VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Chuong Tran

IOS CHAT APPLICATION

LetsChat

Technology and Communication
2021

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Chuong Tran
Title	iOS Chat Application
Year	2021
Language	English
Pages	57
Name of Supervisor	Smail Menani

This thesis aims to develop a messaging application for iOS devices, specifically for iPhone and iPad, named “LetsChat”. This application has requirements specification that other messaging applications in the market have: Login with Facebook and Google account; sending text messages; sending photo and video mes-sages

The technologies used to develop this application were Firebase, Swift and Facebook SDK.

Firebase was used as a Backend for storing all the data of the "LetsChat" application including user’s information, text messages, conversations, storing photos/video. Swift was used as a main programing language. Sing-in and authentication was implemented using Facebook SDK and Firebase.

The result of the thesis is a messaging application enabling users to send text and multi-media messages using mobile devices

This project was tested and prooved to work as required for the further development phase. This project can be deployed to the Appstore.

Keywords Swift, mobile application, messaging application, Firebase

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF CODE SNIPPET

1 INTRODUCTION

Mobile devices have a huge impact on human life in this technology age. Therefore, new technologies have evolved rapidly to make our life easier and more convenient. Millions of mobile applications have been developed to suffice our needs, and the chat application is one of the most essential that every mobile device should have. The users need a chat application which is easy to use, fast and secure.

The purpose of this thesis is to develop a mobile messaging application that users can connect with each other for the communication purpose. Users can create multiple conversations and connect with the others through the chat where they can text, and share media content, such as photos and videos.

This application was built using Swift and Firebase. iOS is a mobile operating system exclusively created and developed by Apple Inc. for its mobile devices and promising to thrive day by day; Swift is a programming language that fast, new, and easy to learn with less clunky syntax to use for developing an iOS application. The database of this application is stored on Firebase which is cloud-service and backend development platform. With Firebase the data will be more secured and increasing performance for the application.

The result of this thesis is an application named “LetsChat” with specific functions that meets the basic needs of a messaging application with an accessible easy-to-use user interface and more importantly users and devices should be communicating with each other through the chat.

This application has features which can only be used on mobile devices and spectacularly iOS operating system only. “LetsChat” limitation is only to focus on developing for sending messages but not for communicating by other methods.

2 USED TECHNOLOGIES

This section introduces all the technologies that are used in this application.

2.1 Xcode Integrated Development Environment

Xcode is free of charge integrated development environment (IDE) which was established by Apple Inc. for the special use of developing software for iOS, iPadOS, macOS, and other Apple products. Xcode can support a lot of programming lan-guages, such as C, C++, Java, Python, and especially Objective-C, AppleScript, and Swift which are the main programming/scripting languages that are supported by Apple for its devices. /1/

2.2 Swift Programming Languages

Swift is a new powerful and intuitive programming language that was developed as a replacement for Objective-C - an earlier Apple programming language; Swift is a general-purpose, multi-diagram, and compiled programming language. /2/

Swift is interaction, safe and produces fast running application, with concise and expressive syntax.

2.3 CocoaPods

CocoaPods is an application-level dependency manager for Swift and Objective-C-which has over 82 thousand libraries till today.

The main purpose of CocoaPods is to specify all the dependencies of the project in a text file called Podfile and resolve them between libraries, fetch the resulting source code then link it together in the Xcode workspace to build the projectCocoaPods helps to incorporate third party dependencies, libraries and framework into the project without worrying about setting up, configure and versions of used libraries. /3/

2.4 Google Firebase

Firebase is a web and mobile applications development platform provided by Google. The entire platform is a Back-end-as-a-service (BaaS) which means the application can link to back-end cloud storage and APIs provided by backend applications. Firebase is really powerful with its provided services. /4/ /5/

2.4.1 Firebase Authentication

Firebase Authentication is a service to authenticate users of the application with comprehensive security, fast implementation and easy-to-use. It supports the use of passwords, email, phone number and popular federated identity providers (for example, Facebook, Google, and Twitter). \6\

2.4.2 Firebase Realtime Database

Firebase Realtime Database is a cloud-hosted non-relational database which stores data as JSON and sync data between users in real-time. Firebase Realtime Database helps to build an application without the need of server with strong security and make it easy for users to access data. \7\

2.4.3 Firebase Storage

Firebase Storage is the service built in on the Firebase platform with the function of storing and managing user contents and media, such as photos, video, or file data. Firebase Storage also provides Google security for safe upload and download files from the application regardless of network quality. \8\

2.5 Facebook Login SDK

Facebook Login is a service of Facebook SDKs that enables users to sign into an application with a Facebook account and authentication with credential. With the Facebook login the user can also permit the application to access user data such as name and email, in a fast and secure way. \9\

2.6 Google Sign-in SDK

Google Sign-In is a secure authentication system which has related solutions with Firebase Authentication that allow users to sign into an application with a Google account, thereby reducing the burden of sign-in method. Google Sign-In also let application access to basic user information, such as name, and email \10\

3 APPLICATION DESCRIPTION AND DESIGN

This chapter contains a detailed description of the application. The application requirements and specifications are analyzed.

3.1 Requirement Specifications

Table 1 shows the requirements specifications of the application.

Table 1. Requirement Specification of Application

References	Description	Priority
F1	Implement Register	1
F2	Implement Login	1
F3	Create Conversation	1
F4	Send Text	1
F5	Send Photo	2
F6	Send Video	2
F7	Send Location	3
F8	Send File	3
F9	Group Message	3

Priority Appendix:

1. Must-have function
2. Should-have function
3. Interesting-to-have function

3.2 Analysis

The functions were implemented by order of priority as following:

- Register: A new user needs to register into the account in order to login to the application. The register phase collects the user information, such asfirst name, last name, profile picture, password and email.
- Login: Users can login with an existing account that is registered or using a Facebook or Google account. If the user logs in with a Facebook or Google account, the user information, such as first name, last name, profile picture and email also be collected.
- Create Conversation: Users create a conversation with others and start messaging.
- Send text: Users are able to send text content messages for in created conversation.
- Send Photo/Video: Users are able to send media content messages to others, such as Photo and Video messages.

All the data of the application should be stored on Firebase.

3.3 Use Case Diagram

The use case diagrams of the application are presented next.

3.3.1 Authentication Use Case Diagram

The authentication use case diagram is shown in Figure 1. As shown in Figure 1, the user has three methods to login into the application which are log in with email/password; log in with a Facebook account, and log in with a Google account.

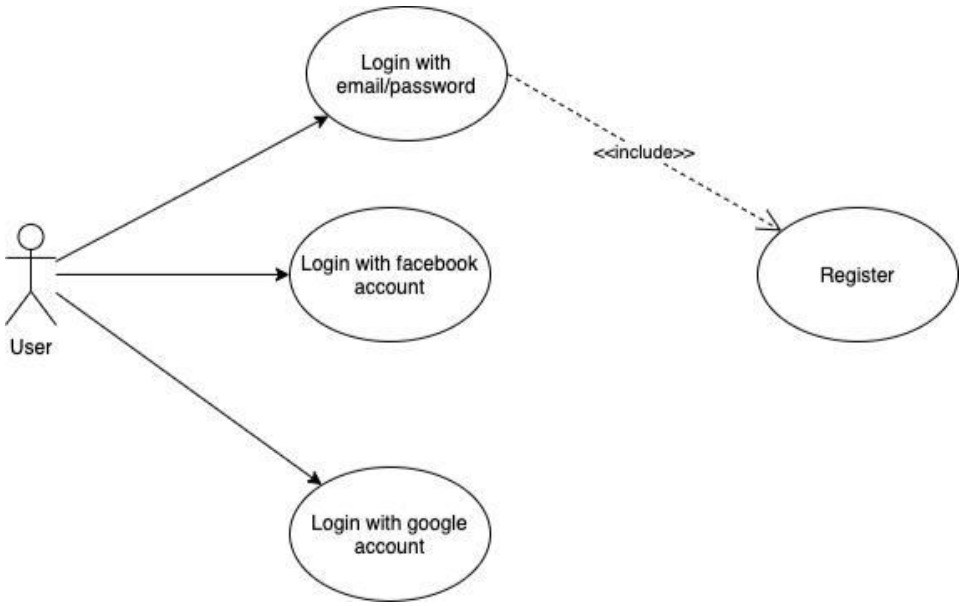


Figure 1. Authentication Use Case Diagram

If users want to log in with their email/password out of any using any third-party method, they have to register the information in which they can provide basic information such as first name, last name, password, email, and optional profile picture.

3.3.2 Send Message Use Case Diagram

Send Message user case diagram is given in Figure 2. It can be seen in Figure 2 that users can send a message to another user after logging into the application by first creating a conversation. After that users can communicate with each other through the conversation that have been created. They can send a text message, send a photo message or video message or even delete the conversation and all of this requires users to first pass the authentication.

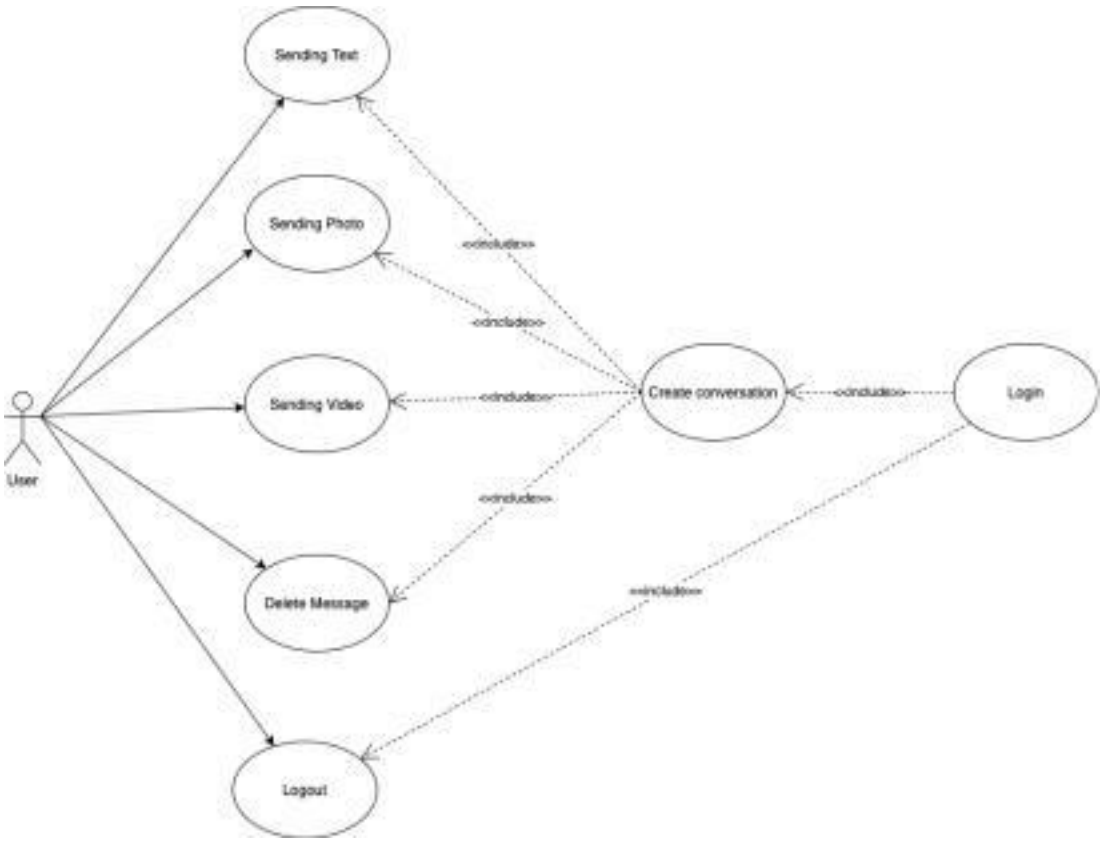


Figure 2. Send Message Use Case Diagram

3.4 Sequence Diagram

The sequence diagrams are presented in this section.

3.4.1 Authentication Sequence Diagram

Figure 3 shows the sequence diagram of the authentication section, the process of the user logging into the application. The user will first need to provide an email and password to be authenticated. If the given email and password pass the authentication, the application will redirect to the home page. If not, the error message pop-up.

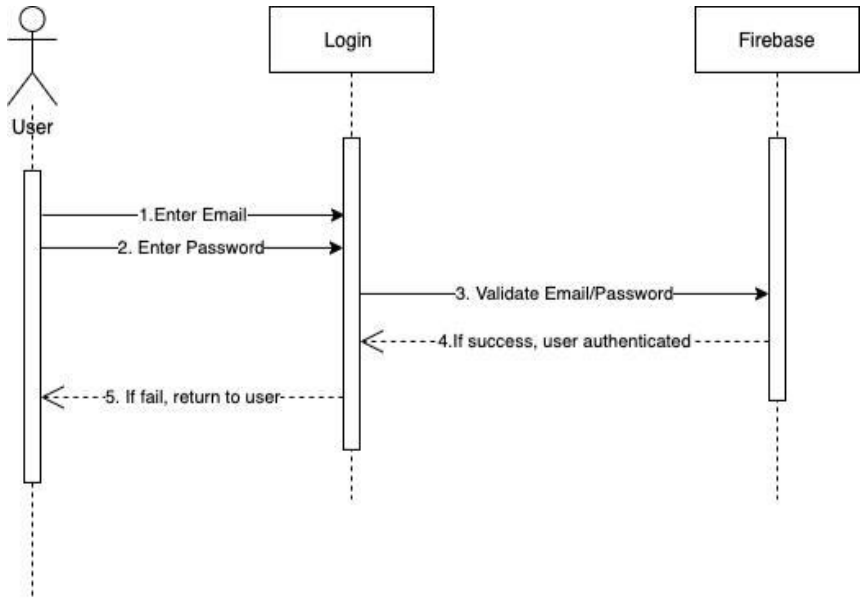


Figure 3. Authentication Sequence Diagram

3.4.2 Send Message Sequence Diagram

Figure 4 demonstrates the process of sending messages between users. When the user wants to start a conversation, firstly the user needs to create a new conversation, then send a message which will be saved to the database. After that users can read the message by getting the message from Firebase.

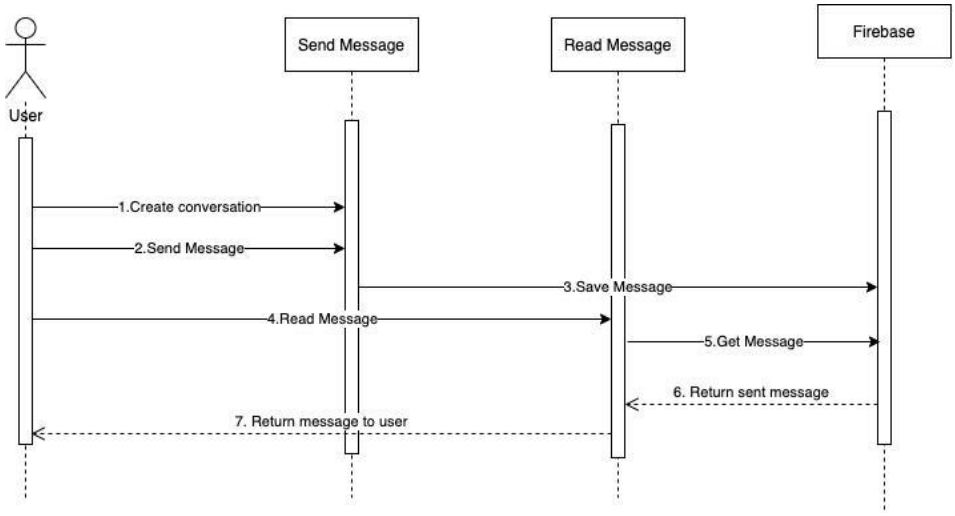


Figure 4. Send Message Sequence Diagram

4 DATABASE AND GUI DESIGN

4.1 Database

Firebase is a non-relational database using a storage model to store data, which means that Firebase does not use tables, rows, primary keys and foreign keys. In this project, the data will be stored in the JSON format. The database designed as shown in Figures 5 and 6 below.

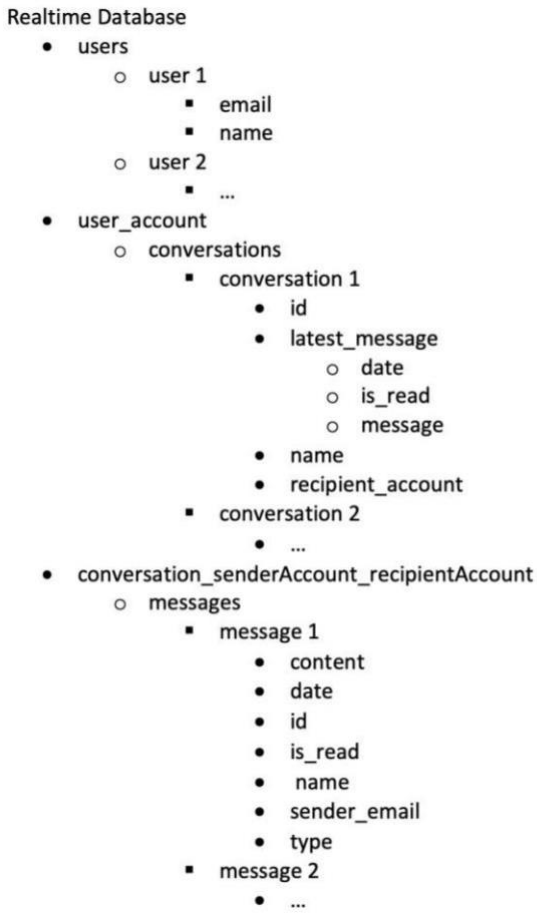


Figure 5. Realtime Database Design

Database included:

- user: stores users account information.

- user-email: the user can have multiple conversations, so this field will store the metadata of each conversation.
- conversation-senderEmail-recipientEmail: stores all the messages between sender and recipient.

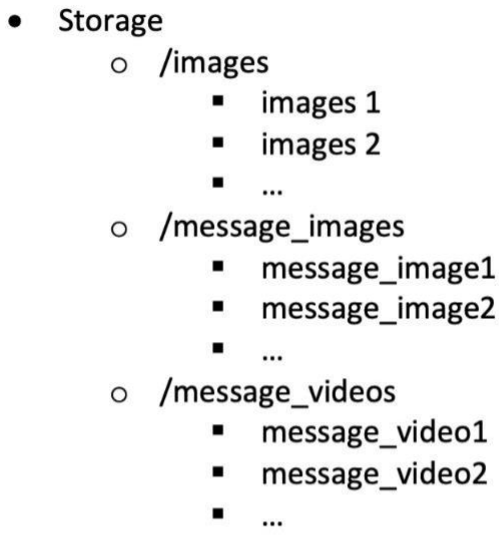


Figure 6. Storage Design

Storage included:

- /images: stores all the profiles pictures.
- /message_images: stores all the images related to the conversation.
- /message_videos: stores all the videos related to the conversation .

4.1.1 Realtime Database

Figure 7 shows all the users that are registered to the application, including the users that use a Facebook and Google account, stored in the Realtime Database of Firebase. The user data is stored under the user dictionary with contains properties which are email and name.

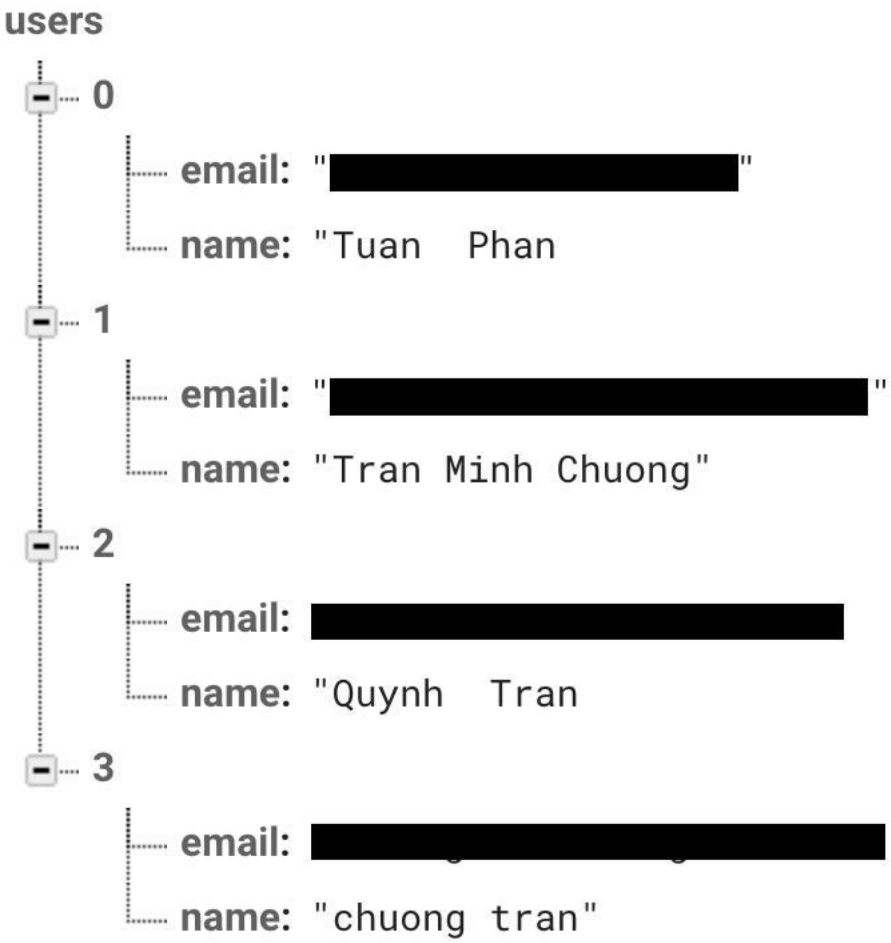


Figure 7. Users Collection in Realtime Database

Figure 8 shows the data of each user saved in Realtime Database having values, such as conversations that the user has, first name and last name of the user. The conversations contain properties that are id, latest message, name of the re-cipient and the email of recipient.

In the latest message table saves the date of the latest message, and the mes-sage.

Figure 8. User Data Save in Realtime Database

Figure 9. Conversation Between Users Save in Realtime Database

Figure 9 shows the conversation between users, which has the key value of conversation Id and divided into messages the properties of which are the content of the message, date of the message sent, id, name, sender email, and type of the message. Code Snippet 1 shows how the conversation Id generated to save in Realtime Database.

```
let conversationId = "conversation_\(firstMessage.messageId)"
```

Code Snippet 1. Conversation Id Generator

As indicated in Code Snippet 2, to store users email on Firebase, all the special character including in email address, such as “@” and “.” should be replaced with “-”.

```
var safeEmail: String {
    var safeEmail = emailAddress.replacingOccurrences(of: ".", with: "-")
    safeEmail = safeEmail.replacingOccurrences(of: "@", with: "-")
    return safeEmail
}
```

Code Snippet 2. Safe Email Generator

4.1.2 Storage

The purpose of Firebase Storage is to store message media content created by users, such as profile picture store in the images folder, photo message store in-message_images folder, and video message stored in message_videos.

Figure 10. Firebase Storage

```
let fileName = "photo_message_" + messageId.replacingOccurrences(of: " ", with: "-") + ".png"
```

Code Snippet 3. File Name Generator

Code Snippet 3 shows how the file name of photo messages are generated to store in Storage, as well as profile picture and video messages.

4.2 GUI Design

In this section, all the GUI Design explained through the capture of iPhone 11 simulator which is provided by Xcode IDE. The application works with both dark mode and light mode.

Figure 11. Launch screen page

The first page pop-up whenever the application started is the launch screen page, which shows the logo of the chat application. The other reason for it is that when the application is loading in the background and users have to wait, they would not want to see a blank white or black screen. This page would also have a copyright.

Login Page contains two fields that require users to be logged i to the application, Email Address and Password with a login button and other login methods that the application provides. A register button takes the user to the register page.

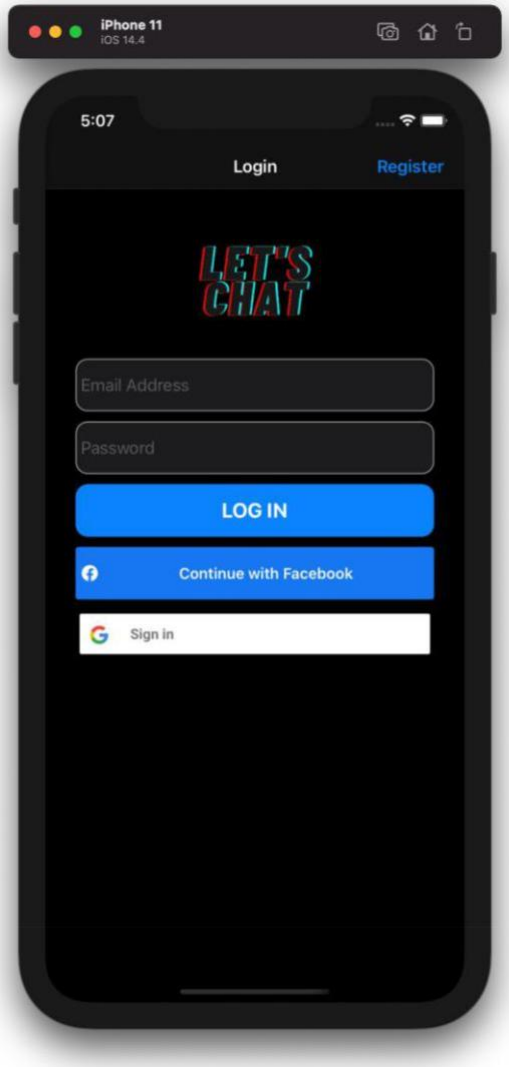


Figure 12. Login Page

Figure 13 shows the Register Page that contain all the field needed to create a new account including First Name, Last Name, Email Address and Password.

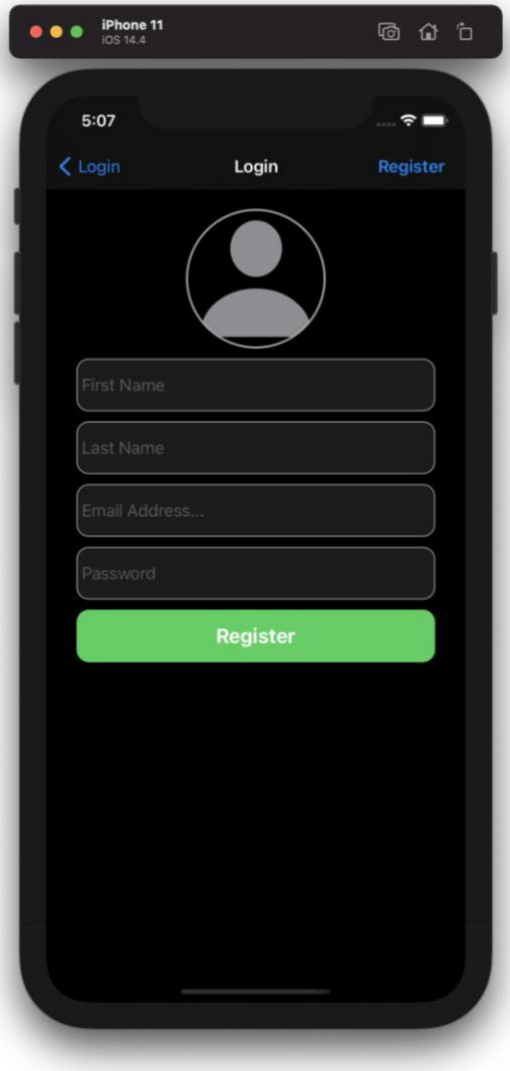


Figure 13. Register Page

The Main Page of the application that is shown in Figure 14 is where all the conversations appear. This page has a navigation bar with title “Chats” and two navigation buttons at the tab bar below to switch between the pages.

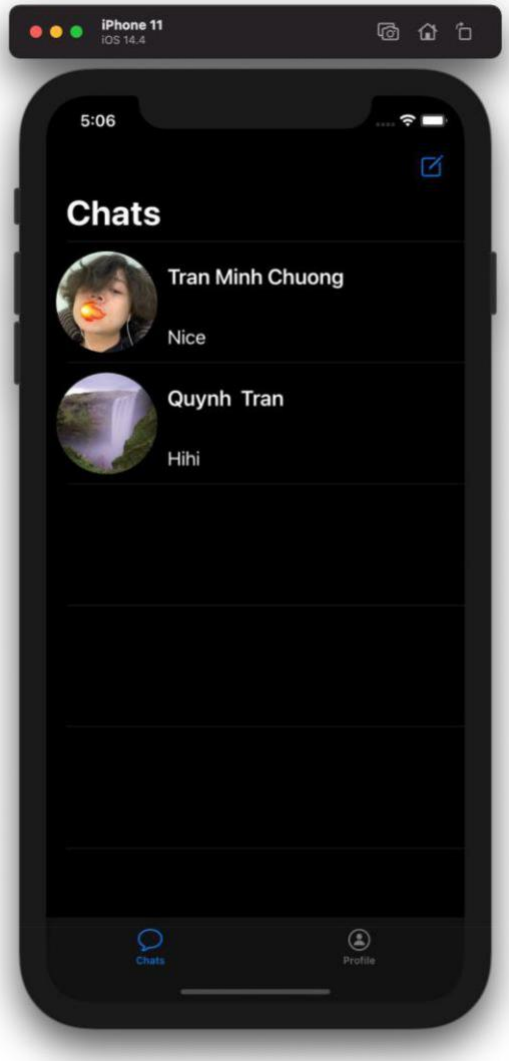


Figure 14. Main Page

The Profile Page that is demonstrated in Figure 15 will show the user profile picture along with the user name, and email. The “Log Out” button will appear on an action sheet that lets the user confirm the log out from the account.

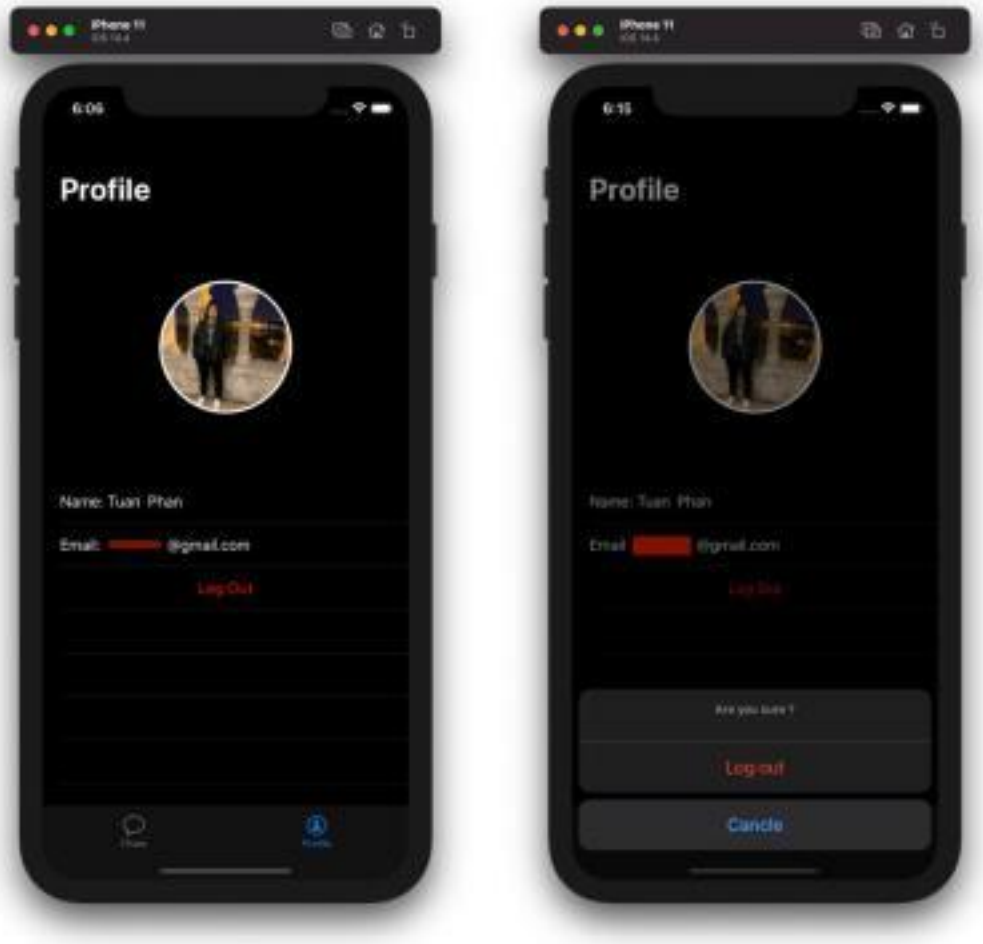


Figure 15. Profile Page

Figure 16 is a capture of the chat screen. With help of MessageKit library the Chat UI of the application is easily created and customizable with given components provided.

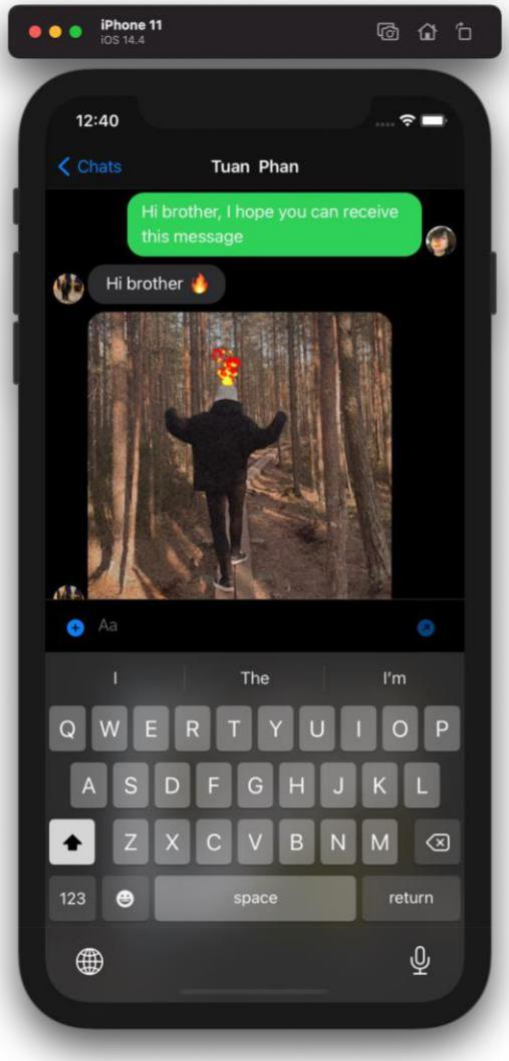


Figure 16. Chat Screen

Figure 17 shows how the application navigation is set up and the relationship between pages

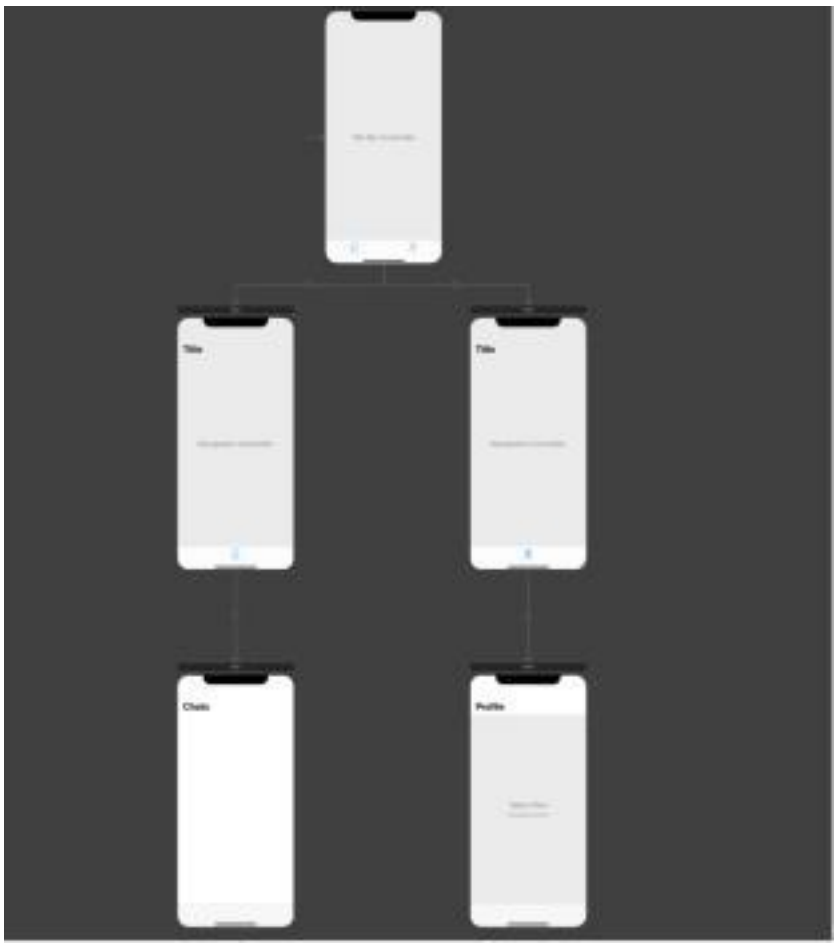


Figure 17. Application Navigation Set Up

5 IMPLEMENTATION

This section demonstrates the implementation of the application, some parts of the source code will be explained.

5.1 Project Structure

Figure 18 below shows the project structure divided to directories

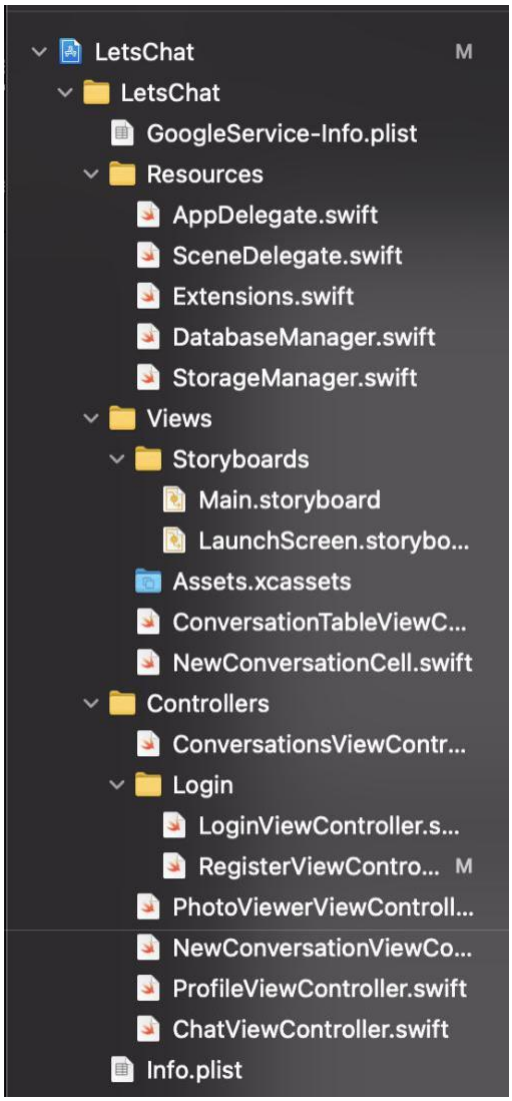


Figure 18. Project Structure

The structure of the directories is as follows:

- LetsChat is the root folder of the application.
- Resources directory is where the database and connection between the application and Firebase is handled.
- View directory is where the graphical user interface is designed.
- Controllers directory is where all functionalities are implemented.
- All the logo images and it relevant are stored in Assets.xcassets

5.2 Database Connection and Dependencies

The application used Firebase as its database. To connect with the Firebase, the *GoogleService-Info.plist* was provided in the set-up phase of the project on Firebase.

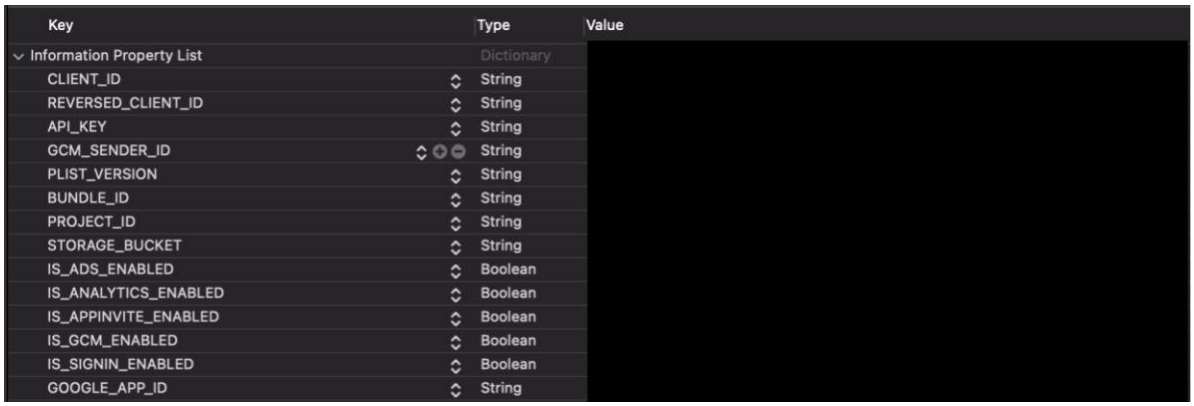
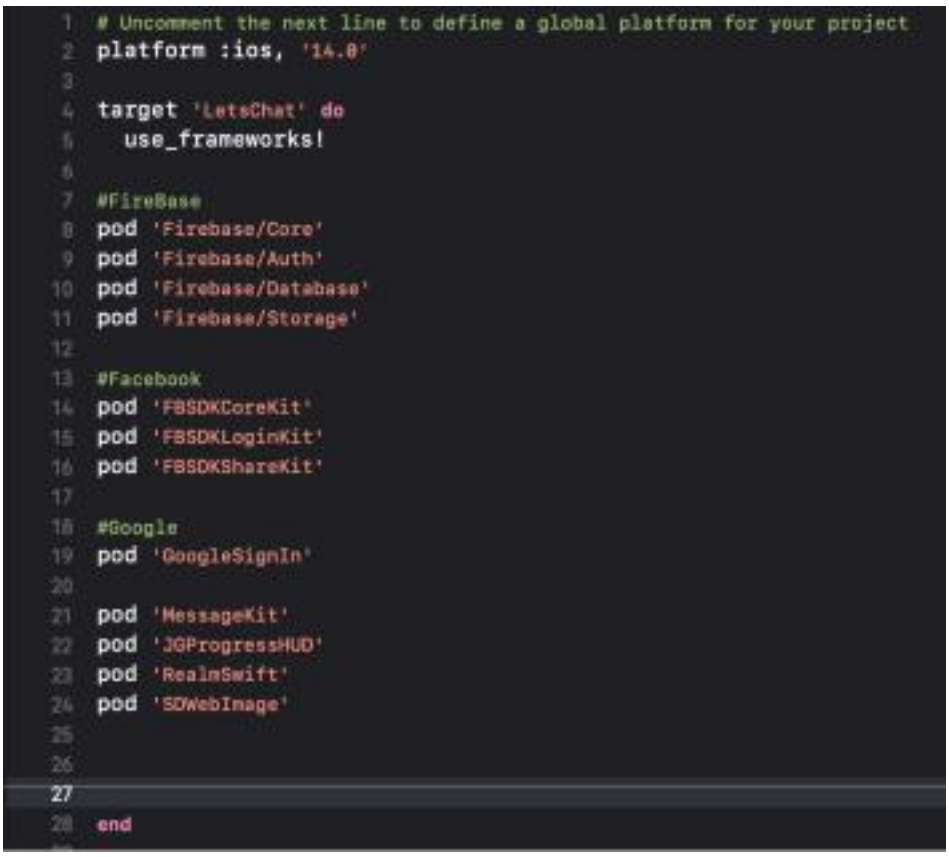


Figure 19. GoogleService-Info.plist

Figure 19 shows, the *GoogleService-Info.plist* implementation with all necessary properties for the application configuration to connect with Firebase services.

Figure 20 demonstrates that *Podfile* contains all the dependencies/libraries that were implemented in the project and the target build of the project is for iOS 14 and bellow.

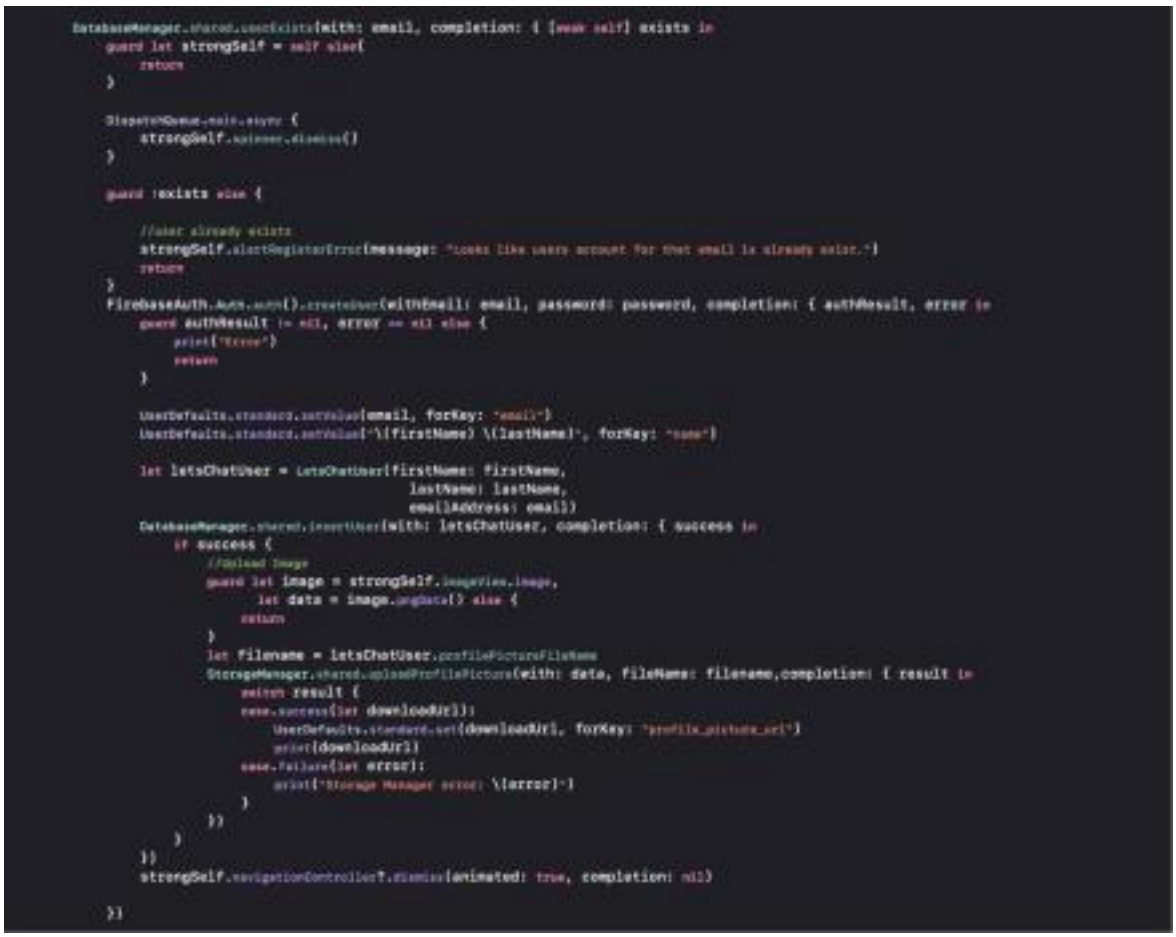


```
# Uncomment the next line to define a global platform for your project
platform :ios, '14.0'

target 'LetsChat' do
  use_frameworks!

#FireBase
pod 'Firebase/Core'
pod 'Firebase/Auth'
pod 'Firebase/Database'
pod 'Firebase/Storage'

#Facebook
pod 'FBSDKCoreKit'
pod 'FBSDKLoginKit'
pod 'FBSDKShareKit'

#Google
pod 'GoogleSignIn'

pod 'MessageKit'
pod 'JGProgressHUD'
pod 'RealmSwift'
pod 'SDWebImage'


end
```

Figure 20. Podfile

5.3 Implementation of Register

Code snippet 4 shows the implementation of the register. First name, last name, email, and password will be read from the text fields component. With the password, the application will check the length of the password. The password could not be shorter than 6 characters. All these fields are required, and cannot be empty, otherwise, the application will alert an error message. After filling in all the required fields, the application will register an account with *Firebase.Auth. auth().createUser* function provided by Firebase SDK. If the registering successful, the user will be saved in Firebase Realtime Database and immediately log in to the application, otherwise the application will return an error message to the user.



Code Snippet 4. Register implementation

5.4 Login Implementation

The user can login with multiple credentials. In this application the authentication is implemented with email/password, Google account, and Facebook account.

The email and password are read from text fields and handled as shown in the Code Snippet 5 below

```
FirebaseAuth.Auth.auth().signIn(withEmail: email, password: password, completion: { [weak self] authResult,error in
    guard let strongSelf = self else{
        return
    }
    DispatchQueue.main.async {
        strongSelf.spinner.dismiss()
    }

    guard let result = authResult, error == nil else{
        self?.alertLoginError()
        return
    }
    print ("\(result)")

    let safeEmail = DatabaseManager.safeEmail(emailAddress: email)
    DatabaseManager.shared.getDataFor(path: safeEmail, completion: { result in
        switch result {
        case .success(let data):
            guard let userData = data as? [String:Any],
            let firstName = userData["first_name"] as? String,
            let lastName = userData["last_name"] as? String else{
                return
            }
            UserDefaults.standard.set("\(firstName) \(lastName)", forKey: "name")

        case .failure(let error):
            print("Failed to read data with error\(error)")
        }
    })

    UserDefaults.standard.set(email, forKey: "email")

    strongSelf.navigationController?.dismiss(animated: true, completion: nil)
})
```

Code Snippet 5. Login with Email/Password

The email and password will be authenticated by *Firebase.Auth.auth().signIn* function provided by the Firebase platform. Firebase will check whether the email and password match with any registered accounts. If matched, the user will be authenticated, otherwise the application will return an error message to the user.

The user can also login with Google and Facebook credentials by connecting with their platforms and the authentication is implemented similarly with using the Email/Password method.

5.5 Creating Conversation Implementation

In order to send a message, the user needs to create a conversation. The Code Snippet 6 shows how the create conversation function was implemented.

```
private func createNewConversation(result: SearchResult){
    let name = result.name
    let email = DatabaseManager.safeEmail(emailAddress: result.email)
    DatabaseManager.shared.conversationExists(with: email, completion: {[weak self] result in
        guard let strongSelf = self else {
            return
        }
        switch result {
        case .success(let conversationId):
            let vc = ChatViewController(with: email, id: conversationId)
            vc.isNewConversation = false
            vc.title = name
            vc.navigationItem.largeTitleDisplayMode = .never
            strongSelf.navigationController?.pushViewController(vc, animated: true)
        case .failure(_):
            let vc = ChatViewController(with: email, id: nil)
            vc.isNewConversation = true
            vc.title = name
            vc.navigationItem.largeTitleDisplayMode = .never
            strongSelf.navigationController?.pushViewController(vc, animated: true)
        }
    })
}
```

Code Snippet 6. Create conversation implementation

This function will check whether the conversation exists or not. If it exists, the user can append the existing conversation, otherwise a new one is created. After having the conversation, the application will navigate to the chat page.

5.6 Send Message Implementation

Each message will follow the message structure with required information. The code snippet below shows the message object definition.

Each message requires unique id to identify the message. The type of message can be text, attribute text, photo, or video. The contents of message are what the user enters to the chat application. The message also saves the date, sender email, and name.

```
let newMessageEntry: [String: Any] = [
    "id": newMessage.messageId,
    "type": newMessage.kind.messageKindString,
    "content": message,
    "date": dateString,
    "sender_email": currentUserEmail,
    "is_read": false,
    "name": name
]
```

Code Snippet 7. Message object

After having the message with all the necessary information, the message will be saved to the Firebase database using the function shown in Code snippet 8 be-low.

As shown in Code Snippet 8 if the user already is in the conversation, the mes-sage will be added to the conversation and the position of the message will be increased by each message. With the position of the message the application can know the order of messages.

```
if var currentUserConversations = snapshot.value as? [[String: Any]] {
    var targetConversation: [String: Any]?
    var position = 0

    for conversationDictionary in currentUserConversations {
        if let currentId = conversationDictionary["id"] as? String, currentId == conversation {
            targetConversation = conversationDictionary
            break
        }
        position += 1
    }

    if var targetConversation = targetConversation{
        targetConversation["latest_message"] = updatedValue
        currentUserConversations[position] = targetConversation
        databaseEntryConversations = currentUserConversations
    }
    else{
        let newConversationData: [String: Any] = [
            "id": conversation,
            "other_user_email": DatabaseManager.safeEmail(emailAddress: otherUserEmail),
            "name": name,
            "latest_message": updatedValue
        ]
       databaseEntryConversations = [
                newConversationData
       ]
        currentUserConversations.append(newConversationData)
        databaseEntryConversations = currentUserConversations
    }
}
```

Code Snippet 8. Save messages to current conversation

If the conversation does not exist, the new conversation will be created, and the message will be added to the new one, as shown in Code Snippet 9 below.

```
        else{
            let newConversationData: [String: Any] = [
                "id": conversation,
                "other_user_email": DatabaseManager.safeEmail(emailAddress: otherUserEmail),
                "name": name,
                "latest_message": updatedValue
            ]
           databaseEntryConversations = [
                    newConversationData
           ]
        }
        strongSelf.database.child("\(currentEmail)/conversations").setValue(databaseEntryConversations, withCompletionBlock: {
            error, _ in
            guard error == nil else {
                completion(false)
                return
            }
```

Code Snippet 9. Save message to new conversation

5.7 Get All Conversation Implementation

This function fetches all the conversations of the current user from database.

```
public func getAllConversations(for email: String, completion: @escaping(Result<[Conversation], Error>) -> Void){
    database.child("\(email)/conversations").observe(.value, with: {snapshot in
        guard let value = snapshot.value as? [[String:Any]] else{
            completion(.failure(DatabaseError.failedToFetch))
            return
        }

        let conversations: [Conversation] = value.compactMap({ dictionary in
            guard let conversationID = dictionary["id"] as? String,
                  let name = dictionary ["name"] as? String,
                  let otherUserEmail = dictionary ["other_user_email"] as? String,
                  let latestMessage = dictionary ["latest_message"] as? [String: Any],
                  let date = latestMessage["date"] as? String,
                  let message = latestMessage["message"] as? String,
                  let isRead = latestMessage["is_read"] as? Bool else {
                return nil
            }

            let latestMessageObject = LatestMessage(date: date,
                                                    text: message,
                                                    isRead: isRead)
            return Conversation(id: conversationID,
                                name: name,
                                otherUserEmail: otherUserEmail,
                                latestMessage: latestMessageObject)

        })

        completion(.success(conversations))

    })
}
```

Code Snippet 10. Get all messages implementation

Code Snippet 10 demonstrates how all conversations can be fetched from the database for the current user, each conversation contains the id of the conversation, name, recipient user email, latest message that been sent, date that message been sent and content of the message.

5.8 Get All Messages for Conversation Implementation

This function fetches messages from the conversation id. Code Snippet 11 shows that all the messages are fetched by the conversation id, the function imple-mented with Firebase SDK which is *database.child().observe().* Each message has the properties of name, message id, content of the message, sender email, type of the message and date.

```
public func getAllMessagesForConversation(with id: String, completion: @escaping(Result<[Message], Error>) -> Void){
    database.child("\(id)/messages").observe(.value, with: {snapshot in
        guard let value = snapshot.value as? [[String:Any]] else{
            completion(.failure(DatabaseError.failedToFetch))
            return
        }
        let messages: [Message] = value.compactMap({ dictionary in
            guard let name = dictionary["name"] as? String,
                  let messageID = dictionary["id"] as? String,
                  let content = dictionary["content"] as? String,
                  let senderEmail = dictionary["sender_email"] as? String,
                  let type = dictionary["type"] as? String,
                  let dateString = dictionary["date"] as? String,
                  let date = ChatViewController.dateFormatter.date(from: dateString)else {
                return nil
            }
```

Code Snippet 11. Get all message for a conversation

5.9 Delete Conversation Implementation

Code Snippet 12 below shows how the conversation be deleted. The conversation will be deleted by the conversation id. This function gets all conversation for current user and delete conversation with target id. After that the conversations will be reset their position.

```
public func deleteConversation(conversationId: String, completion: @escaping (Bool) -> Void){
    guard let email = UserDefaults.standard.value(forKey: "email") as? String else {
        return
    }

    let safeEmail = DatabaseManager.safeEmail(emailAddress: email)
    print("Deleting conversaiton with id\(conversationId)")

    //Get all conversations for current user
    //delete conversations in collection with target id
    //reset those conversations for user in database
    let ref = database.child(("\(safeEmail)/conversations"))
    ref.observeSingleEvent(of: .value){ snapshot in
        if var conversations = snapshot.value as? [[String: Any]]{
            var positionToRemove = 0
            for conversation in conversations {
                if let id = conversation["id"] as? String,
                   id == conversationId{
                    print("found conversation to delete")
                    break
                }
                positionToRemove += 1
            }
            conversations.remove(at: positionToRemove)
            ref.setValue(conversations, withCompletionBlock: { error, _ in
                guard error == nil else{
                    completion(false)
                    print("failed to write new conversations array")
                    return
                }
                print("deleted")
                completion(true)
            })
        }
    }
}
```

Code Snippet 12. Delete conversation Implementation

5.10 Upload Media to Firebase Storage Implementation

The media contents messages are stored in Firebase Storage.

```
public func uploadMessagePhoto(with data: Data, fileName: String, completion: @escaping UploadPictureCompletion){
    storage.child("message_images/\(fileName)").putData(data, metadata: nil, completion: { [weak self] metadata, error in
        guard error == nil else {
            //Failed
            print("Failed to upload data to firebase picture")
            completion(.failure(StorageErrors.failedToUpload))
            return
        }
        self?.storage.child("message_images/\(fileName)").downloadURL(completion: { url, error in
            guard let url = url else {
                print ("Failed to get download URL")
                completion(.failure(StorageErrors.failedToGetDownloadUrl))
                return
            }
            let urlString = url.absoluteString
            print("Download URL returned: \(urlString)")
            completion(.success(urlString))
        })
    })
}
```

Code Snippet 13. Upload Photo Message

Code Snippet 13 indicates that how photos are uploaded to the Firebase Storage. The photo message will be uploaded to be stored with the function implemented with Firebase SDK which is *storage.child().putdata().*The photo URL is generated to download by the function *storage.child().downloadURL*. Video messages and profile pictures were also implemented in the same way.

6 TESTING

This section describes the testing of the application. The results will be analyzed, then it is discussed if the application gives the result as expected.

The testing phase was performed with Quality Control (QC) testing which verifies the product's compliance with functional requirements.

The tests can be performed using two iOS simulators that Xcode provided such as iPhone 12 Pro Max (light mode), iPhone 11 Pro Max (dark mode) and physical device: iPhone XS Max. The test was performed in following functions:

- Register new account: Users are able to register a new account.
- Login: Users are able to login with Facebook, Google or registered account.
- Search recipient user and create conversation: The user be able to search for another user and create a new conversation.
- Sending text: Users are able to send text message content to the recipient and the recipient is able to read the messages
- Sending video/photo: Users are able to send a video/photo to the recipient user and the recipient is able to see the photo and play the video.
- Delete conversation: Users are able to delete conversation.

6.1 Registering a New account

The register section is tested as follows:

- Enter all the field that meets the requirements with a profile picture from Photo Library.
- Enter the existing email address.
- Enter the password that do not meet the requirements.

The expected result:

- Registered account/ user data saved to Firebase Realtime Database.
- Registered account data saved to Firebase Authentication.
- Alert will appear if any of the failure caught in the process.
- User profile picture stored in Firebase Storage.

The result described in pictures:

Try to get rid of this white space, change the order of the pictures, if necessary, place a smaller photo here to fill the page.

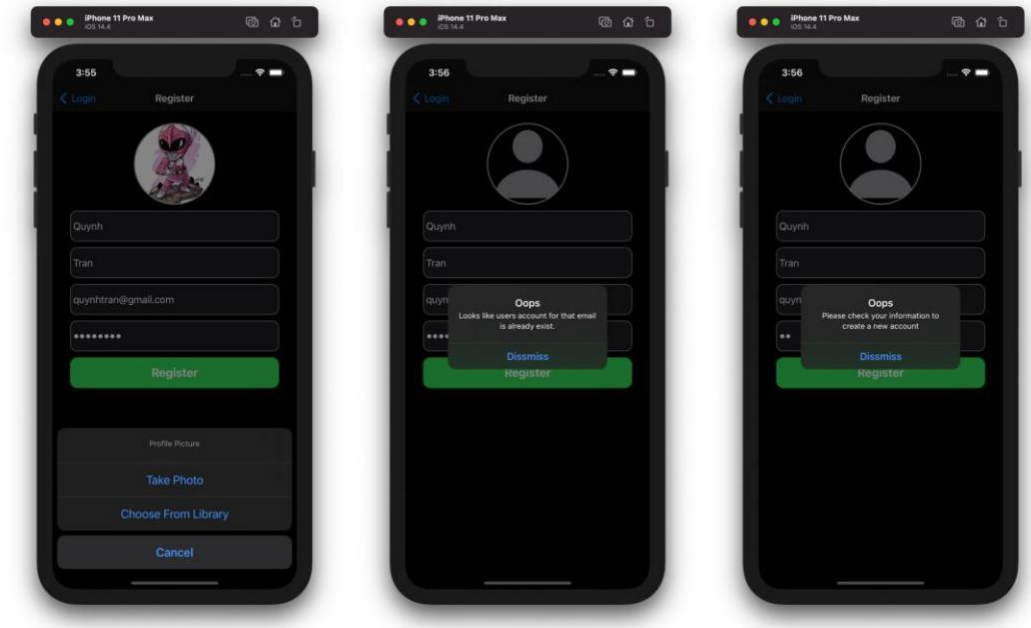


Figure 21. Register Account Process

Figure 22. Firebase Authentication with Registered User.

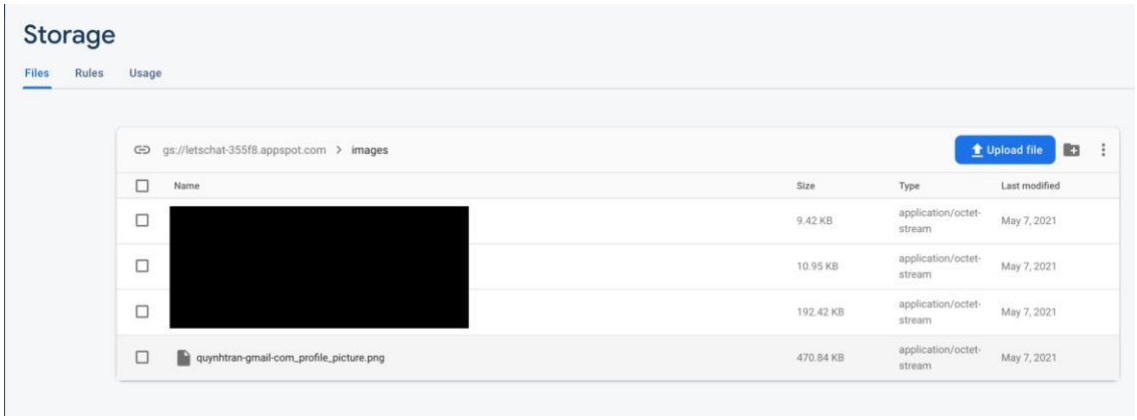


Figure 23. Firebase Storage with The Profile Picture.

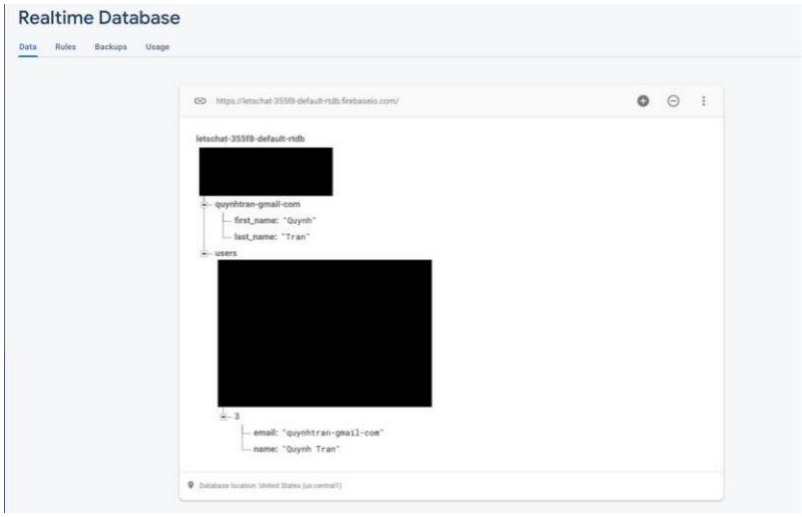


Figure 24. Realtime Database Saved User Data.

6.2 Login with Registered Account

The login section was tested as follows:

- Enter a wrong password/username.
- Enter the registered account that already exist on Firebase.

The expected result:

- The user can log in to the application.
- On the Profile page, the user information, such asProfile picture, name and email can be fetched from Firebase Storage and Firebase Realtime Database.
- The alert will appear if the log-in fails.

The results described in pictures:

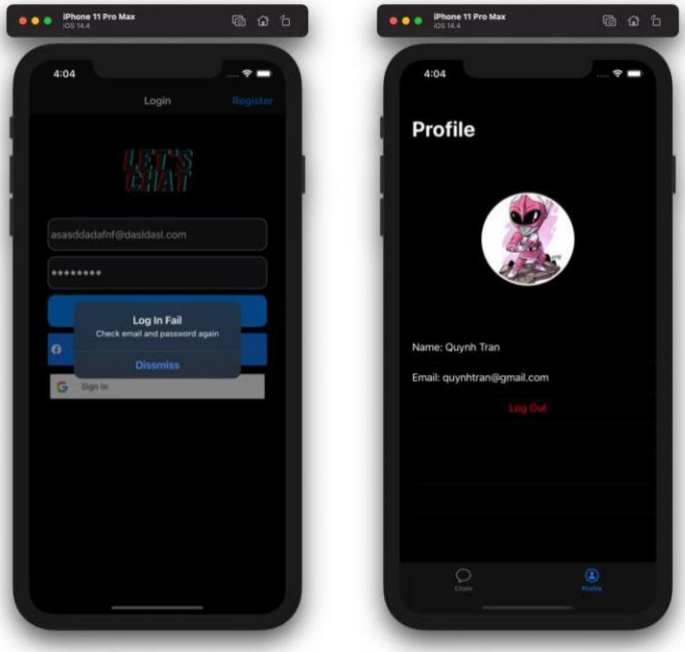


Figure 25. Login Process

6.3 Login with Facebook Account

The login with Facebook Account section was tested as follows:

- Facebook login button

tap. The expected result:

- Facebook SDK provided login page will appear.
- The user can give credentials.
- Data collected from Facebook account can be stored and fetched from Firebase.

The result described in pictures:

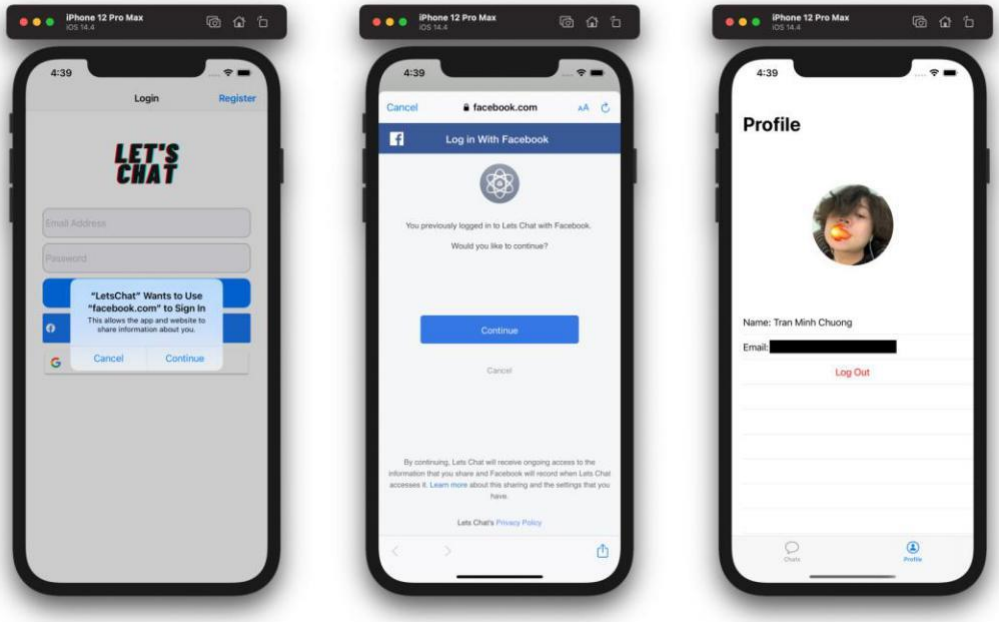


Figure 26. Facebook Login Process

6.4 Login with Google Account

The login with Google Account section was tested as follows:

- Google login button tap.

- The expected result:Google Sign-In provided login page will appear.
- The user can give credentials.
- Data collected from te Google account can be stored and fetched from Firebase.

The results described in pictures:

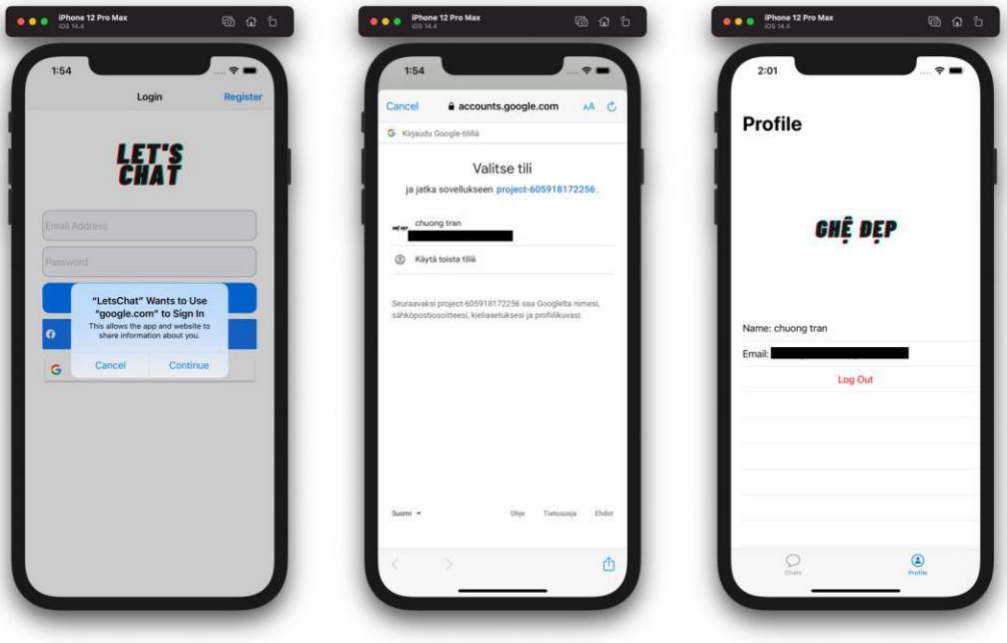


Figure 27. Google Sign-in Process

6.5 Search Recipient User and Create New Conversation

This process was tested under Facebook logged-in user (Tran Minh Chuong).

The process of search for the recipient user and starting the conversationwas tested as follows:

- Search for the recipient user by the name.
- Sender user search for itself.
- Search for non-existing user.
- Tap the search result.

The expected result:Result expected:

- Able to search for an existing user to be a recipient.
- Not be able to search for a logged in user to avoid creating a conversation with itself.
- Non-existing user gives no result.
- The conversation created.

The results described in pictures:

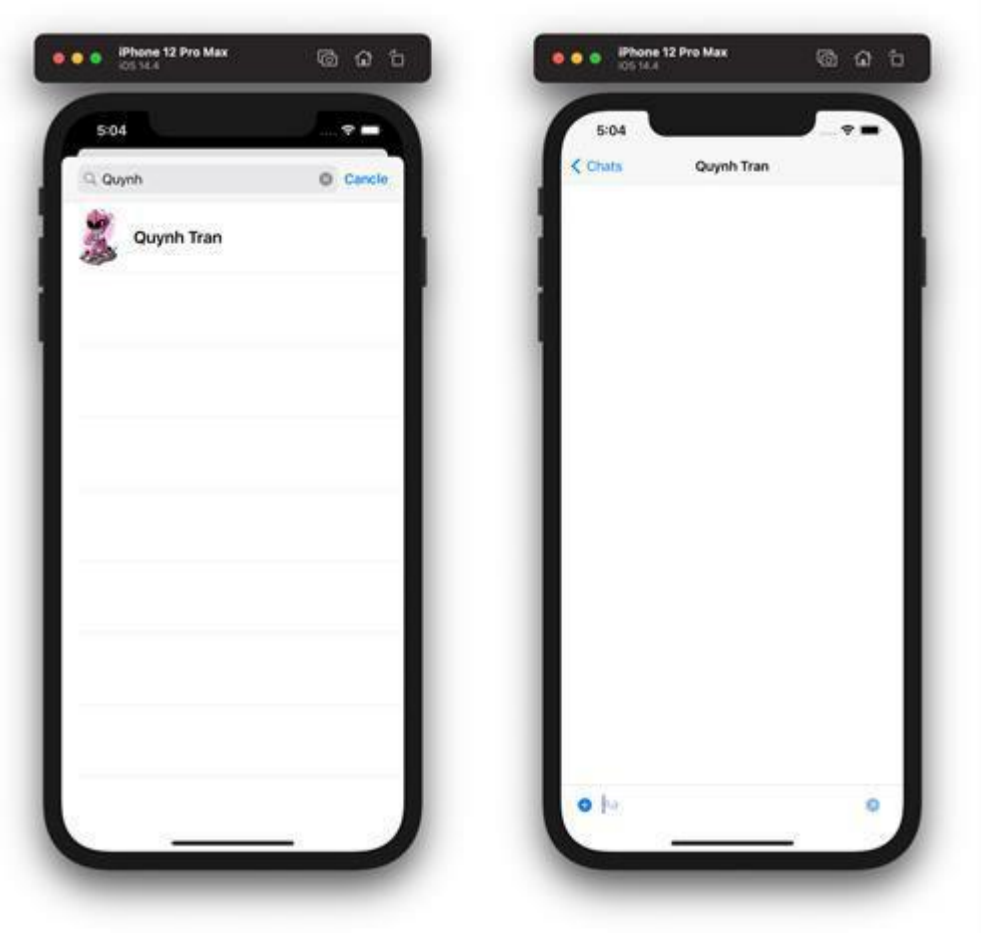


Figure 28. Search and Create Conversation

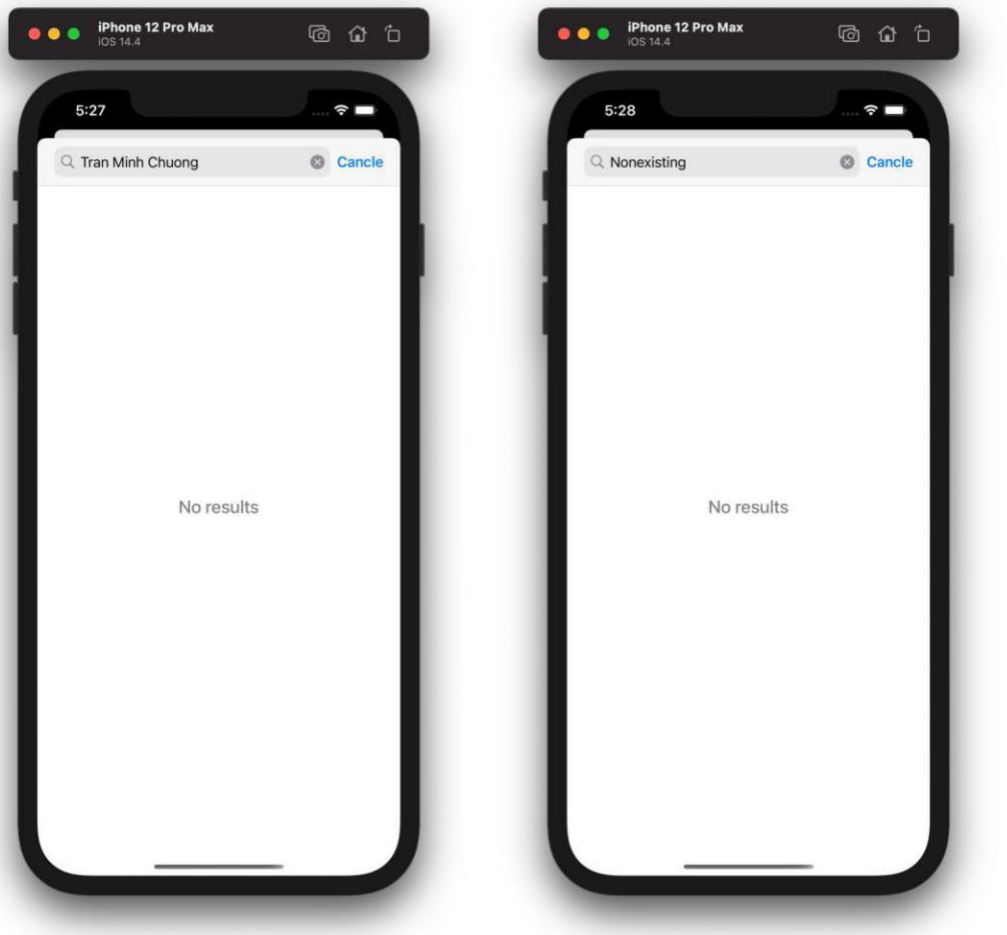


Figure 29. Searching Own Self and Non-existing User

6.6 Sending and Reading Text Message

This process was tested under Facebook logged-in user (Tran Minh Chuong) as a sender and registered user Lam Nguyen as a recipient.

The process sending text was tested as follows:

- A conversation created with the recipient user.
- Sending text message to the recipient.
- Recipient user reply.
- The expected result: Recipient is able to see the message.

- Users are able to chat with each other by sending text messages.
- Conversation is created and its messages saved in Realtime Database.
- The conversation should appear for both users on Chats page which contains all the user conversations.

The result described in pictures:

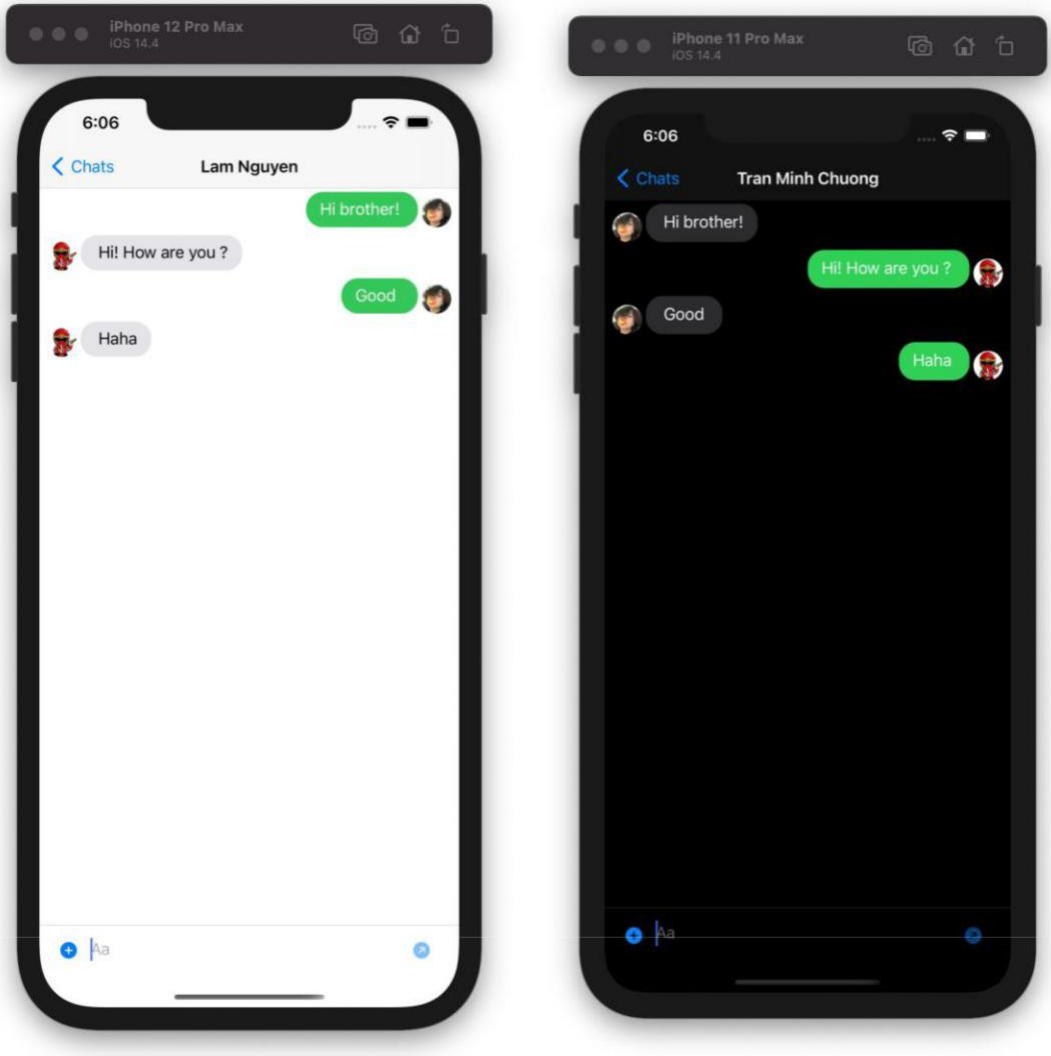


Figure 30. Users Send and Receive Text Messages

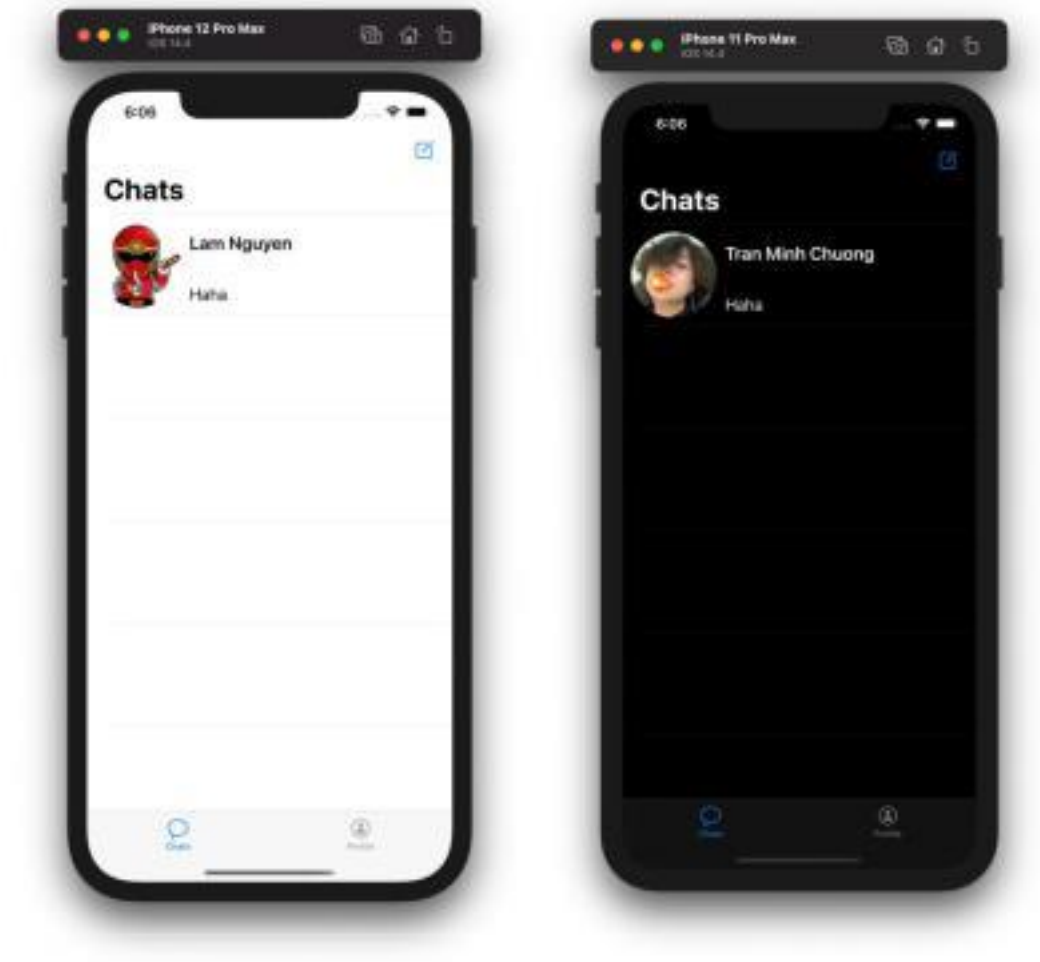


Figure 31. Users Chats Page with Conversation

Figure 32. Conversation saved in Realtime Database

6.7 Sending Photo/Video Message

This process was tested under Facebook logged-in user (Tran Minh Chuong) as a sender and registered user Lam Nguyen as a recipient. In this section, the Facebook account user was tested on the physical iPhone XS Max device to test the sending of a photo message with a photo capture from the camera; thus, the simulator could not be tested.

The process sending text was tested as follows:

- Sending photo message from library
- Sending photo message from camera
- The expected result: Recipient is able to see the message.

- The photo should be sent and it should appear.
- Photo/Video messages should be stored in the Firebase Storage.
- Photo/Video should be sent from Camera
- Photo/Video should be sent from Library
- Photo/Video should be able to download from Firebase.
- When users choose to send a video from library, the library should only show the video type file, not the photo and vice versa.
- The conversation will be updated with “Sent Photo” or “Sent Video” for both users on Chats page which contains all the user conversations.
- Video message should be playable.

The results described in pictures:

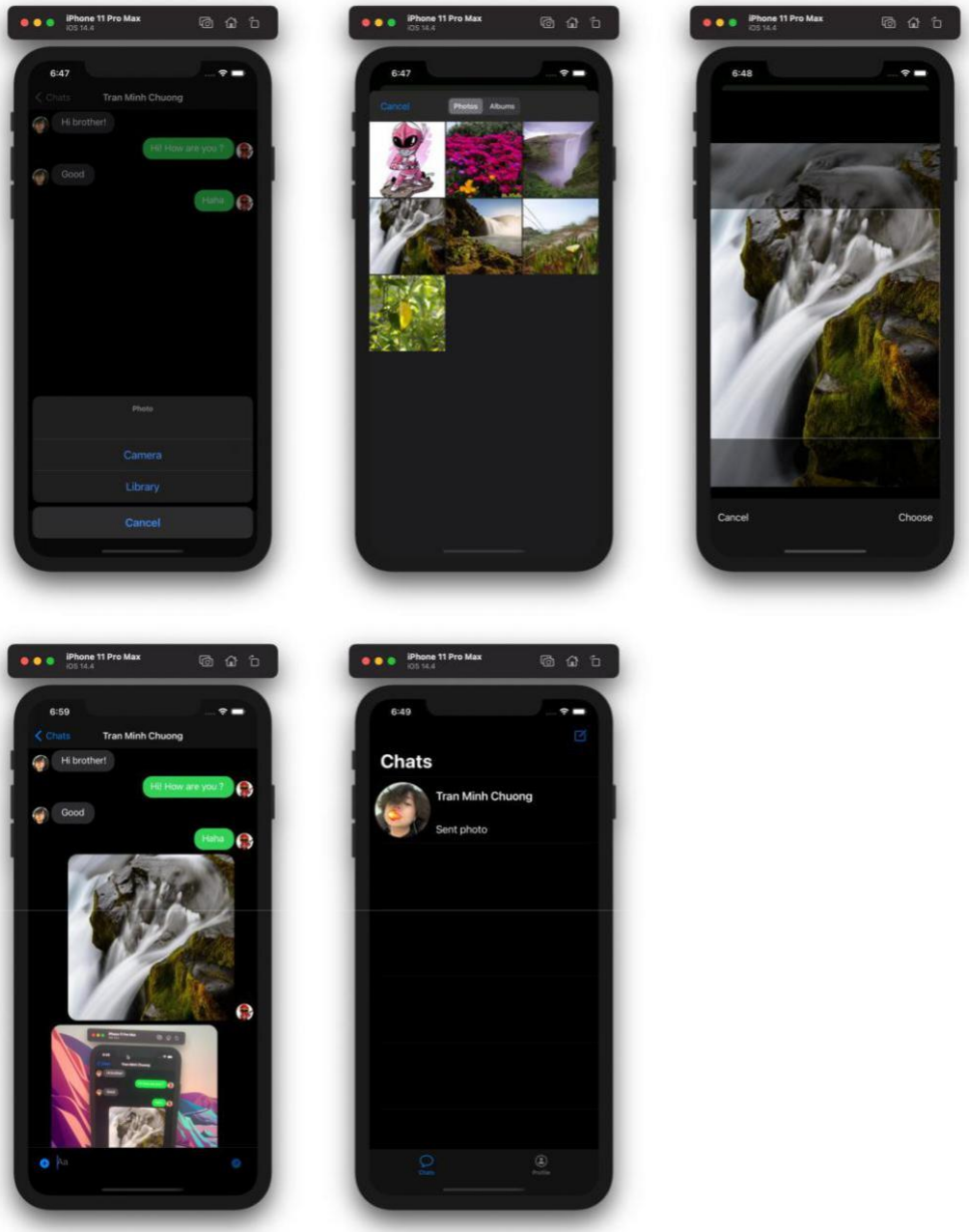


Figure 33. Sending Photo from Library Process

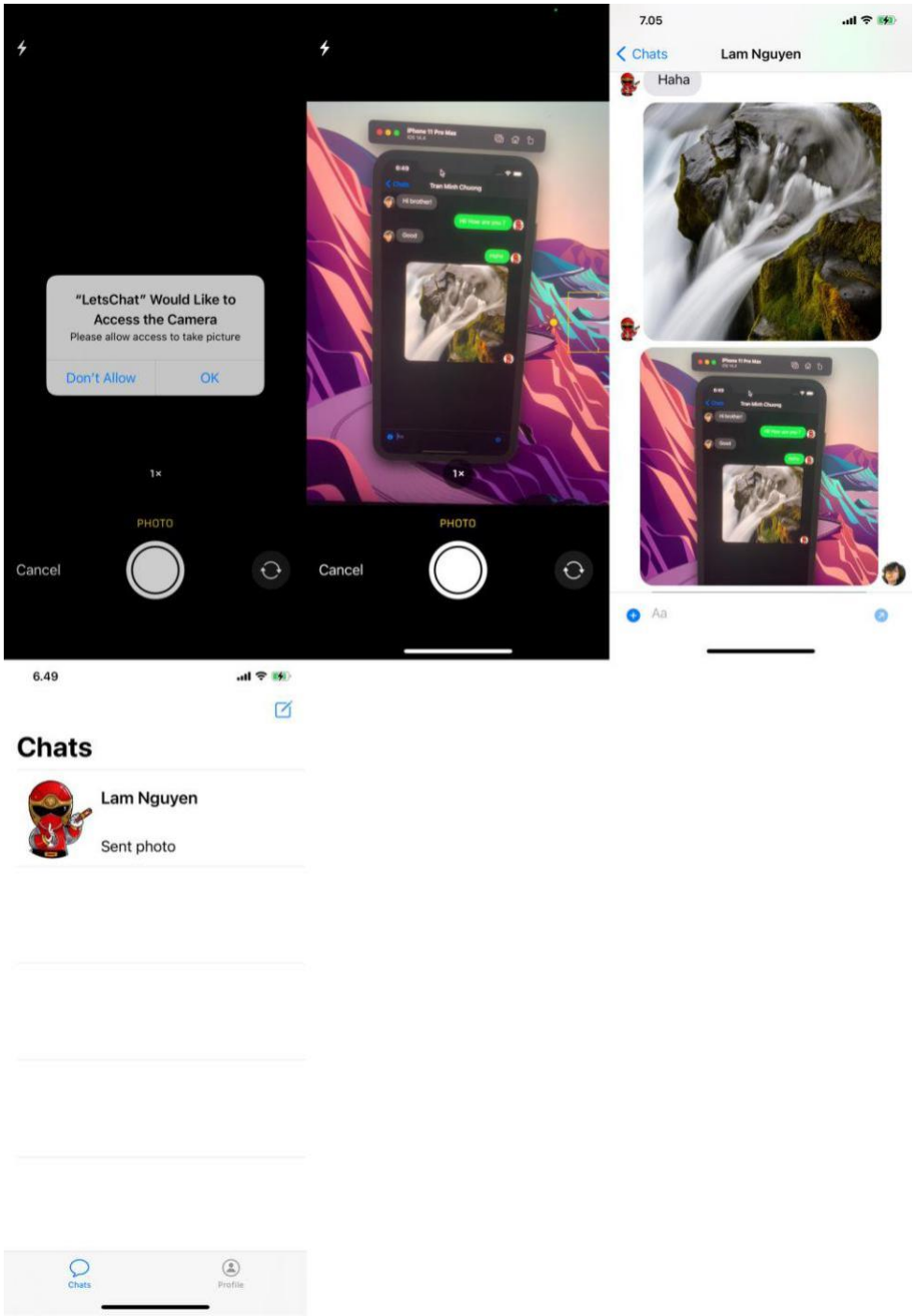


Figure 34. Sending Photo from Camera Process

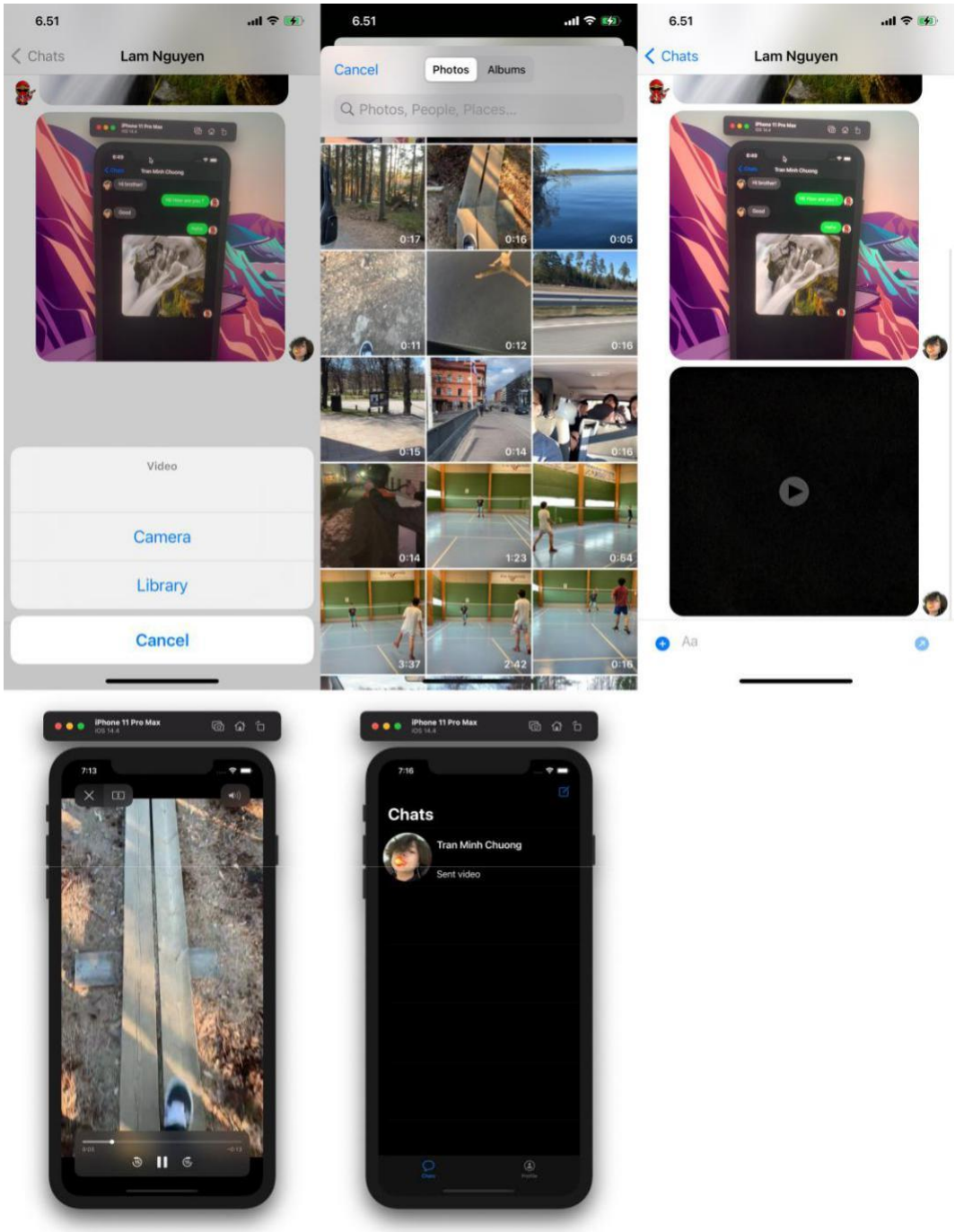


Figure 35. Sending Video from Library Process

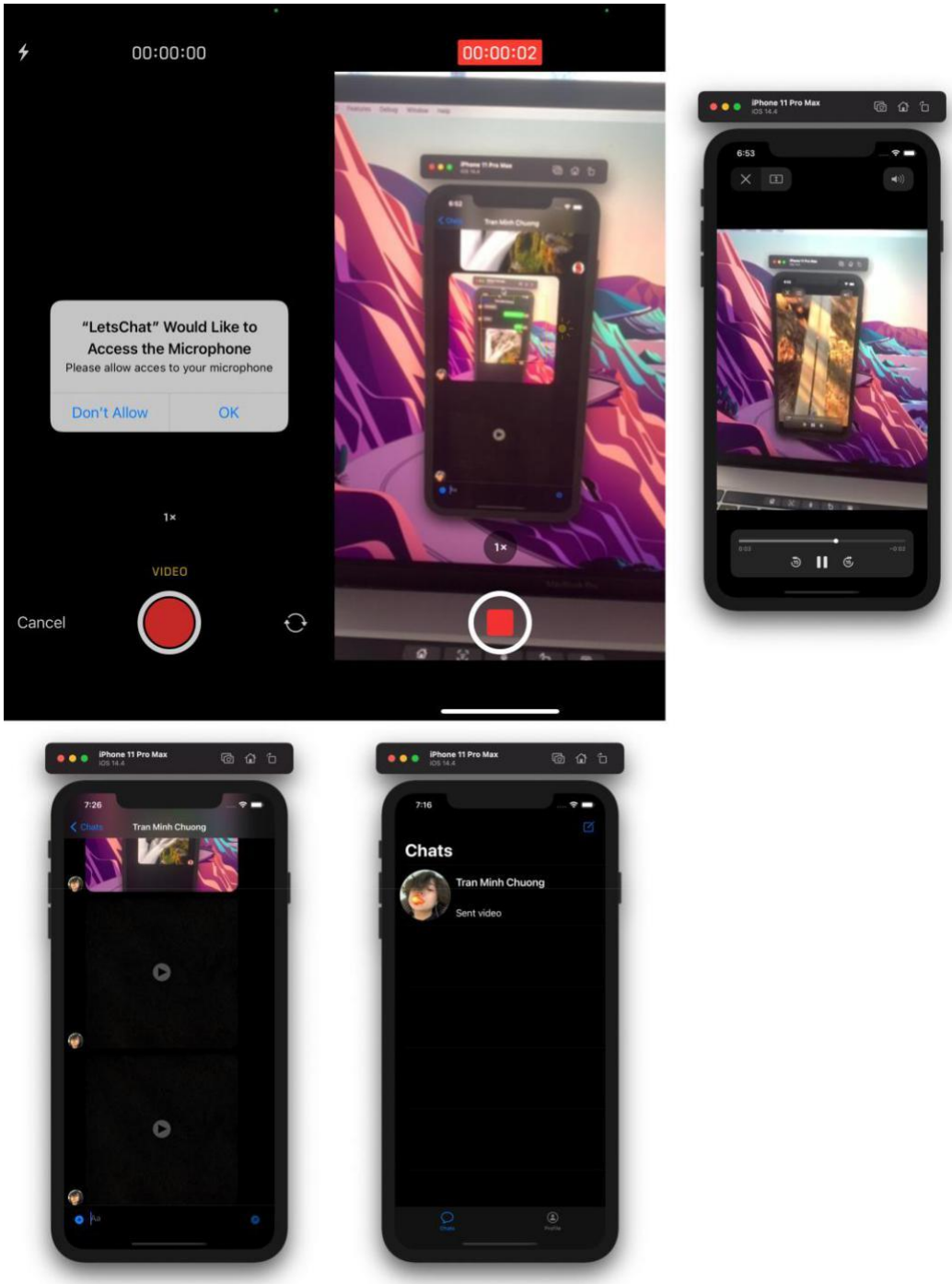


Figure 36. Sending Video from Camera Process

Figure 37. Photo Messages Store on Firebase Storage

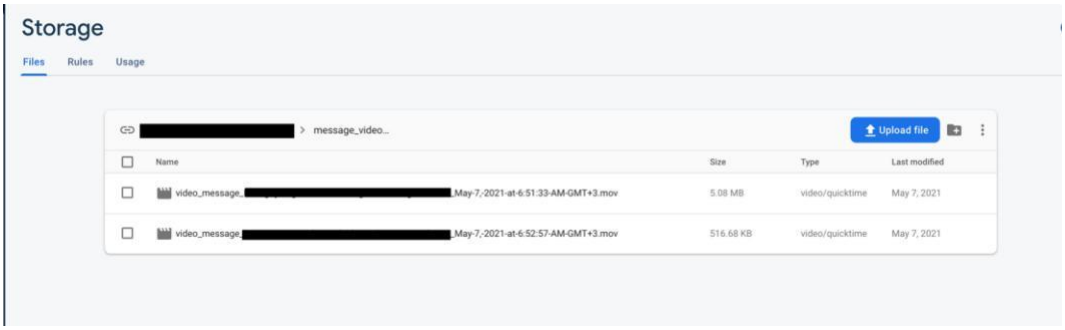


Figure 38. Video Messages Store on Firebase Storage

6.8 Deleting Conversation

This process was tested under registered user Lam Nguyen.

The process deleting conversation was tested as follows:

- The user swipes the selected conversation to be deleted to the left

The expected result::

- The conversation should no longer appeared on the Chats page.

The results described in pictures:

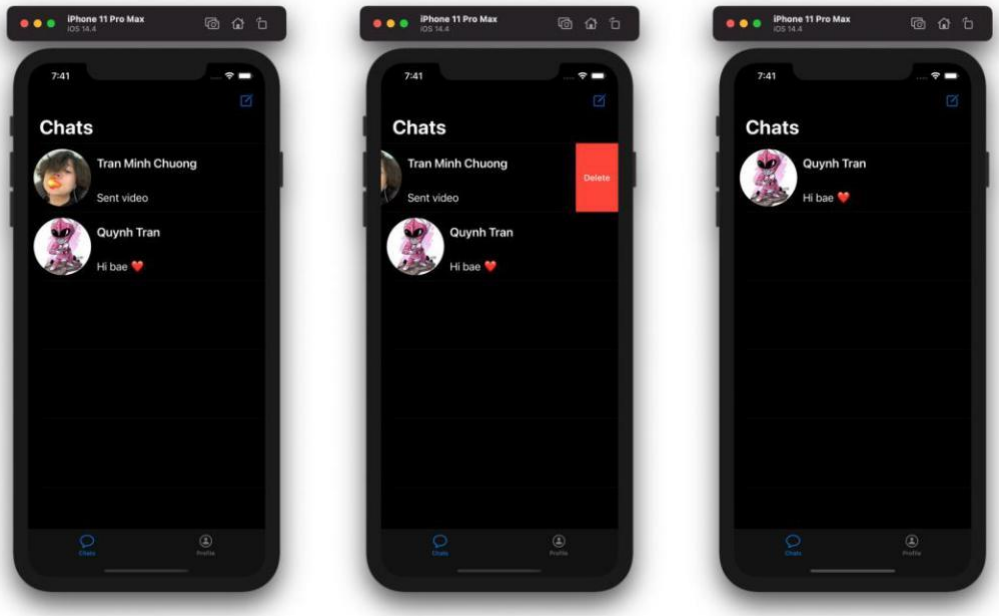


Figure 39. Deleting Conversation

7 CONCLUSIONS

The main purpose of this project was to develop a chat application called "LetsChat" for the group of iOS users using Swift and Firebase The application has the basic specifications of any chat application on the market, including register, log in/log out, sending text messages, sending photo messages, sending video messages, delete conversations.

The application works as expected, communication between users in conversation is smoothly performed without a crash, even through there are functions that could be improved. All the data of the application is stored securely on the Firebase. Firebase takes a critical part in the project; Firebase is a really powerful platform that has 18 products to help the implementation and development easier.

The a user interface of the application is not the best and cannot compare with the others on the market but it is nice looking and straightforward, easy to use with the help of Swift's components and libraries. Facebook Login and Google Sign-in made the log-in section less burdened.

Although the application is not in a big scale, during the development of the project, previously unknown technologies had to be studied, learnt and adopted , which was a challenge. A lot of definitions, skills were well adapted alongside completing the project.

This project has given me a lot of needed knowledge as I want to become a software developer in the future .Working with the project gave an insight to application development.

Some of the figures in this report have been covered up for the purpose of privacy and security.

In the future, with some more effort put into this project, the function will be enhanced with a better, faster implementation; other interesting-to-have specifications will be developed, such as sending location, sending file, make a group chat.

A better GUI design will be upgraded. The application will be maintained frequently and the ultimate objective is to put it on the Appstore to compete with others existing chat application on the market.

REFERENCES

/1/ Xcode. Accessed 05.05.2021.

https://en.wikipedia.org/wiki/Xcode.

/2/ Swift Programming language. Last accessed 05.05.2021

https://en.wikipedia.org/wiki/Swift_(programming_language)

/3/ What is CocoaPods? Last accessed 05.05.2021

https://stackoverflow.com/questions/22261124/what-is-cocoapods

/4/ What is Firebase? Last accessed 05.05.2021

https://howtofirebase.com/what-is-firebase-fcb8614ba442

/5/ Mobile backend as a service. Last accessed 05.05.2021

https://en.wikipedia.org/wiki/Mobile_backend_as_a_service

/6/ Firebase Authentication. Last accessed 05.05.2021

https://firebase.google.com/docs/auth

/7/ Firebase Realtime Database. Last accessed 05.05.2021

https://firebase.google.com/docs/database

/8/ Firebase Storage. Last accessed 05.05.2021

https://firebase.google.com/docs/storage

/9/ Facebook Login. Last accessed 05.05.2021

https://developers.facebook.com/docs/facebook-login/

/10/ Google Sign-In. Last accessed 05.05.2021

https://developers.google.com/identity