developers to manage payments and reduce the risk of fraud. For example, the SDK includes built-in support for address verification, card verification, and fraud detection, helping to prevent fraudulent transactions and reduce chargebacks. Additionally, the SDK provides detailed reporting and analytics tools, enabling developers to track payment activity and identify potential issues before they become problems.

One of the key benefits of using the Stripe SDK is its integration with other popular iOS development tools and frameworks. For example, the SDK provides support for popular iOS development frameworks such as Swift and Objective-C, as well as popular third-party tools such as Cocoa Pods and Carthage. Additionally, the SDK provides a range of pre-built integrations with popular e-commerce platforms such as Shopify, WooCommerce, and Magento, enabling developers to easily integrate payment processing into their existing e-commerce workflows. Hence, the Stripe SDK is a powerful and flexible tool that provides developers with an easy-to-use and customizable platform for integrating payment processing into their iOS applications. The SDK provides a range of features and tools that help to reduce the risk of fraud, manage payments, and improve the overall payment processing experience for customers. With its seamless integration with popular iOS development tools and frameworks, the Stripe SDK is a popular choice for developers looking to build secure, reliable, and efficient payment processing capabilities into their mobile applications.

**Basic Architecture of How Stripe Works -**



**Firebase (Backend) -**

Firebase is a mobile and web application development platform, which provides a wide range of backend services to developers. It is a cloud-based service that enables developers to build, test, and deploy applications. Firebase was founded in 2011 and was acquired by Google in 2014. Since then, it has grown into a popular platform among developers, with over 1.5 million applications currently using Firebase. Firebase offers several backend services that make it easier for developers to build their applications. These services include real-time databases, authentication, hosting, storage, cloud messaging, and more. One of the main advantages of using Firebase is that it allows developers to focus on building the front end of their application, while Firebase takes care of the backend. One of the key features of Firebase is the real-time database. This is a NoSQL database that allows developers to store and sync data in real time. The database is hosted in the cloud, which means that it is highly scalable and can handle large amounts of data. Developers can easily integrate the real-time database into their applications using Firebase SDKs for various platforms, including iOS.

Another advantage of using Firebase is its authentication service. Firebase Authentication provides easy-to-use SDKs for authentication and authorization. It allows developers to authenticate users using a variety of methods, including email and password, Google, Facebook, Twitter, and more. Firebase Authentication also supports custom authentication, which means that developers can use their own authentication system. Firebase hosting is another useful service that allows developers to deploy and host their applications quickly and easily. Firebase hosting provides a secure, global CDN (Content Delivery Network) that ensures fast loading times for applications. It also supports SSL (Secure Sockets Layer) by default, which means that all connections to Firebase hosting are encrypted. Firebase storage is another backend service that allows developers to store and serve user-generated content, such as images and videos. Firebase storage is highly scalable and provides fast access to stored content. It also includes security features such as access control, which allows developers to control who can access the stored content.

Firebase cloud messaging is another service that allows developers to send notifications to users of their applications. Firebase cloud messaging supports sending notifications to iOS devices and allows developers to target specific users or groups of users with their notifications. It also provides advanced features such as message priority and data payload, which allow developers to send rich notifications to their users. In addition to these backend services, Firebase also provides a range of tools and SDKs that make it easy for developers to integrate Firebase into their applications. Firebase SDKs are available for a variety of platforms, including iOS, Android, and web. These SDKs provide easy-to-use APIs that allow developers to integrate Firebase services into their applications quickly and easily. Another advantage of using Firebase is its scalability. Firebase is a cloud-based service that is highly scalable and can handle large amounts of data. This makes it ideal for applications that are expected to grow rapidly or have a large user base. Firebase also provides a range of monitoring and analytics tools that allow developers to track the performance of their applications and make informed decisions about how to improve them.

Finally, Firebase is a cost-effective solution for backend development. Firebase offers a range of pricing plans, including a free plan that provides access to many of its services. This makes it an attractive option for developers who are just starting out or who have limited budgets. In conclusion, Firebase is a powerful backend development platform that offers a wide range of services to developers. Its real-time database, authentication, hosting, storage, cloud messaging, and other services make it easy for developers to build and deploy their applications quickly and easily. Its scalability, monitoring and analytics tools, and cost-effective pricing plans make it an attractive option for developers of all levels of experience.

**Basic Architecture of Working -**

Diagram

Description automatically generated

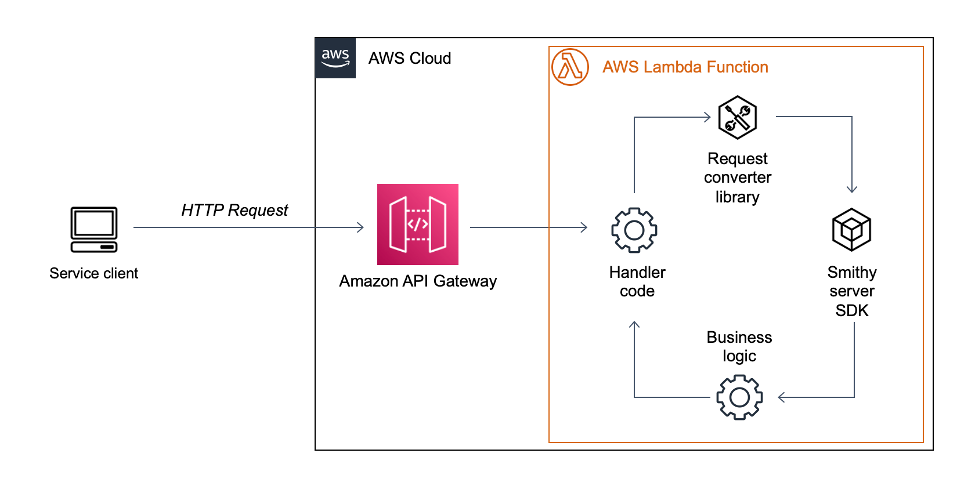
**AWS Lambda -**

AWS Lambda is a serverless computing service offered by Amazon Web Services (AWS) that allows developers to run code in response to specific events or triggers without having to provision or manage any servers. It is a popular choice for building serverless architectures and enables developers to build and run applications without worrying about the underlying infrastructure. Lambda is designed to work with other AWS services such as Amazon S3, Amazon DynamoDB, and Amazon Kinesis. It also provides support for programming languages such as Node.js, Python, Java, C#, Go, and Ruby.

The primary use case of AWS Lambda is to run code in response to specific events or triggers. This could be anything from an HTTP request to a file upload to an IoT device sending data to the cloud. The service is also commonly used for running back-end processes, such as data processing, image or video transcoding, and data analysis. One of the main advantages of AWS Lambda is its scalability. As the service is fully managed by AWS, it automatically scales up or down based on the amount of traffic or workload. This means that developers only pay for the actual computer time used and do not have to worry about managing servers, scaling the infrastructure, or capacity planning.

In terms of payment processing, AWS Lambda can be used in conjunction with other AWS services such as Amazon API Gateway, AWS Lambda, and Amazon DynamoDB to build a scalable and secure payment processing system. For example, developers can use AWS Lambda to process incoming payment requests, and then use Amazon API Gateway to securely expose the payment processing API to external applications. Overall, AWS Lambda is a powerful and flexible service that provides developers with a scalable and secure platform for building serverless applications. Its ability to automatically scale based on workload, support for multiple programming languages, and high level of security make it a popular choice for a wide range of use cases, including payment processing for mobile and web applications.

**The basic architecture of Lambda Use in our Project.**

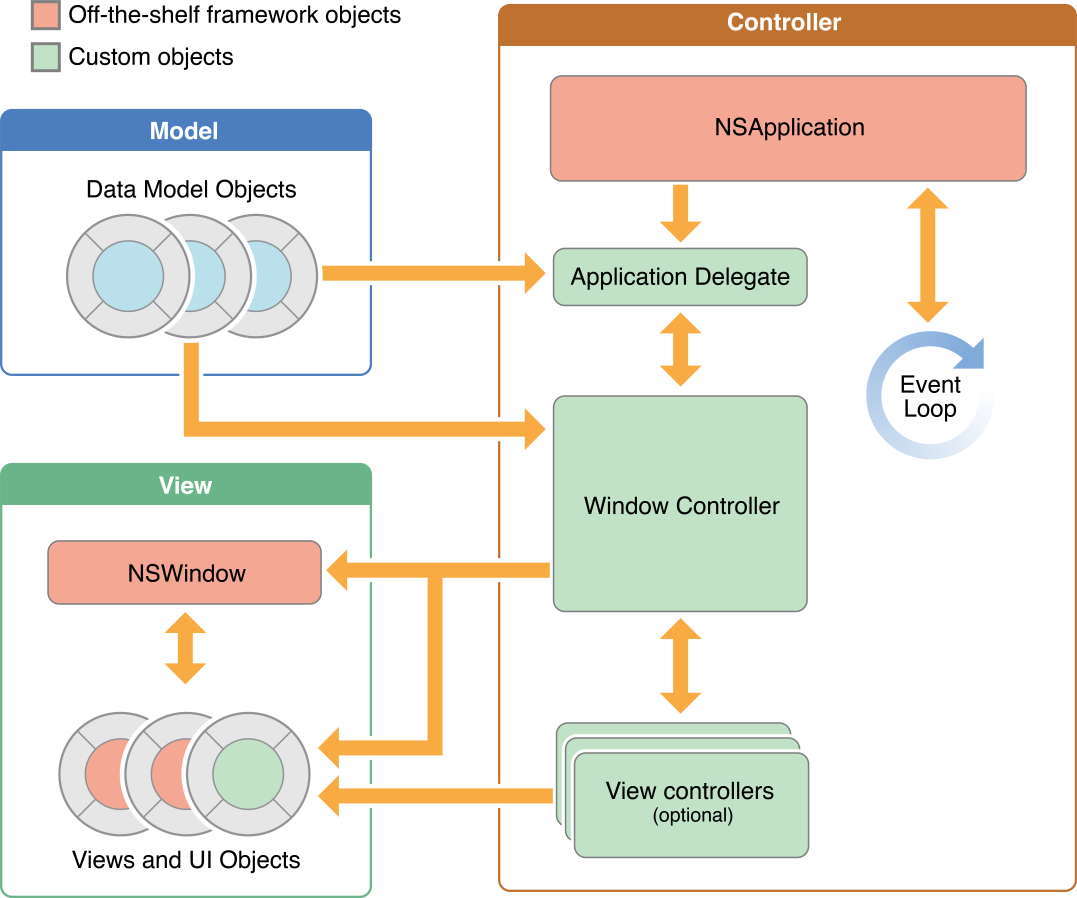


**X Code**

Xcode is an integrated development environment (IDE) for developing software applications for Apple's platforms, including iOS, macOS, type OS, and Watch OS. It is a powerful tool for developing apps, and it is the primary tool used by iOS developers. Xcode has been designed to provide developers with all the necessary tools for creating iOS applications, from coding and debugging to testing and publishing.

* **Native development:** Xcode is the official tool used by Apple for developing iOS applications. It provides native support for Apple's programming languages, including Swift and Objective-C. This means that Xcode is optimized for creating iOS apps, and it provides developers with all the necessary tools to create high-quality and responsive apps.
* **Rich development environment:** Xcode provides a rich development environment that includes a range of tools for coding, debugging, and testing iOS applications. The Xcode IDE includes code editors, a debugger, a visual interface builder, and a range of testing tools, making it easier for developers to create and test their apps.
* **Easy to use interface:** Xcode has a user-friendly interface that makes it easy for developers to create and manage iOS applications. The Xcode interface provides easy access to all the necessary tools and features, including project management, code editing, debugging, and testing. The interface also allows developers to easily navigate through their codebase, making it easier to find and fix bugs.
* **Collaboration:** Xcode has built-in collaboration tools that allow developers to work together on a single project. This makes it easier for teams to work together, share code, and collaborate on the same project.
* **Apple's App Store**: Xcode is required for publishing iOS apps to the Apple App Store. This means that developers must use Xcode to build, test, and submit their apps to the App Store. Xcode provides all the necessary tools and resources to create high-quality apps that meet Apple's standards for submission to the App Store.
* **Continuous updates:** Apple regularly updates Xcode with new features and improvements, ensuring that developers always have access to the latest tools and technologies for building iOS applications. This makes it easier for developers to create apps that take advantage of the latest iOS features and technologies.

**Basic Architecture of how X code environment supports IOS applications -**

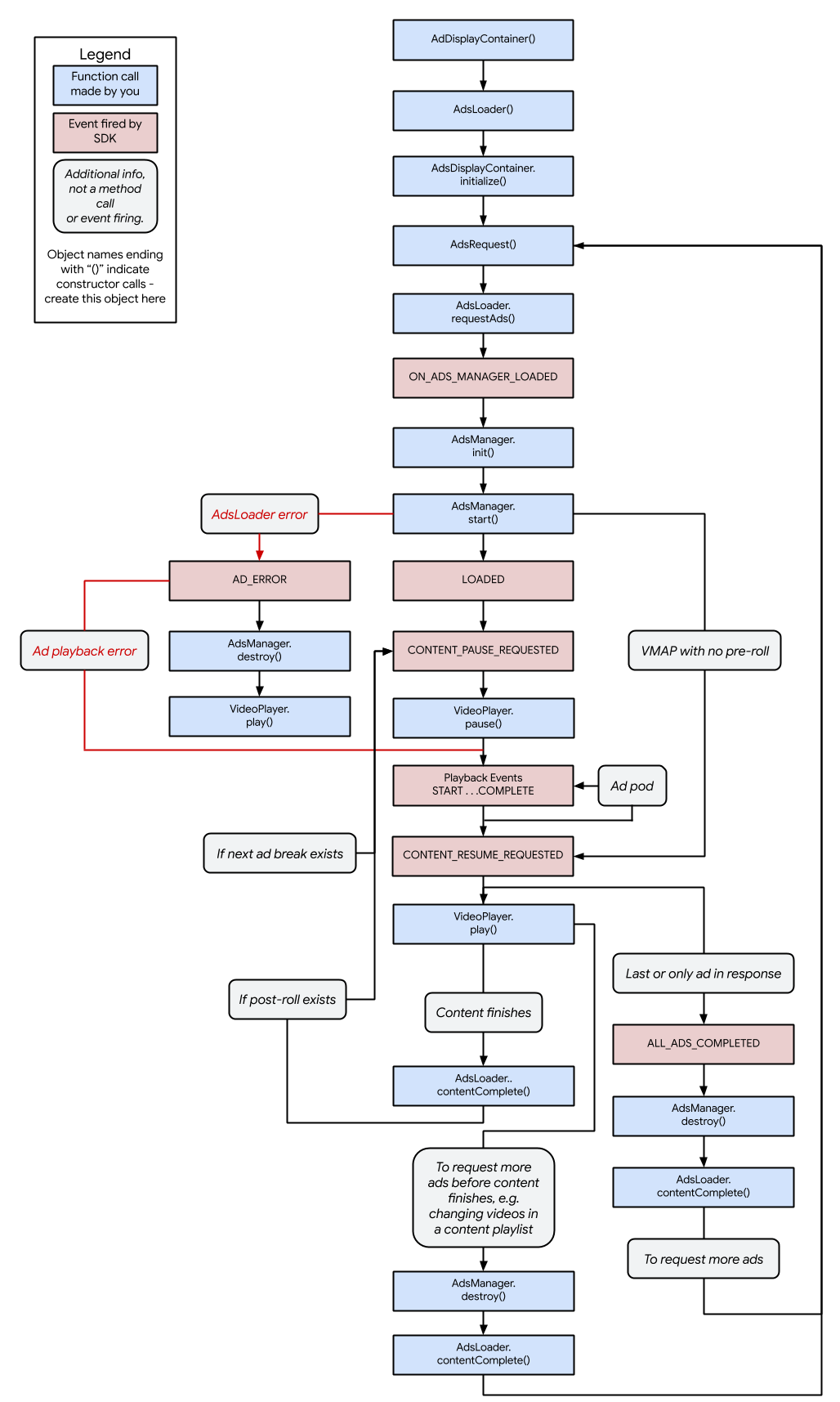


**SDK (API, QI Keyboard SDK, Side Menu Keyboard) -**

SDK stands for Software Development Kit, which is a set of tools and resources that developers use to build software applications. In the context of mobile app development, SDKs are often used to add specific features or functionalities to an app.

API stands for Application Programming Interface, which is a set of protocols, tools, and routines for building software applications. APIs allow different software systems to communicate with each other and share data. In the context of mobile app development, APIs are often used to integrate with third-party services, such as payment gateways or social media platforms. QI Keyboard SDK is a software development kit that enables developers to build custom keyboards for iOS devices. With QI Keyboard SDK, developers can create keyboards that support different languages, provide predictive text suggestions, and offer specialized input methods, such as emoji or GIFs.

Side Menu Keyboard is another SDK that enables developers to create custom keyboards for iOS devices. With Side Menu Keyboard, developers can create keyboards that offer additional features, such as quick access to frequently used phrases or the ability to switch between different input methods. Overall, SDKs like QI Keyboard SDK and Side Menu Keyboard provide developers with a range of tools and resources that can help them build high-quality, feature-rich mobile apps. By using SDKs, developers can save time and effort in developing specific features and functionalities, allowing them to focus on other aspects of the app development process. Additionally, using SDKs can help ensure that an app is compatible with different devices and operating systems, improving the user experience and increasing the app's overall appeal.

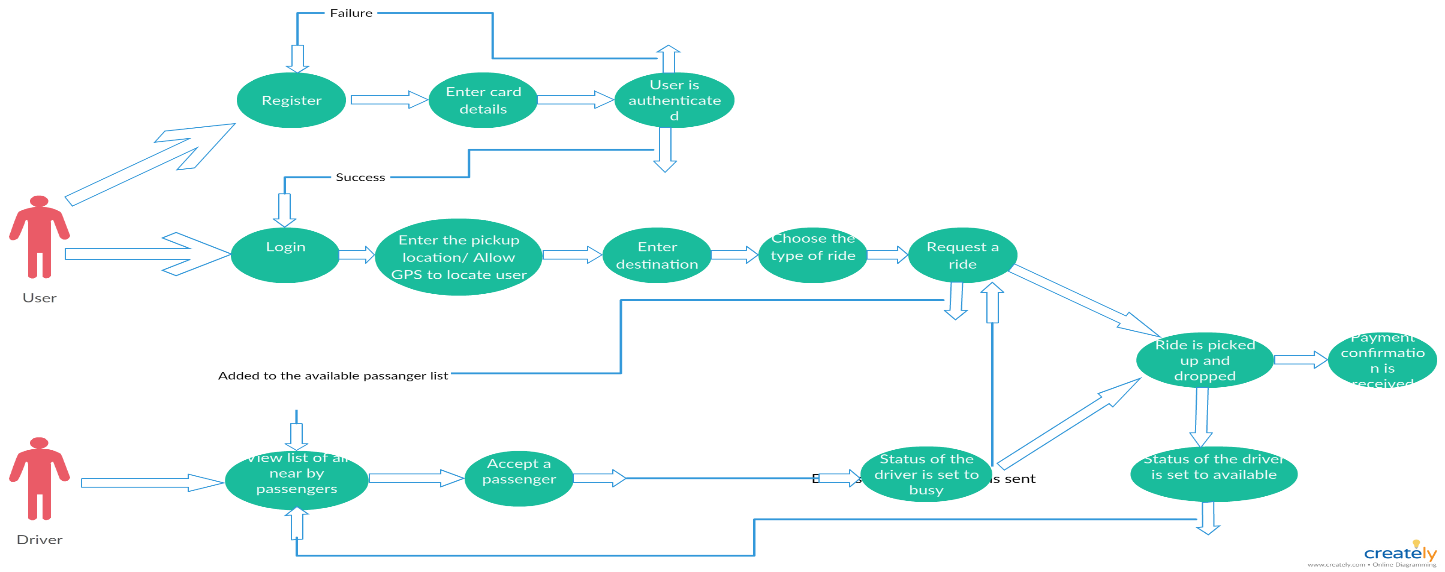


**Use Cases of our Project -**

Our Learn Drive project has numerous use cases that can benefit a wide range of users. Here are some of the key use cases:

* **Learner Drivers**: The primary users of our app are individuals who want to learn how to drive. They can use the app to find qualified driving instructors who are available to teach them how to drive. The app provides learners with a list of available instructors, along with their availability and rates, allowing them to choose the best instructor based on their preferences.
* **Driving Instructors**: Driving instructors can also use our app to connect with learners who are seeking their services. They can sign up on the app and create a profile highlighting their skills, experience, and rates. This will enable them to attract more learners, expand their customer base, and grow their business.
* **Uber and Door Dash Drivers**: Uber and Door Dash drivers who are qualified to teach driving can also use our app to earn extra income by providing driving lessons to learners. They can create a profile on the app and showcase their experience and qualifications, allowing them to reach out to a wider audience and increase their earnings.
* **Travelers**: Travelers who are visiting a new city or country can use our app to learn the rules of the road and obtain a driver's license in that location. By using our app, they can connect with local driving instructors who are familiar with the local driving laws and regulations. This will enable them to quickly obtain a license and drive safely in the new location.
* **Parents**: Parents who are teaching their children how to drive can use our app to find qualified instructors who can provide professional driving lessons. This can help parents ensure that their children receive the best possible training and become safe and responsible drivers.
* **Employers**: Employers who require their employees to have a driver's license can use our app to connect with qualified driving instructors who can provide training to their employees. This can help employers ensure that their employees are properly trained and can drive safely while on the job.
* **Driving Schools**: Traditional driving schools can also use our app to expand their customer base and reach out to more learners. By signing up on our app, they can showcase their services and attract new customers who are seeking driving lessons.

Our Learn Drive app has a wide range of use cases that can benefit learners, driving instructors, Uber and Door Dash drivers, travelers, parents, employers, and driving schools. By leveraging the power of technology, we are making it easier and more convenient for people to learn how to drive and become safe and responsible drivers.



**Use Case Diagram -**

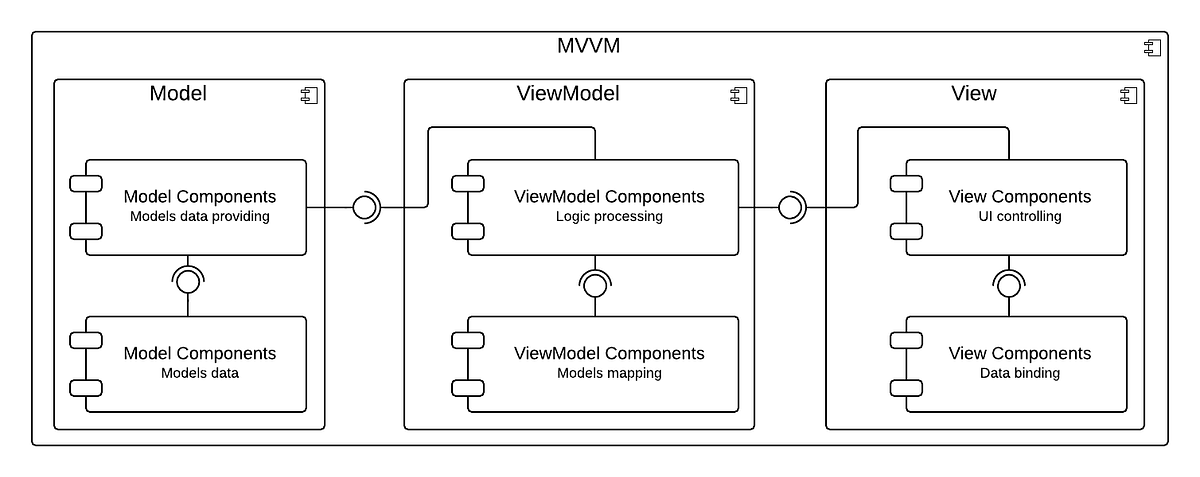
A use case diagram is used to describe the interaction between actors (users) and the system. It represents the functionalities of the system from the perspective of the users. Here is an example of a use case diagram for the Learn Drive project.

Diagram

Description automatically generated

**Class Diagram -**

A class diagram is used to show the classes, their attributes, methods, and their relationships with each other. Here is an example of a class diagram for the Learn Drive project.

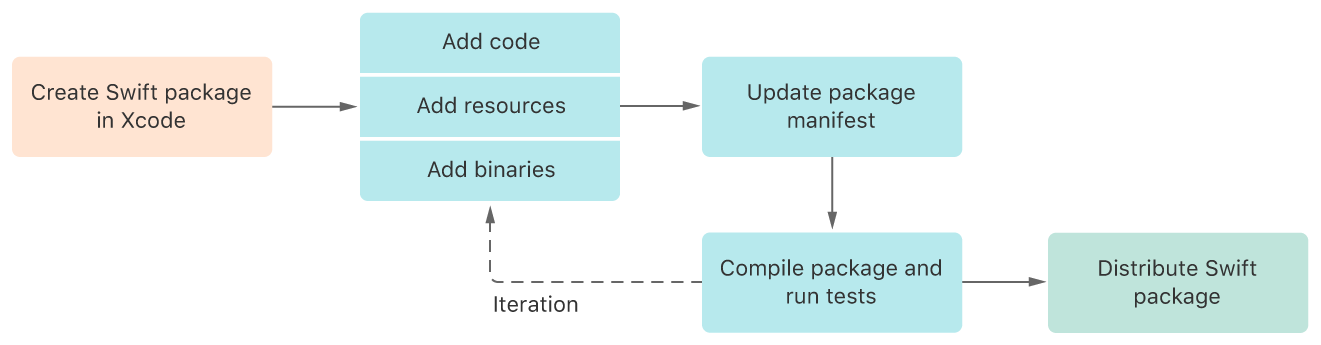


**Sequence Diagram -**

A sequence diagram is used to describe the interactions between objects in a specific order. It shows the messages exchanged between objects and the order in which they occur. Here is an example of a sequence diagram for booking a driving lesson in the Learn Drive project.

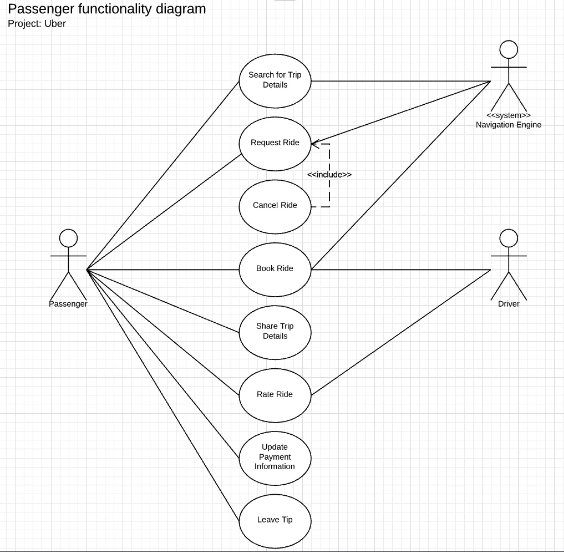
**Activity Diagram -**

An activity diagram is used to show the flow of activities or processes in the system. It shows the actions and decisions involved in a specific process. Here is an example of an activity diagram for the booking process in the Learn Drive project.



**Component Diagram -**

A component diagram is used to show the components of a system and their relationships with each other. It shows how the system is divided into different parts and how they interact with each other. Here is an example of a component diagram for the Learn Drive project.



**Basic Overview of the Project Methodology -**

Using **X-Code (IDE)** as an environment, our project uses **Swift** programming to write our code into develop our application. In that environment, we have our storyboard spaces to display our screens with the use of View controller to modify or write our code for the application.

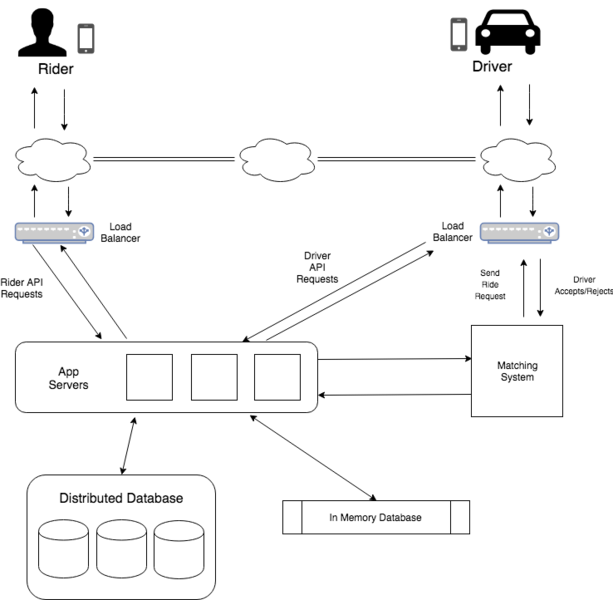
* We used **Segue** to connect and push the screens into each other so that they are interconnected by the Swift programming code.
* After connecting Segue, we write the code programmatically to display the pages whenever we want. They have buttons, labels, UI text fields, space bar and set up the keyboards using SDK.
* **Side Menu SDK** has also been used to put more fields into the application.
* **Firebase auth** is used for the authentication for User and driver login and sign up.
* **Fire store**, it is the database to store all the data in the IOS application to retrieve and used for the future.
* **Map kit** has been attached to our application to display the map for the user’s location and directions to the destination location and also search for the destination location.
* **UI Time picker,** we have used to select for user and driver to schedule the time slots.

**Basic Architecture of our project -**

**Presentation layer**: This layer represents the user interface of the application. It includes all the screens, buttons, and other visual components that users interact with. The presentation layer communicates with the other layers of the application to retrieve and display data.

**Application layer**: This layer contains the business logic of the application. It handles user requests, processes data, and communicates with other layers to perform operations. The application layer is responsible for managing the flow of data and ensuring that the application functions as intended.

**Data access layer**: This layer is responsible for interacting with the database or other data sources to retrieve and store data. It provides a way for the application layer to access and manipulate data without directly interacting with the database.



**Infrastructure layer**: This layer contains the services and components that support the application. It includes the hosting platform, networking components, and other infrastructure services required to run the application.

In the context of your Learn Drive project, the presentation layer would include the mobile app interface that users interact with. The application layer would handle requests for driving lessons, scheduling, and payment processing. The data access layer would interact with a database or other data source to retrieve information about available driving instructors, lesson schedules, and other data needed by the application. Finally, the infrastructure layer would provide the necessary components to support the application, such as cloud hosting, networking, and security services.

Overall, the architecture of your Learn Drive project should be designed to ensure that the application is scalable, reliable, and secure. By separating the various components of the application into distinct layers, you can more easily manage changes and updates, while also ensuring that the application remains stable and functional for users.

**CODE AND SAMPLE OUTPUT**

**Sample code for Root View controller**:

// ViewController.swift

// L Drive

//

// Created by Dharma Teja Donepudi on 2/27/23.

//

import UIKit

class ViewController: UIViewController {

// Outlet to a label named "driverLabel" in the storyboard

@IBOutlet weak var driverLabel: UILabel!

override func viewDidLoad() {

super.viewDidLoad()

// Set the text of driverLabel to an empty string

driverLabel.text = ""

// Initialize a variable to keep track of the character index

var charIndex = 0.0

// The text to be displayed in driverLabel

let titletext = "Make money as a Driver"

**Output-**

Graphical user interface

Description automatically generated

**Description**

This code is for a view controller in an iOS app called "L Drive". Here's what it does:

The view controller has an outlet to a label named "driverLabel" in the storyboard, and this label is initially set to an empty string.

In the viewDidLoad() method, the code sets up a loop to iterate through each character in the string "Make money as a Driver". For each character, a timer is created that waits for a delay based on the character index and then appends the current character to the text of driverLabel. This creates an animation effect of the text being typed out.

The view controller also has an action method for a button named "loginUser" in the storyboard. When this button is pressed, it performs a segue with identifier "loginuser", which presumably takes the user to a login page.

The view controller overrides prepare(for:sender:), which is a method that is called just before a segue occurs. Currently, this method is empty, so it doesn't do anything. However, this method can be used to perform any necessary setup before a segue occurs, such as passing data between view controllers.

**Sample code for User Signup page**

@IBAction func signupClick(\_ sender: UIButton) {

emailTextf.endEditing(true)

usernameText.endEditing(true)

passwordTextf.endEditing(true)

confirmpasswordTextf.endEditing(true)

phonenumberText.endEditing(true)

guard let email = emailTextf.text,

let password = passwordTextf.text,

let confirmpassword = confirmpasswordTextf.text,

let username = usernameText.text,

let phoneNumber = phonenumberText.text,

!email.isEmpty,

!password.isEmpty,

!phoneNumber.isEmpty,

!username.isEmpty,

password == confirmpassword

else { return }

**Output**

Graphical user interface, application

Description automatically generated

**Description**

This code is a Swift implementation of a sign-up screen for an iOS application. It allows users to sign up with their email address, password, phone number, and username.

The code uses the Firebase authentication service to create a new user account with the provided email and password. If the account creation is successful, the code sends a verification email to the user's email address and stores the user's information (username, email, and phone number) in a Firestore database.

The view controller class Signup\_User is the main component of this code, and it inherits from UIViewController. It includes several properties, such as text fields for user input (usernameText, emailTextf, phonenumberText, passwordTextf, confirmpasswordTextf) and a reference to the Firestore database (db).

The viewDidLoad() method sets the delegate for each of the text fields to the Signup\_User class, so that the class can handle user input events. The textFieldShouldReturn() method is implemented to handle the user pressing the return key on the keyboard.

The signupClick() method is called when the user taps the "Sign Up" button. It checks that all the required fields are filled in and that the password and confirm password fields match. If the validation succeeds, the code uses Firebase authentication to create a new user account with the provided email and password. If the account creation is successful, it sends a verification email to the user's email address and stores the user's information (username, email, and phone number) in a Firestore database.

Overall, this code provides a basic implementation of a sign-up screen for an iOS application that uses Firebase authentication and Firestore to store user information.

**Sample code for User Login Page**

@IBAction func loginClicked(\_ sender: UIButton) {

emailTextfield.endEditing(true)

passwordTextf.endEditing(true)

guard let email = emailTextfield.text else { return }

guard let password = passwordTextf.text else { return }

Auth.auth().signIn(withEmail: email, password: password) { firebaseResult, error in

if let e = error {

if error == nil {

// Login successful

let alertController = UIAlertController(title: "Login Successful", message: "You have successfully logged in.", preferredStyle: .alert)

let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)

alertController.addAction(okAction)

self.present(alertController, animated: true, completion: nil)

} else {

// Login failed

let alertController = UIAlertController(title: "Login Failed", message: error?.localizedDescription, preferredStyle: .alert)

let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)

alertController.addAction(okAction)

self.present(alertController, animated: true, completion: nil)

}

print(e)

}

else {

self.performSegue(withIdentifier: "gotouser", sender: self)

}

**Output-**

Graphical user interface, application

Description automatically generated

**Description**

This is Swift code for a login page of an iOS app called "L Drive". The code uses Firebase authentication for the login process. Here is a breakdown of the different parts of the code:

The first few lines of code import the necessary modules, namely UIKit and Firebase.

The Loginuserpage class is defined as a subclass of UIViewController and conforms to the UITextFieldDelegate protocol.

The @IBOutlet keyword is used to create a connection between the email and password text fields on the storyboard and the emailTextfield and passwordTextf variables in the code, respectively.

In the viewDidLoad() method, the two text fields' delegate is set to self, which means that the view controller is responsible for handling the user's input into the text fields.

When the user taps the "Login" button, the loginClicked(\_:) function is called. In this function, the endEditing(\_:) method is called on both text fields to dismiss the keyboard.

The user's input is then extracted from the email and password text fields using optional binding (guard let statements). If either text field is empty, the function simply returns without attempting to sign in.

The signIn(withEmail:password:completion:) method is called on the Auth object to authenticate the user. The completion block is used to handle the result of the authentication attempt.

If an error occurs during the authentication, the error is printed to the console, and an alert controller is presented to the user indicating that the login failed. If the login is successful, a different alert controller is presented to the user indicating that the login was successful, and a segue is performed to transition to a different view controller.

Finally, the textFieldShouldReturn(\_:) method is implemented to dismiss the keyboard when the user taps the "Return" key on the keyboard.

**Sample code for Driver Signup**

@IBAction func singupClick(\_ sender: UIButton) {

emailTextfield.endEditing(true)

usernameTextfield.endEditing(true)

passwordTextfield.endEditing(true)

confirmpasswordText.endEditing(true)

phonenumberText.endEditing(true)

guard let email = emailTextfield.text,

let password = passwordTextfield.text,

let confirmpassword = confirmpasswordText.text,

let username = usernameTextfield.text,

let phoneNumber = phonenumberText.text,

!email.isEmpty,

!password.isEmpty,

!phoneNumber.isEmpty,

!username.isEmpty,

password == confirmpassword

else { return }

**Output-**

Table

Description automatically generated with low confidence

**Description**

This code is for the signup view controller for a driver in an iOS application. The user is prompted to enter their email address, phone number, username, password, and confirm password.

When the user clicks on the signup button, the code first checks if all the fields are filled out and if the password matches the confirm password field. If any of the fields are missing or the password does not match, the code simply returns and does nothing.

If all the required fields are filled out and the passwords match, the code uses Firebase authentication to create a new user with the given email and password. If the creation of the user is successful, the user's display name is set to their username, and a verification email is sent to their email address.

If there is any error while creating the user or sending the verification email, an error message is displayed to the user using a UIAlertController.

After the user is successfully created, their username, email, and phone number are stored in Firestore, a NoSQL cloud database service provided by Firebase.

The view controller also implements the UITextFieldDelegate protocol to dismiss the keyboard when the user taps the return key on the keyboard. This ensures a better user experience by making the keyboard disappear when the user is done entering data into a text field.

**Sample code for Driver Login Page**

@IBAction func loginClicked(\_ sender: UIButton) {

emailTextfield.endEditing(true)

passwordTextfield.endEditing(true)

guard let email = emailTextfield.text else { return }

guard let password = passwordTextfield.text else { return }

Auth.auth().signIn(withEmail: email, password: password) { firebaseResult, error in

if let e = error {

if error == nil {

// Login successful

let alertController = UIAlertController(title: "Login Successful", message: "You have successfully logged in.", preferredStyle: .alert)

let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)

alertController.addAction(okAction)

self.present(alertController, animated: true, completion: nil)

} else {

// Login failed

let alertController = UIAlertController(title: "Login Failed", message: error?.localizedDescription, preferredStyle: .alert)

let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)

alertController.addAction(okAction)

self.present(alertController, animated: true, completion: nil)

}

print(e)

}

else {

self.performSegue(withIdentifier: "gotohome", sender: self)

**Output-**

Graphical user interface, application

Description automatically generated

**Description**

This is a Swift file that contains the implementation of a view controller that allows a driver to log in to an app called L Drive. The view controller has two text fields, one for the email and another for the password. The view controller is a subclass of UIViewController and conforms to the UITextFieldDelegate protocol.

In the viewDidLoad method, the view controller sets itself as the delegate for both text fields and sets the text content type of the email text field to .username and the password text field to .password. This allows the operating system to display the appropriate keyboard and suggestions to the user.

In the loginClicked method, the view controller retrieves the user's email and password from the text fields and passes them to the signIn method of Firebase Authentication. This method attempts to authenticate the user with the provided email and password. If the authentication succeeds, the view controller performs a segue to the app's home screen. If the authentication fails, the view controller displays an alert with the error message.

In the textFieldShouldReturn method, the view controller dismisses the keyboard when the user presses the return key on either of the text fields.

Overall, this code allows a driver to log in to an app using their email and password and provides feedback to the user if the login attempt fails.

**Sample Code for Driver Forgot Password**

@IBAction func sendClick(\_ sender: UIButton) {

emailText.endEditing(true)

let auth = Auth.auth()

auth.sendPasswordReset(withEmail: emailText.text!) { (error ) in

if error != nil {

let alertController = UIAlertController(title: "Reset Password", message: "Would you like to reset your password?", preferredStyle: .alert)

let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)

let resetAction = UIAlertAction(title: "Reset", style: .default) { (\_) in

// Perform the reset password action here

}

alertController.addAction(cancelAction)

alertController.addAction(resetAction)

self.present(alertController, animated: true, completion: nil)

**Output-**

Graphical user interface, application

Description automatically generated

**Description**

This code defines a view controller for the "forgot password" feature in an iOS app called "L Drive". It allows users to reset their password by entering their email address and clicking on a "send" button.

The forgotdriverpass class inherits from UIViewController and implements the UITextFieldDelegate protocol. It has a single text field (emailText) where the user can enter their email address.

The viewDidLoad() function sets the text field's delegate to the view controller and there is an implementation for the textFieldShouldReturn function that ends editing for the text field.

The sendClick function is triggered when the user clicks on the "send" button. It retrieves the text entered by the user in the email text field and calls the sendPasswordReset function of the Firebase Auth API to send an email with instructions to reset the password to the specified email address.

If the sendPasswordReset function fails with an error, the user is presented with an alert to confirm if they would like to reset their password. If the user clicks "Reset", the app would proceed to perform the password reset action.

The prepare function is commented out in the code, but it is usually used to perform any necessary setup before transitioning to a different view controller in the app.

**Sample code for User Forgot Password**

@IBAction func forgotClick(\_ sender: UIButton) {

emailTextf.endEditing(true)

let auth = Auth.auth()

auth.sendPasswordReset(withEmail: emailTextf.text!) { (error ) in

if error != nil {

if let error = error {

// If there was an error sending the reset password email, display an error message to the user

let alert = UIAlertController(title: "Error", message: error.localizedDescription, preferredStyle: .alert)

alert.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))

self.present(alert, animated: true, completion: nil)

} else {

// If the reset password email was sent successfully, display a message to the user

let alert = UIAlertController(title: "Password Reset Email Sent", message: "Please check your email to reset your password.", preferredStyle: .alert)

alert.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))

self.present(alert, animated: true, completion: nil)

**Output-**

Graphical user interface, application, Teams

Description automatically generated

**Description**

This is a Swift file for a ForgotpasswordUser view controller, which allows a user to reset their password if they have forgotten it. The view controller has a single UITextField called emailTextf and a UIButton called forgotClick.

In the viewDidLoad() method, the view controller sets itself as the delegate of the emailTextf text field.

When the user taps the forgotClick button, the forgotClick method is called. In this method, the view controller gets a reference to the Auth object provided by Firebase and calls its sendPasswordReset method with the email address entered by the user in the emailTextf text field.

If there is an error sending the reset password email, the view controller displays an alert message with the error description. Otherwise, the view controller displays a message asking the user if they would like to reset their password. If the user chooses to reset their password, the view controller would perform the reset password action.

Finally, the view controller has a textFieldShouldReturn method, which is called when the user taps the return key on the keyboard. This method dismisses the keyboard.

There is also an empty prepare(for:sender:) method that is typically used for preparing data to be passed to another view controller during a segue, but it is not used in this implementation.

**Sample code for User Home Page**

//

// UserHome.swift

// L Drive

//

// Created by Dharma Teja Donepudi on 3/15/23.

//

import UIKit

import SideMenu

import Firebase

import CoreLocation

import MapKit

class UserHome: UIViewController, CLLocationManagerDelegate, UITextFieldDelegate, MKLocalSearchCompleterDelegate{

private let sidemenu = SideMenuNavigationController(rootViewController:MenuController(with: ["user profile", "Support", "Settings"]) )

**Output-**

Graphical user interface, application

Description automatically generated with medium confidence

**Description**

This is a Swift file for an app's home screen view controller.

The view controller has the following functionalities:

A SideMenuNavigationController object that is initialized with a MenuController instance.

A CLLocationManager instance that requests permission to access the user's location and starts updating the location.

Two UITextField instances for entering the current location and destination.

A MKLocalSearchCompleter instance for autocompleting location search queries.

Three buttons that trigger different actions.

A logout button that signs out the user from the app.

A prepare(for:sender:) method that sets the startLocation and endLocation properties on the MapViewController instance when the user taps the "Ride" button and navigates to the map view.

A method for setting the region for the completer to search within.

A CLLocationManagerDelegate method that is called when location updates are available.

A CLLocationManagerDelegate method that is called when location authorization status changes.

A MKLocalSearchCompleterDelegate method that is called when search results are available.

A UITextFieldDelegate method that is called when the return key is pressed on a text field.

**Sample code for User booking ride by basic and Expert**

// Show the user's location on the map

mapView.showsUserLocation = true

mapView.userTrackingMode = .follow

// Request permission to access the user's location

locationManager.delegate = self

locationManager.requestWhenInUseAuthorization()

// Set the current location

if let userLocation = locationManager.location {

currentLocation = CLLocation(latitude: userLocation.coordinate.latitude, longitude: userLocation.coordinate.longitude)

**Output-**

Map

Description automatically generated with medium confidence

**Description**

The code you have provided is a Swift file that contains a view controller class called SecondMapViewController. This view controller is responsible for managing a map view and a search bar that allows the user to search for locations on the map. The view controller also has a UIDatePicker that the user can use to select a time duration.

The view controller class adopts several protocols, including UISearchBarDelegate, MKMapViewDelegate, UISearchResultsUpdating, CLLocationManagerDelegate, and UINavigationControllerDelegate.

The viewDidLoad method sets up the search controller and the map view. It also requests permission to access the user's location and sets the current location if available.

The timeSelected method is called when the user selects a time duration on the UIDatePicker. This method calculates the price of the ride based on the selected time and the distance between the current location and the destination location. It then saves this information to a Firestore database.

The showAlert method displays an alert to the user with a title and message.

The locationManager(\_:didChangeAuthorization:) method is called when the user grants permission to access their location. If the permission is granted, the method starts updating the user's location.

The locationManager(\_:didUpdateLocations:) method is called when the user's location changes. The method updates the user's current location on the map and stops updating the location.

The booknowTap method is called when the user taps the "Book Now" button. This method fetches the selected time from the UIDatePicker and calculates the remaining time in minutes. It then displays an alert to the user with the selected time and the calculated price of the ride.

**Sample code for payment method**

@IBAction func applePayButtonTapped(\_ sender: UIButton) {

// Handle Apple Pay button tapped

// Convert the price to cents (Stripe uses cents as the unit for currency)

let \_ = Int(price \* 100)

// Create a PaymentRequest for Apple Pay

let paymentRequest = StripeAPI.paymentRequest(withMerchantIdentifier: "merchant.CO.DHARMA.Learn-L-Drive.L-Drive", country: "US", currency: "USD")

// Configure the PaymentRequest with the required information

paymentRequest.paymentSummaryItems = [PKPaymentSummaryItem(label: "Your Item", amount: NSDecimalNumber(value: price))]

// Create a PaymentAuthorizationViewController with the PaymentRequest

let paymentAuthorizationViewController = PKPaymentAuthorizationViewController(paymentRequest: paymentRequest)

paymentAuthorizationViewController?.delegate = self

// Present the PaymentAuthorizationViewController

if let paymentAuthorizationViewController = paymentAuthorizationViewController {

self.present(paymentAuthorizationViewController, animated: true, completion: nil)

}

}

**Output-**

Graphical user interface

Description automatically generated with medium confidence

**Description**

This is a Swift file that defines a view controller for processing payments in an iOS app. It includes several methods for handling payment-related tasks, such as validating and processing credit card details, creating and processing payment methods, and handling Apple Pay payments.

The PaymentViewController class is defined as a subclass of UIViewController and conforms to several protocols: UITextFieldDelegate, PKPaymentAuthorizationViewControllerDelegate, and UINavigationControllerDelegate. These protocols allow the view controller to handle events and data from text fields, payment authorization views, and navigation controllers.

The PaymentViewController class includes several properties, including priceLabel, cardNumberTextField, expDateTextField, and cvvTextField, which are defined as outlets to interface elements in the view. There is also a price property, which is used to store the price of the item being purchased.

The viewDidLoad method sets up the view and initializes the priceLabel outlet to display the price of the item being purchased.

The viewWillAppear method sets up the Stripe API key.

The confirmButtonTapped method handles the event when the user taps the "Confirm" button. It first validates the credit card details entered by the user, then creates a STPCardParams object with the details, validates the card params, and creates a STPPaymentMethodParams object with the card details. Finally, it creates a payment method using the Stripe API client and processes the payment.

The applePayButtonTapped method handles the event when the user taps the "Apple Pay" button. It creates a payment request for Apple Pay, configures it with the required information, creates a payment authorization view controller with the payment request, and presents it to the user.

The showAlert method displays an alert message to the user.

The processPayment method further processes the payment after the payment method has been created. It extracts and stores the payment method ID, type, and details.

**Sample code for Drivers home Page**

//

// DriverHome.swift

// L Drive

//

// Created by Dharma Teja Donepudi on 3/15/23.

//

import UIKit

import SideMenu

import Firebase

class DriverHome: UIViewController {

private let sidemenu = SideMenuNavigationController(rootViewController:Menucontroller(with: ["Home", "Support", "Settings", "Logout"]) )

**Output-**

Text

Description automatically generated

**Description**

The code provided is an iOS app written in Swift that provides a basic user interface for a ride-hailing service, presumably for a driver's app. The app consists of a main screen where the driver can choose to start a ride immediately or schedule a future ride, a menu button to access other app functionalities, and a logout button to sign out of the app.

The DriverHome class is the main view controller of the app, responsible for managing the main screen and handling user interactions. It sets up the side menu by creating a SideMenuNavigationController and setting it as the default left menu navigation controller using SideMenuManager. It also adds a pan gesture to the main view to allow users to open and close the side menu. The viewDidLoad function sets the leftSide property of the side menu to true, which makes the menu slide in from the left side of the screen.

The Menucontroller class is a subclass of UITableViewController that manages the content of the side menu. The class takes an array of menu items as input, which are displayed as rows in the table view. The tableView(\_:numberOfRowsInSection:) function returns the number of items in the menuitems array, and tableView(\_:cellForRowAt:) function configures the content of each cell by setting the text label to the corresponding menu item. The tableView(\_:didSelectRowAt:) function is called when the user taps on a menu item, and handles the appropriate action for each item. In this case, selecting "Logout" signs out the user and navigates back to the root view controller.

**Sample code for driver Schedule Page**

@IBOutlet weak var endTimePicker: UIDatePicker!

override func viewDidLoad() {

super.viewDidLoad()

if #available(iOS 13.0, \*) {

// Set the appearance of the UIDatePicker to always follow system appearance

let datePicker = UIDatePicker()

datePicker.overrideUserInterfaceStyle = .unspecified

}

endTimePicker.minimumDate = startTimePicker.date

// Do any additional setup after loading the view.

}

**Output-**

Graphical user interface

Description automatically generated

**Description**

This is the implementation of a view controller class called ScheduleViewController in an iOS app written in Swift.

The class inherits from UIViewController, which means it represents a screen in the app.

The class has two UIDatePicker outlets called startTimePicker and endTimePicker and a button called scheduleButton.

When the view loads, viewDidLoad() method sets the minimum date of the endTimePicker to be the same as the date selected in the startTimePicker.

When the user taps on the scheduleButton, the scheduleButtonTapped() method is called. This method gets the selected start and end times from the date pickers, validates them, calculates the time interval between the current time and the start time, creates a local notification, and schedules it to fire when the start time arrives. Finally, it shows an alert to inform the user that the schedule is set.

The showAlert() method is a helper method that shows a simple alert message with a title and a message.

**CONCLUSION**

In conclusion, the Learn Drive project is an innovative solution to the problem of high cost and inconvenience associated with traditional driving schools. By leveraging mobile technology and real-time matching algorithms, the project enables users to easily find and book affordable and convenient driving lessons with experienced drivers. Throughout the development process, we utilized several cutting-edge technologies such as Swift programming, Firebase backend, Stripe SDK, and AWS Lambda, to create a user-friendly and reliable mobile application. The application's design includes several key features, including booking management, driver tracking, payment processing, and user feedback, to ensure the best possible experience for both users and drivers.

The success of this project can be attributed to the effective use of these technologies and the application's focus on addressing the needs of both learners and drivers. The use of Swift programming and SDKs such as Stripe and Firebase has allowed for a seamless and secure payment process, while the integration of AWS Lambda has enabled effective server-side communication. Overall, the Learn Drive project has the potential to revolutionize the driving school industry by providing a more affordable and convenient option for learners. The project's use cases span across a diverse range of users, including students, professionals, and travelers, seeking to learn driving. By addressing the challenges of traditional driving schools, the Learn Drive application represents a significant step towards more accessible and inclusive driving education.

In the future, the Learn Drive project could be further enhanced by incorporating new features such as gamification, personalized coaching, and social media integration. These additional features could increase user engagement and help to establish the Learn Drive brand as a leading provider of innovative driving education solutions. In conclusion, the Learn Drive project has the potential to create a positive impact on society by providing a cost-effective and convenient alternative to traditional driving schools. The application's use of cutting-edge technologies and focus on user experience positions it as a leader in driving education.