UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

iOS Application for live, interactive voice and video calls

Author:
Olha LEVANDIVSKA

Supervisor:
Serhiy MISKIV

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences and Information Technologies
Faculty of Applied Sciences

Declaration of Authorship

I, Olha LEVANDIVSKA, declare that this thesis titled, "iOS Application for live, interactive voice and video calls" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

iOS Application for live, interactive voice and video calls

by Olha LEVANDIVSKA

Abstract

In today's digital age, communication has become an integral part of our lives. With the widespread availability of high-speed internet, video and voice calls have become a popular mode of communication. This thesis focuses on the development of a new, easy to use, secure application for voice and video calls.
Code implementation is in repository: github.com/olyaLevand/CallFriendProject

Acknowledgements

I would like to acknowledge the support of my family and friends, who provided me with encouragement and motivation throughout my thesis work. I also want to thank every defender of Ukraine, thanks to whom I have safe conditions to live, study and work.

Contents

Declaration of Authorship **i**

Abstract **ii**

Acknowledgements **iii**

1 Introduction **1**
- 1.1 Motivation 1
- 1.2 Goal 1
- 1.3 Solution 1

2 App Architecture **3**
- 2.1 Choosing Architecture 3
- 2.2 What is MVVM? 4
- 2.3 Why MVVM? 5

3 Database **7**
- 3.1 Why Firebase? 7
- 3.2 Authentication with Firebase 7
- 3.3 Cloud Firestore 8
  - 3.3.1 Using Cloud Firestore for storing the list of available users . . . 9

4 Instruments **10**
- 4.1 Basic Instruments 10
- 4.2 Why Sinch framework? 10
- 4.3 Callkit framework 11

5 Call Flow **13**
- 5.1 Creating the client 13
- 5.2 Register client with JWT? 13
  - 5.2.1 Why we need JWT 13
  - 5.2.2 How to create the client 14
- 5.3 Making outgoing call 14
- 5.4 Receiving a call 14
  - 5.4.1 What is VOIP pushes? 14
  - 5.4.2 How Sinch send VOIP pushes in iOS 15
  - 5.4.3 How to receive VOIP pushes in iOS 15

6 App Structure **17**
- 6.1 User Flow 17
- 6.2 App Screens 17
  - 6.2.1 Login Screen 17
  - 6.2.2 Home Screen 17
  - 6.2.3 Voice or video call Screens 18

6.2.4 List of available users Screen 18

List of Figures

2.1 MVVM 4
6.1 User Flow 18

List of Abbreviations

APNS	**A**pple **P**ush **N**notification **S**ervice
VoIP	**V**oice **O**ver **IP**
UI	**U**ser **I**nterface
JSON	**W**eb **T**okens
SDK	**S**oftware **D**evelopment **K**it
MVVM	**M**odel **V**iew **V**iew-**M**odel

Dedicated to my parents...

Chapter 1

Introduction

1.1 Motivation

Vice and video calling have become increasingly important in today's world, particularly with the widespread adoption of remote work and virtual communication.

One of the biggest benefits of voice and video calling is that they allow people to communicate with each other in a more personal and effective manner than traditional text-based methods. With voice and video calling, people can hear and see each other's expressions, which helps to convey emotions and context that can be missed through text-based communication.

Voice and video calling can also help to enhance personal connections among people, particularly in situations where physical distance is a barrier. For example, video calling can allow grandparents to see and interact with their grandchildren who live far away. This helps to maintain important relationships and connections, even when physical distance separates people.

1.2 Goal

The goal is to provide users with a service that would allow them to call in the fastest and easiest way possible.

1.3 Solution

The solution is to create the application for voice and video calls with following characteristic:

- **Ease of Use**. One of the most important considerations when creating an application for voice and video calls is ease of use. The application should be designed in such a way that it is easy for people of all technical abilities to use. This means that the user interface should be intuitive and user-friendly, with clear instructions and help resources available if needed.

- **Security**. Another key consideration when creating an application for voice and video calls is security. The application should be designed with strong security measures in place to protect users' privacy and prevent unauthorized access. This may include end-to-end encryption, two-factor authentication, and other security features.

- **Reliability**. The application should also be reliable, with minimal downtime and interruptions. Users should be able to rely on the application to provide a high-quality, uninterrupted voice and video calling experience. This may involve incorporating redundancy and failover mechanisms to ensure that the application remains operational even in the event of a hardware or software failure.

Chapter 2

App Architecture

2.1 Choosing Architecture

Choosing a certain architecture for your iOS app determines the design of different aspects of your software. Within that, it also determines what kind of design patterns you will want to use. A Pattern in the context of software is a common response to a recurring problem. It uses a common language to outline the elements needed to solve a software challenge. Patterns are particularly useful when working with a large team of developers. They can make your developers substantially more efficient by using a refined approach to solving problems. Patterns ensure changes to the code are consistent and reduce risk for major changes. Good use of Architecture Patterns makes it easy for developers to understand the software. In the end that makes it easy to change it. This means new features can be delivered faster, with fewer bugs, and therefore easier fixes.

Aim of an Architecture

- Increase testability
- Improve maintainability
- Scale with team size

The solution is actually quite simple: **Modularization**

Modularization means separating a program's functionality into independent, interchangeable units, each responsible for only one aspect of the program.

For example, in Swift, you can outsource code into frameworks, such as a server module, which is only responsible for communicating with a server. In the app, server requests are no longer performed but only via the server module. This server module can be replaced, rewritten and tested separately from the rest of the app. But, you do not have to outsource everything into frameworks. It's often enough if you write specialized classes that are only responsible for one aspect and are referenced via protocols. Again, the code is bundled, does only one thing and can be easily replaced thanks to the protocol. Divided into modules, one usually quickly fixes the Massive-View-Controller problem because code that does not actually belong in a ViewController is outsourced into modules. The ViewController then shrinks automatically. Interchangeability increases testability because it makes it easy to mock dependencies and test modules separately. And if every member of the team only works on their independent code module that also has a highly standardized interface, merge conflicts should be very rare, so it scales well with the team size. All the common architectures are usually based on modularization. Special classes such as controllers, models or workers, however you may call them, separate code and become more testable via protocols. How exactly this has to be separated is dictated by the respective design pattern.

Nowadays we have many options when it comes to architecture design patterns:

- MVC
- MVP
- MVVM
- VIPER

2.2 What is MVVM?

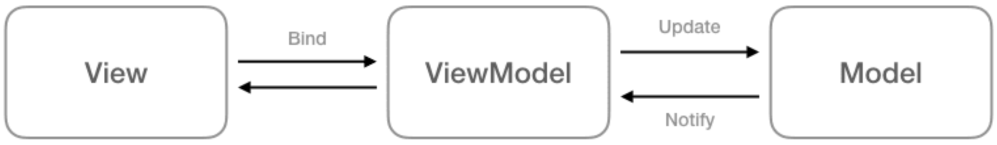

FIGURE 2.1: MVVM

The Model-View-ViewModel (MVVM) architecture is a design pattern commonly used in iOS app development. It provides a structured way to separate the user interface (UI) logic from the business logic of an application.

Here's a breakdown of the three core components in MVVM:

- **Model**: The model represents the data and business logic of the application. It typically consists of classes or structs that define the data structures, perform data fetching and manipulation, and encapsulate the application's state.
- **View**: The view represents the user interface components that the user interacts with. It includes UI elements such as buttons, labels, and text fields. In MVVM, the view should be kept as lightweight as possible and focus solely on displaying data and capturing user input. It should not contain any business logic.
- **ViewModel**: The view model acts as a mediator between the view and the model. It exposes the data and commands required by the view and provides methods or properties for data binding. The view model receives input from the view, processes it using the model, and updates the view with the results. It also notifies the model of any changes in the view's state.

In MVVM, the view and the view model are connected through data binding mechanisms. This means that the view binds to the properties of the view model, and any changes in the view model automatically update the view, and vice versa. This decoupling allows for easier testing, reusability, and maintainability of the code.

Additionally, MVVM encourages the use of reactive programming and observable patterns. This means that the view model can expose observable properties that the view can subscribe to, and whenever those properties change, the view automatically reflects the updated data.

MVVM architecture can be implemented in iOS using various frameworks and libraries such as UIKit, SwiftUI, Combine, or ReactiveCocoa, depending on the version of iOS and personal preference.

2.3 Why MVVM?

The Model-View-ViewModel (MVVM) architecture has gained popularity in iOS app development due to its ability to promote code organization, maintainability, and reusability. By separating the concerns of data, UI, and business logic, MVVM offers several advantages over traditional approaches. This essay explores the benefits of adopting MVVM architecture in iOS applications.

- Separation of Concerns: One of the key advantages of MVVM is its ability to separate concerns between the model, view, and view model. This clear separation allows developers to focus on specific aspects of the application, making the codebase more maintainable. The model encapsulates data and business logic, the view represents the user interface, and the view model acts as the mediator between the two. This separation enables better code organization and ease of understanding.

- Testability: MVVM architecture promotes testability by reducing dependencies between components. Since the view model does not have a direct dependency on the view, it becomes easier to write unit tests for the business logic without involving the UI. With the help of dependency injection, developers can provide mock objects to the view model, allowing comprehensive testing of the application's functionality. The separation of concerns in MVVM architecture facilitates a more robust and efficient testing process.

- Reusability: MVVM architecture encourages the creation of reusable components. By separating the UI logic from the business logic, views can be designed in a way that they can be easily reused across multiple screens or even in different projects. View models encapsulate the logic specific to a particular view and can be shared between different instances of that view. This reusability not only improves development efficiency but also ensures consistent behavior and user experience throughout the application.

- Data Binding and Reactive Programming: MVVM architecture is well-suited for leveraging data binding and reactive programming paradigms. With the use of frameworks like Combine or ReactiveCocoa, changes in the view model can be automatically propagated to the view, and vice versa, through observable properties and bindings. This real-time synchronization simplifies UI updates and reduces the need for manual event handling. Reactive programming enables developers to write clean and concise code, as complex UI interactions can be expressed in a declarative and reactive manner.

- Scalability: As iOS applications grow in complexity, maintaining a scalable architecture becomes crucial. MVVM architecture provides a scalable structure by decoupling the components and ensuring a modular approach. New features or changes can be added to the application without impacting other parts of the codebase, as long as the interfaces between the model, view, and view model remain consistent. This scalability makes MVVM suitable for both small and large iOS applications, allowing for easier code maintenance and extensibility.

The Model-View-ViewModel (MVVM) architecture offers several advantages in iOS app development. By promoting separation of concerns, testability, reusability, data binding, and scalability, MVVM improves code organization, maintenance, and overall development efficiency. Adopting MVVM architecture can lead to more maintainable, testable, and scalable iOS applications, providing a better user experience and reducing development time and effort. As MVVM continues to gain popularity, it is becoming an essential design pattern for iOS developers seeking to build robust and efficient applications.

Chapter 3

Database

3.1 Why Firebase?

Firebase is a powerful and versatile backend platform that offers a range of features and services that can help you build robust and scalable iOS applications. Whether you are building a simple app or a complex application, Firebase can provide the tools and functionality you need to bring your app to life. There are several reasons why you may choose to use Firebase as the database for your iOS app, some of which are discussed below.

- Real-time database: Firebase provides a real-time database that allows your iOS app to receive and update data in real-time, without requiring frequent server requests or refreshing the page. This can be particularly useful for apps that require real-time updates, such as chat applications or collaborative apps.
- Scalability: Firebase is designed to be scalable, meaning that it can handle large amounts of data and traffic without compromising performance or stability. This makes it a great choice for iOS apps that are expected to grow in popularity or size over time.
- Easy to use: Firebase is known for its ease of use and user-friendly interface, making it a great choice for developers who are new to backend development or who want to save time and effort on server-side tasks. Firebase offers a wide range of pre-built tools and integrations, which can help you quickly implement and customize your app's backend functionality.
- Analytics and Crash Reporting: Firebase provides analytics and crash reporting features that can help you track app usage, identify issues, and optimize your app's performance. This can be particularly useful for developers who want to improve their app's user experience and troubleshoot any problems that may arise.

3.2 Authentication with Firebase

Firebase Authentication is a simple and secure way to authenticate users in your iOS application, using a variety of sign-in methods such as email/password, phone number, Google, Facebook, Twitter, and more. In this essay, we will discuss how Firebase Authentication can be used to authenticate users in your iOS application.

One of the biggest advantages of Firebase Authentication is its ease of use. Firebase provides an easy-to-use SDK that allows you to integrate Firebase Authentication into your iOS app with just a few lines of code. Once you have integrated

Firebase Authentication into your app, you can start using it to authenticate users and manage their access to your app's resources.

To use Firebase Authentication in your iOS app, you will need to create a Firebase project and configure your app to use the Firebase SDK. Once you have done this, you can use the Firebase Authentication API to authenticate users in your app. You can choose from a variety of sign-in methods, including email/password, phone number, and third-party sign-in providers like Google, Facebook, and Twitter.

Firebase Authentication also provides a variety of security features to help protect your users' data and prevent unauthorized access to your app. For example, Firebase Authentication provides secure token-based authentication, which ensures that only authenticated users can access your app's resources. Firebase Authentication also supports multi-factor authentication, which provides an additional layer of security by requiring users to provide additional information to verify their identity.

Another advantage of Firebase Authentication is its flexibility. Firebase Authentication allows you to customize the sign-in flow for your app, so you can provide a seamless and consistent user experience. You can also use Firebase Authentication to manage user accounts and permissions, allowing you to control access to your app's resources and data.

3.3 Cloud Firestore

Cloud Firestore is a cloud-based NoSQL database that is part of the Firebase platform provided by Google. It is a great way to store simple data for iOS applications because it is scalable, easy to use, and reliable. Developers can use Cloud Firestore to store user-generated content, application data, and configuration files. Here are some of the reasons why Cloud Firestore is an excellent choice for storing simple data in iOS apps:

Ease of Use: Cloud Firestore provides an intuitive API that allows developers to easily store and retrieve data from the cloud. This API is available in multiple programming languages, including Swift, making it easy to integrate into iOS applications. The API provides simple and easy-to-use methods to perform CRUD (create, read, update, and delete) operations on data.

Scalability: Cloud Firestore is a scalable database that can handle large amounts of data. It provides automatic sharding and indexing, which allows developers to scale their applications as needed. This means that as an iOS application grows and the data storage requirements increase, Cloud Firestore can automatically scale to meet the demand.

Real-time Updates: Cloud Firestore provides real-time updates, which means that changes to data are instantly propagated to all connected devices. This feature is particularly useful for iOS applications that require real-time updates, such as chat applications or real-time multiplayer games.

Security: Cloud Firestore provides built-in security features such as server-side security rules that help to protect data from unauthorized access. Developers can use these rules to restrict access to specific data based on the user's authentication status, role, or other criteria.

Offline Support: Cloud Firestore provides offline support, which means that iOS applications can continue to work even when the device is offline. When the device comes back online, any changes made while offline are automatically synchronized with the cloud.

3.3.1 Using Cloud Firestore for storing the list of available users

To make application using more convenient I add "Show the list of all available users" function. With that user can see all available user we can cake a call. To implement this feature I used Cloud Firestore to keep up to date list of all available users.The Successful log in added the user to the list and deleted him with log out function.

Chapter 4

Instruments

4.1 Basic Instruments

With regards to iOS app creation, I plan to utilize the most recent Apple toolkits. I will be using SwiftUI, which is compatible with iOS 16 and later versions, as well as Combine and Firebase for the backend.

SwiftUI is an approach to designing user interfaces that is declarative in nature and employs a reactive binding principle to dynamically update its contents.

Combine is a reactive framework that allows for the processing of data values over a period of time.

Firebase, developed by Google, is an SDK that expedites the app development process.

4.2 Why Sinch framework?

Sinch is a popular framework for voice and video calling in iOS that provides a range of features for developers. One of the primary reasons why developers choose Sinch for their voice and video calling needs is its ease of use. Sinch provides a simple and intuitive API that developers can use to quickly integrate voice and video calling into their iOS apps, without having to worry about the complexities of the underlying technology.

- **High-quality audio and video**. Another important benefit of Sinch is its high-quality audio and video. Sinch supports a range of codecs that provide high-quality audio and video, making it ideal for use cases where clear communication is essential. This can be particularly important for business or professional use cases, where high-quality communication is critical for effective collaboration.

- **Range of features for developers to customize and control calls**. Sinch also provides a range of features for developers to customize and control their voice and video calling experience. For example, developers can customize the user interface of their calling experience, add custom buttons or controls, and even integrate other features like messaging or file sharing into their calling experience. This flexibility allows developers to create a calling experience that is tailored to their specific needs and the needs of their users.

- **Security**. Sinch provides a range of security features to ensure that calls are secure and private. Sinch uses end-to-end encryption to ensure that calls are only accessible to the intended recipients, and also provides other security features like authentication and authorization to ensure that only authorized users can access calls.

- **Easy to use**. One of the ways that Sinch simplifies the integration process is through its comprehensive documentation and extensive support resources. The platform offers clear and concise documentation, tutorials, and sample code that guide developers through the process of integrating its services. Additionally, Sinch provides dedicated support resources that are available to developers 24/7. These resources include a support portal, email support, and a dedicated support team that is always ready to assist with any issues that arise.

  Another factor that makes Sinch easy to use is its flexible and customizable APIs. The platform offers a range of APIs that can be easily customized to suit the needs of individual applications. This allows developers to create bespoke communication solutions that are tailored to the unique requirements of their applications.

4.3 Callkit framework

CallKit is a framework introduced by Apple in iOS 10 that allows VoIP apps to integrate with the native Phone app and provide a better user experience for VoIP calls. With CallKit, VoIP apps can provide a more seamless and integrated calling experience for users, similar to the experience of making traditional phone calls.

CallKit provides a set of APIs that allow VoIP apps to integrate with the native Phone app and display incoming and outgoing calls in the same interface as traditional phone calls. When a VoIP call comes in, CallKit displays a full-screen interface that shows the caller's name, profile picture, and call duration, just like a regular phone call. This makes it easier for users to identify and answer VoIP calls without having to open the app.

CallKit also provides several features that allow users to manage their calls more effectively. For example, users can use the built-in call-waiting and call-hold features to manage multiple calls at once. Additionally, users can use the built-in call-merging feature to merge multiple calls into a conference call.

CallKit also provides built-in support for audio handling, which can help to reduce the overall power consumption of the app. CallKit can also integrate with other iOS features, such as Siri and Bluetooth, to provide a more seamless and integrated calling experience for users.

Here are some of the advantages of using CallKit with VoIP pushes in iOS:

- Better user experience: CallKit provides a native interface for handling VoIP calls, which makes it easier for users to receive and manage VoIP calls just like regular phone calls. When a VoIP call comes in, CallKit displays a full-screen interface that shows the caller's name, profile picture, and call duration, just like a regular phone call. This makes it easier for users to identify and answer VoIP calls without having to open the app.

- Improved call management: CallKit provides several features that allow users to manage their calls more effectively. For example, users can use the built-in call-waiting and call-hold features to manage multiple calls at once. Additionally, users can use the built-in call-merging feature to merge multiple calls into a conference call.

- Increased reliability: VoIP pushes provide a more reliable way of receiving incoming calls compared to traditional push notifications. With VoIP pushes, the app can wake up and establish a connection to the server before the call comes in, which helps to ensure that the call is delivered even if the app is not running in the foreground. This can help to reduce missed calls and improve the overall reliability of the app.

- Reduced battery consumption: Using VoIP pushes with CallKit can also help to reduce battery consumption on the user's device. Since the app can establish a connection to the server before the call comes in, it can use less power when the call is actually received. Additionally, CallKit provides built-in support for audio handling, which can help to reduce the overall power consumption of the app.

- Improved security: VoIP pushes use a more secure mechanism for delivering incoming calls compared to traditional push notifications. VoIP pushes are encrypted end-to-end and can only be decrypted by the receiving device. This helps to ensure that incoming calls are delivered securely and cannot be intercepted or tampered with.

Chapter 5

Call Flow

5.1 Creating the client

When integrating Sinch's communication functionality into your iOS application, you need to create a client that communicates with the Sinch servers using Sinch's APIs and SDKs.

We need to create a Sinch client in iOS before making calls for following reasons:

- 1) **Authentication**: Creating a Sinch client allows you to authenticate your users and devices before they can use the communication functionalities provided by Sinch. This provides an additional layer of security and ensures that only authorized users and devices can use your application's communication functionalities.

- 2)**Customization**: Creating a Sinch client allows you to customize the communication experience for your users. You can integrate Sinch's communication functionality into your iOS application and offer a seamless user experience that is tailored to your application's needs.

- 3) **Configuration**: Creating a Sinch client allows you to configure the communication settings for your iOS application. You can specify the type of communication functionalities you want to use, set up call routing rules, and configure other settings that are important for your application's communication functionalities.

5.2 Register client with JWT?

5.2.1 Why we need JWT

During the client creating we need to register client with JSON Web Tokens (JWT). This tokens are commonly used for authentication and authorization in web and mobile applications. Sinch, being a cloud communication platform, uses JWT to authenticate and authorize clients when they interact with the Sinch API.

When you create a Sinch client in iOS, you need to provide a JWT token as a parameter to the client initialization method. This JWT token contains information about the client's identity and permissions, and it is used by Sinch to verify that the client is authorized to perform the requested actions.

By requiring a JWT token for client creation, Sinch ensures that only authorized clients can interact with its API, which helps to prevent unauthorized access and misuse of the API. Additionally, JWT tokens are encrypted and signed, which makes them tamper-proof and secure. Therefore, by using JWT tokens, Sinch can ensure the security and integrity of client interactions with its platform.

5.2.2 How to create the client

Initializing the Sinch client involves several steps, which are outlined below:

- Register for a Sinch account.
- Initialize the Sinch client: To initialize the Sinch client, we need to create a new instance of the SinchClient class and pass in your Sinch API key and secret. We will also need to implement the SinchClientListener interface to receive callbacks from the Sinch client.
- Specify a user ID: Once you have initialized the Sinch client, we will need to specify a user ID for the client. This can be done by calling the setUserId() method on the Sinch client instance.
- Start the Sinch client: Once we have specified a user ID, we can start the Sinch client by calling the start() method on the client instance. This will establish a connection to the Sinch servers and allows to begin using Sinch's communication services.
- Handle errors and exceptions: It is important to handle any errors or exceptions that may occur during the use of the Sinch SDK. This can be done by implementing the appropriate error handling methods provided by the Sinch SDK.

5.3 Making outgoing call

To make a call to a user using Sinch on iOS, the first step is to set up a Sinch client instance and connect it to the Sinch servers. The client instance is responsible for managing the call sessions, signaling, and media streams for the call. Once the client is connected, the app can initiate a call by specifying the recipient's identifier, such as their phone number or user ID.

To initiate a call, the app creates a SinchCall object and calls the client's callUserWithId method, passing in the recipient's identifier and a set of call options. The call options can include settings such as whether the call should be video or audio-only, whether it should be a one-to-one or conference call, and whether the call should be recorded.

Once the call request is sent to the Sinch servers, the recipient's device is notified and prompted to accept or decline the call. If the recipient accepts the call, the Sinch servers establish a peer-to-peer connection between the two devices, allowing them to exchange audio and video streams directly.

During the call, the app can use the SinchCall object to monitor the call's state and control various aspects of the call, such as muting or holding the call. When the call ends, the app can use the SinchCall object to clean up the call session and release any resources used by the call.

5.4 Receiving a call

5.4.1 What is VOIP pushes?

In iOS, VOIP (Voice over Internet Protocol) pushes are a type of remote notification that allows an application to establish a network connection and receive real-time communication, such as voice or video calls, even when the application is in the background or not running. VOIP pushes are an essential component of many communication apps, enabling users to receive incoming calls and messages without having to keep the app open or actively running on their device.

When an app receives a VOIP push, it is woken up in the background, and the system launches the app to establish a network connection with the server. Once the app has established the connection, it can receive and handle the incoming call or message. VOIP pushes use the Apple Push Notification Service (APNS) to deliver the notification to the user's device.

The process of sending VOIP pushes involves several steps. First, the app must register with the APNS for remote notifications and enable VOIP push notifications. The app must also configure its network socket and start listening for incoming connections. When a VOIP push is received, the system wakes up the app in the background and launches it. The app then uses the network socket to establish a connection with the server and receive the incoming call or message.

The use of VOIP pushes in iOS requires special considerations to optimize the user experience and ensure efficient use of resources. For example, developers must carefully manage the use of network sockets and minimize the amount of data sent and received over the network to avoid excessive battery drain. They must also consider the impact of incoming calls and messages on the user's device and provide appropriate notification and alert mechanisms.

5.4.2 How Sinch send VOIP pushes in iOS

One of the critical features that Sinch offers is the ability to send VOIP pushes in iOS. VOIP (Voice over Internet Protocol) pushes are a type of remote notification that wakes up an application and enables it to establish a network connection for real-time communication.

To send VOIP pushes in iOS using Sinch, the first step is to configure the application's push notification settings. This involves obtaining the necessary certificates and credentials from the Apple Developer Portal and configuring the application to register for push notifications using the Sinch SDK. Once the push notification settings are configured, the app can send VOIP pushes by calling the SinchClient's startListeningOnActiveConnection method. This method registers the app to receive incoming calls and VOIP pushes over the active network connection.

When a VOIP push is received, the app is woken up in the background, and the Sinch SDK establishes a network connection to the Sinch servers. The servers then initiate a call session, signaling the recipient's device to establish a peer-to-peer connection for the call.

To optimize the performance of VOIP pushes, Sinch provides a range of configuration options that developers can use to customize the behavior of the push notifications. For example, developers can set the maximum payload size for the push notification, configure the notification's priority level, and specify the expiration time for the notification.

5.4.3 How to receive VOIP pushes in iOS

Receiving VOIP (Voice over Internet Protocol) pushes in iOS involves configuring the app to receive and handle remote notifications from the Apple Push Notification Service (APNS) and setting up a network connection to the server to handle incoming calls or messages. Here are the steps to receive VOIP pushes in iOS:

1) Configure Push Notifications: The first step is to configure the app to receive remote notifications from APNS. This involves registering for remote notifications and enabling the "voip" option in the notification types. Developers must obtain the necessary certificates and credentials from the Apple Developer Portal to set up push notifications.

2) Set up Network Socket: The app must also set up a network socket and start listening for incoming connections. The app must use a special port number (port 5060) and the User Datagram Protocol (UDP) for handling incoming calls or messages. We can use the Sinch SDK or other VOIP providers to simplify this step.

3) Handle Incoming Notifications: When the app receives a VOIP push notification, the system wakes up the app in the background and launches it. The app can use the payload data of the notification to determine the type of incoming communication and establish a network connection to the server to handle the call or message. The app can also use the payload data to display appropriate notifications or alerts to the user.

4) Establish Network Connection: Once the app has received the notification, it must establish a network connection to the server to handle the incoming communication. The app can use the network socket set up in step 2 to establish the connection and exchange data with the server.

5) Handle Incoming Communication: After the network connection is established, the app can receive and handle the incoming communication. The app can use the data exchanged with the server to display appropriate UI elements and handle user interactions, such as answering or rejecting an incoming call.

Chapter 6

App Structure

6.1 User Flow

For start to use application user should follof next steps:

- 1) User opens the application and is prompted to create an account or log in.
- 2) User enters their credentials and logs in to the application.
- 3) Once logged in, the user is taken to the main screen of the application where they can see their contacts or call history.
- 4) User selects a contact they want to call and taps on the call icon.
- 5) The application prompts the user to select either a voice call or video call.
- 6) User selects the type of call they want to make and initiates the call.
- 7) The call is connected, and the user can see and hear the person on the other end.
- 8) During the call, the user has access to features like mute, speaker, and end call.
- 9) When the call is finished, the user ends the call and is taken back to the main screen of the application.

6.2 App Screens

6.2.1 Login Screen

Login Screen is one of the entry point of the application. We start to explore application from this screen when we use it first time or in case when we logged out. Here we can create the user.

6.2.2 Home Screen

When we are logged in we start the app every time from this screen. Here we can make the following actions:

- 1) Choose type of call.
- 2) Make a call to user with the username that we entered in text field.
- 3) See the list of all available users.

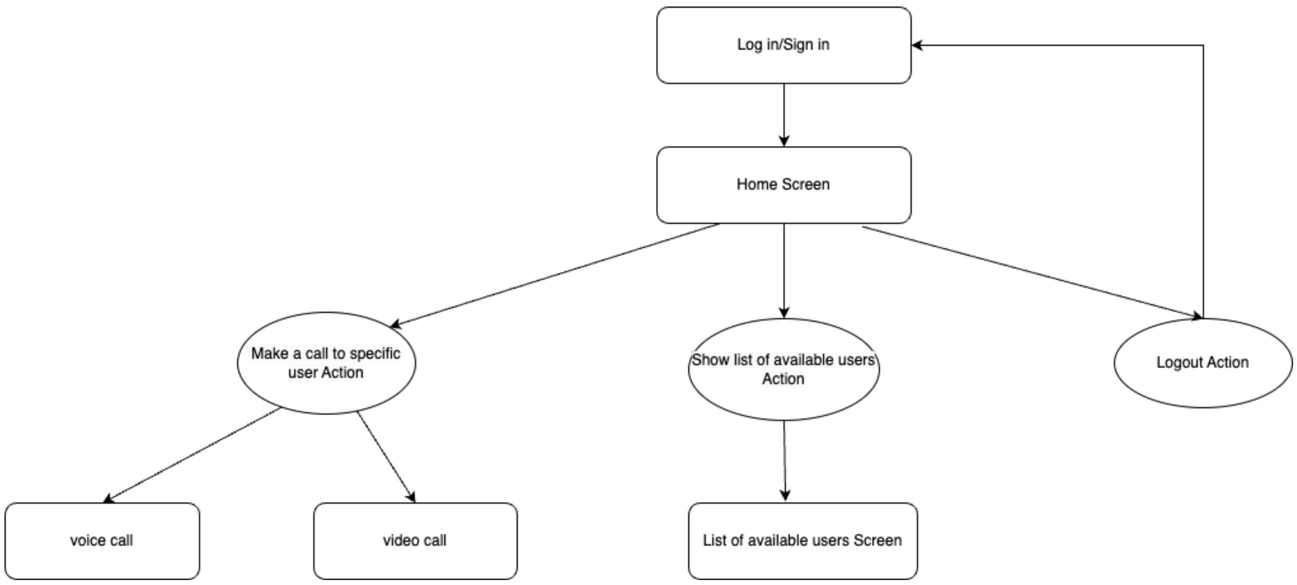
Log in/Sign in
Home Screen
Make a call to specific user Action
Show list of available users Action
Logout Action
voice call
video call
List of available users Screen

FIGURE 6.1: User Flow

- 4) Log out. In case when we to change user state to inactive, and don't receive any calls we can use logout for it. Than to use the app again we can log in with credentials.

6.2.3 Voice or video call Screens

When the call is incoming we can accept or decline the call. In case when the is outgoing we can cancel current call if we need it or just wait when call starts.

6.2.4 List of available users Screen

We can use action on Main screen to show the list of available users. There we will see all active users which we can call.