# The Product

This chapter talks about the methodologies and the design used to build the entire product, i.e., the Language Dependent Messaging App. It starts with the *Design and Project Management* sub-section which discusses the requirements of the app, the product design and the tools used to manage the project and ensure that it is going on time. This section is then followed by the *Development Tools and Methodologies* sub-section which focusses on the programming methods and tools used and the reason behind it. The *Implementation* sub-section illustrates the specific programming tools and resources used to develop the features in the product in chronological order. Finally, the *Result* subsection which demonstrated the final output and how it tackles the problem defined above.

## Design and Project Management:

Before implementing the App, the app was first designed to visualize how the app will function and what will be its basic features. Standard software development process was used while developing the project. It starts with listing the Requirements, then building the Use Case diagram and then the Component Diagram. Activity Diagrams for each functional requirement was also designed at this stage, but it’s shown in Implementation section for simplicity. The modelling diagrams used in this section are all a part of UML (Unified Modelling Diagrams). It is a standard that is used to design the software built.

### Requirements

At first its Functional and Non-Functional requirements were identified and listed. Functional requirements are specific functions and behaviour that a stake-holder has directly asked for and Non-Functional requirements are those that can be used to judge the App.

Following are the Functional Requirements:

**Must Have**:

1. A user must be able to Sign-up to, Login to and Logout from the app

Acceptance Criteria:

* The Sign-up and Login available to the user from the Home screen.
* The Logout option easily available to the user.

1. A user must be able to choose the language they prefer to communicate in.

Acceptance Criteria:

* The user is asked to choose the language only one time during signing up.

1. A user must be able to view all the users who signed up to the app and select one to message.

Acceptance Criteria:

* The users list available to the users after they sign-up or login.
* Each user list item in the list is selectable.

1. A user must be able to view all the messages sent and received from a selected user.

Acceptance Criteria:

* The messages are available in chronological order.
* The messages are available such that it is evident who sent the message.
* The messages are available in the correct language of their choice.

1. A user must be able to send and receive messages in the language they choose.

Acceptance Criteria:

* An area is available where messages can be typed in with a send button.
* A new message received is in the correct language of their choice.
* A new message sent is available in the messages list immediately.

1. When there is a new message and the app is not in the foreground then the user must be sent notification about the message.

Acceptance Criteria:

* Notification permission is taken when the app is first installed.
* The sender and the message are sent in the notification.
* On clicking the notification, it opens the chat screen to the sender.

**Should Have**:

1. A user should be able to set a nick name for themselves that will be available to other users.
2. A user should be able to reset their email, password and language.
3. A user should be able to change their password if they forget it.
4. A user should be able to view the original message sent if the received message is a translated one.

**Won’t have:**

1. The app won’t provide options to send and receive images and files.
2. This app won’t provide option for creating group chats.
3. The app won’t provide option to set the profile picture.
4. The app won’t provide option to make audio or video calls.
5. The app won’t provide options to set user status.
6. The app won’t provide options to block a user.
7. The app won’t provide options to delete chat history.

The Non-Functional Requirements are:

1. Usability:

Requirement: The App should be easy to use for all users

Acceptance Criteria:

* There should be clear instructions for the user on how to reach one screen to another
* The UI designed is to include users with colour blindness.

1. Platform Compatibility:

Requirement: The App should be compatible with all Android versions

Acceptance Criteria:

* Successful testing is done with Android emulators with Android version 6 onwards.

1. Robustness:

Requirement: The App is able to handle errors during execution and invalid inputs.

Acceptance Criteria:

* The opportunities for the users to deviate from the normal path is very limited.
* When the user enters invalid input such as wrong password then they are given immediate feedback

### Use Case Diagram

After the functional and non-functional requirements were set, next step was to build the Use Case Diagram. Use Case Diagrams are a part of the Behaviour Diagram under UML. It simply represents the various users and its interaction with the different use cases of the system. Following is the use case diagram built for

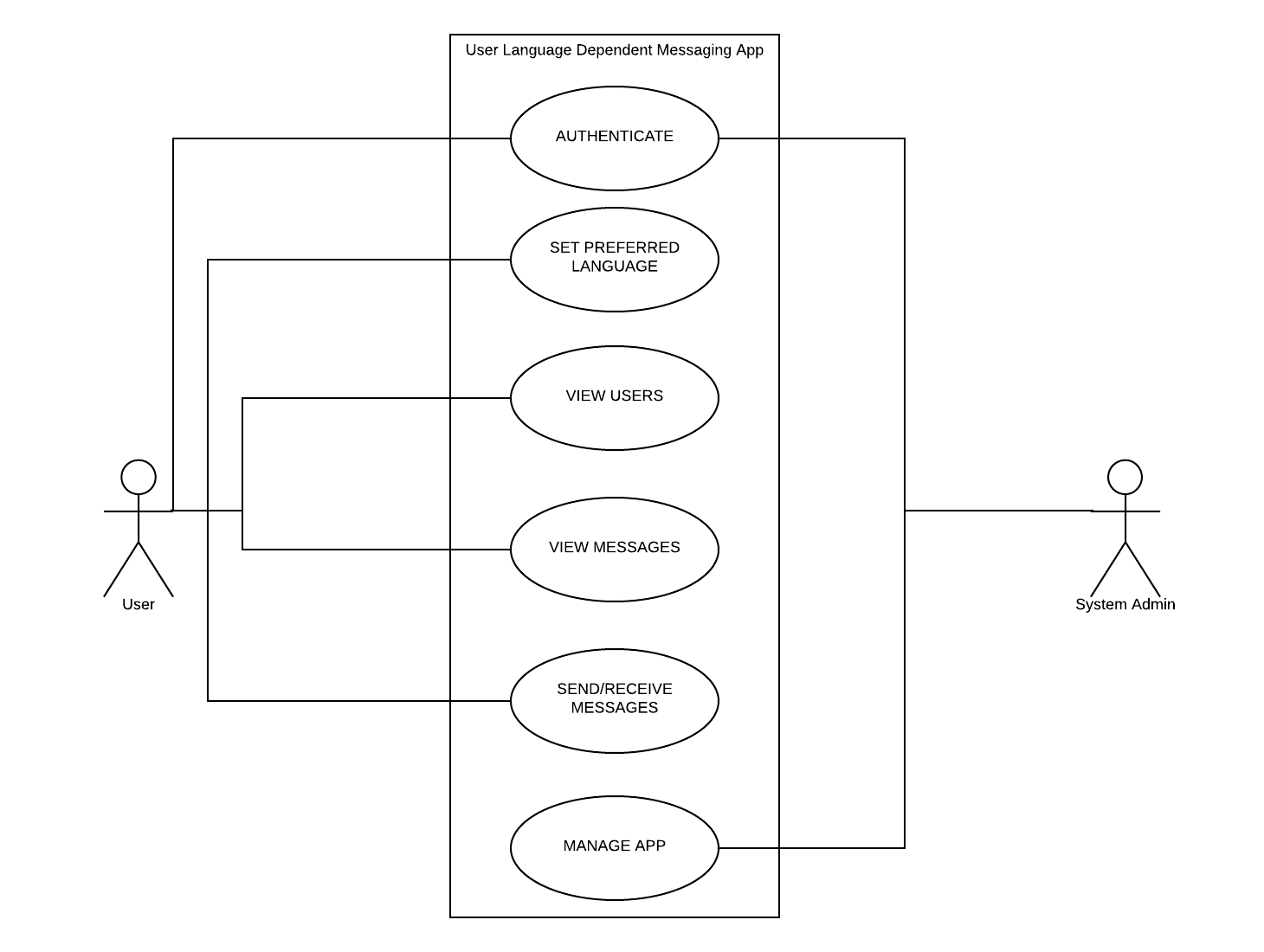


Figure: Use Case Diagram of Language Dependent Messaging App

### Component Diagram

Next is the Component Diagram. This diagram forms a part of the Structure Diagram of UML. It shows how components are joined together to form the software as a whole. The following is the Component Diagram used to design the various components of the app.

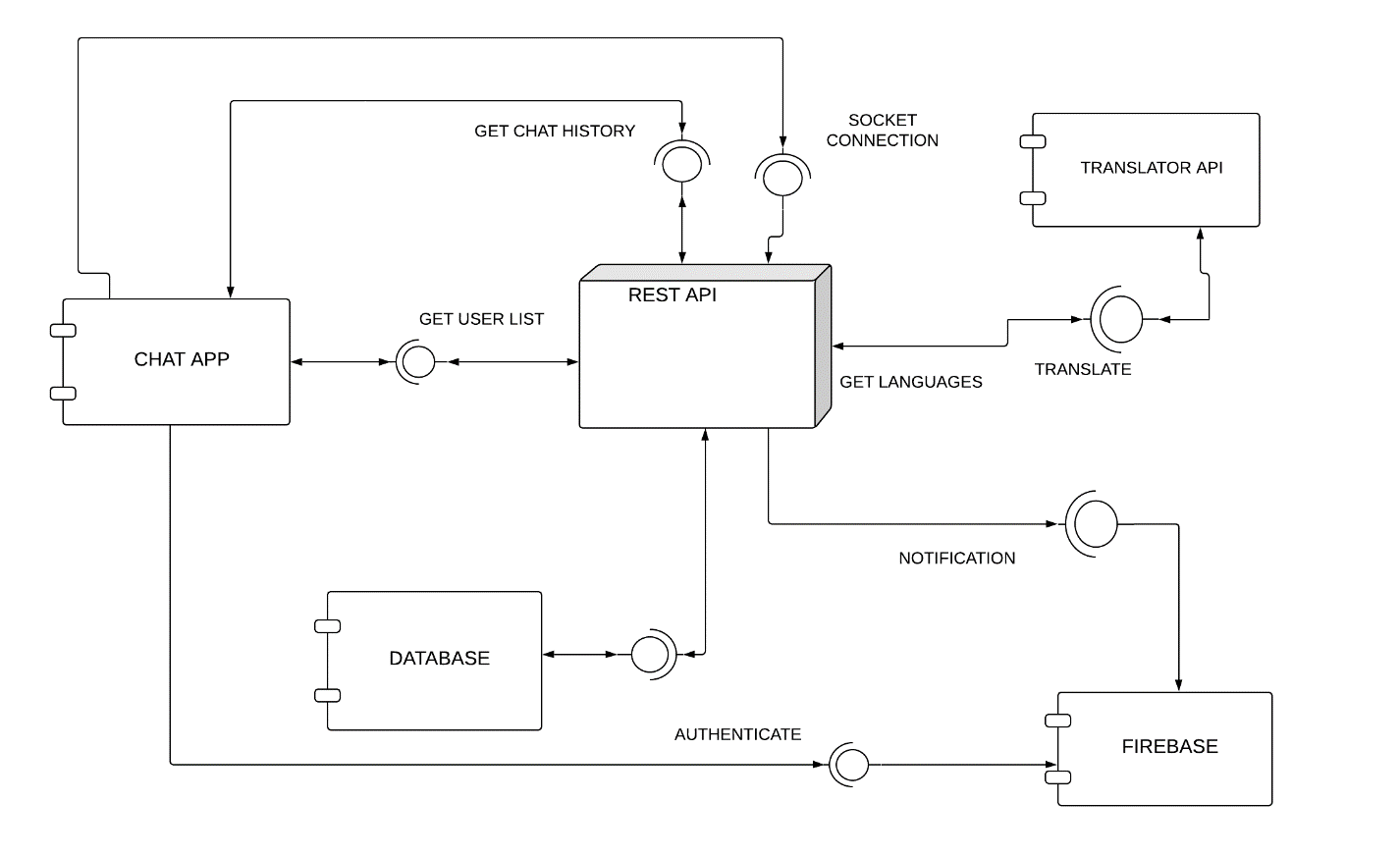


Figure: Component Diagram of Language Dependent Messaging App

### Project Management

Gantt Chart was used as a tool for project management. It was invented by Henry Gantt. It is a part of bar chart where activities are represented in the y-axis and time in the x-axis. Following is the Gantt Chart created for this project.

## Development Tools and Methodologies used

After the design of the App is completed, the tools and methodologies to be used to implement it was decided. This section talks about all the frameworks and libraries used in this project, including coding style. It starts with the software development methodology - Waterfall used and its lifecycle. It then leads on to discussions about the technologies used such as Mobile app development technology, REST API development technology, design patterns used etc.

### Software Development Life Cycle (SDLC)

SDLC is a process used to design, code and test high quality software. The process is followed in the form of model which define the software development process. The model followed in this project is the Incremental Model. In this model the product is designed, developed and tested incrementally. The product is decomposed into smaller portions and each portion are developed one at a time in the order of priority. In the first increment, working version of the product is released and then in the subsequent releases, features are added to the first increment.

The reason why this model was chosen was because of its advantages. The main advantage of this model is that it allows dividing the project into small iterations and thus have control on designing and developing them. Specific deadlines can be fixed on each iteration and development can be tracked. This model generates a working model in the beginning of the project and testing and debugging each iteration is easier.

As it is evident from the Gantt Chart explained above, the development of this project was done according to the Incremental model. In the Requirements section, the main requirements of the project were first identified. Then prior to implementation the design of the overall project was completed to understand the architecture of the project and then it is divided into smaller increments such as Login, 1-2-1 communication etc. During development, one iteration was completed and then features were added on top of it after satisfactory testing of the previous iteration was completed.

Some other software development models which are very common are the Agile Methodologies and Waterfall Model.

In Agile methodology, the software is built incrementally using short time frames from 1 – 4 weeks so that it can align with the changing Business requirements. The reason why this methodology was not used is because it provides best practices when working in a team of at least 3 – 4 members. Moreover, in Agile Methodology planning is done before each development cycle where the effort in that development cycle is assessed prior to its start and as this project was completed by me as the single developer it was not possible to estimate the effort required accurately.

In Waterfall Model, also referred to as *linear-sequential life cycle model* the software is divided into smaller units called phrases. Each phrase needs to be complete before the starting of the next phrase. The output of one phrase acts as the input of the next phrase sequentially. The phrases in the model are: Requirement Analysis, System Design, Implementation, Testing, Deployment of System and Maintenance. This model is only used when the requirements are fixed and clear, the technology is clear and abundant expert resources are available for completing this project. These uses make this model unsuitable for applying to this project as the model requires fixed requirements and technology which was not the case for this project.

### Design Patterns

Design patterns are general, reusable solution to a commonly occurring problems. These are the best practices followed by experienced object-oriented software developers. Some of the most common design patterns are Singleton, Adapter, Iterator, Command, Factory Method and Modal View Controller (MVC) pattern. In this project the design pattern MVC is implemented. The Modal deals with the data related logic, the View deals with the User Interface logic and the Controller deals with the business logic. Its main advantage is that it allows code reusability and easy code maintenance. For the Modal, NoSQL database MongoDB is used. For the View, React Native is used and for the Controller, a REST API is used.

### Programming Style

In this project, JavaScript is the programming language that is mainly used. Standard coding conventions were used while developing the App. The code is kept simple and easy to read. The naming of the variables and the functions are done using Camel Case naming. Hard coding of values was avoided as much as possible, especially while creating the UI. Also, comments are provided wherever necessary for easy maintainability of the code.

In addition to proper coding convention, defensive programming was followed as well. It is a form of defensive design that makes sure that the software keeps working when the user is diverted from the happy path. Defensive Programming is handled by appropriate error handling and by sending response status of 4xx if the user sent wrong request or 5xx if the server fails handling the request.

### Source Control

Source Control or Version Control is a medium to track and manage the different versions of code and documents. The most common tool used for version control is Git. It is a distributed Version Control system. In this project GitHub is used for version control which is owned by Microsoft and provides version control functionality of Git in addition to its own.

### Programming Tools and Methods used

In this project a large number of programming tools are used to develop the App. Following is a list of the main tools used and the reason why they were chosen:

* React Nativr:

For mobile app development React Native was chosen. It is an open source framework built by Facebook. It is used to build Apps in JavaScript which is compatible with both Android and iOS operating system. It helps developers reuse code to run software in both mobile app and web. The reason why this framework was chosen for building the app is because even though the initial project proposal was to build an Android App, using this framework gives the opportunity to expand the app to iOS operating system as well, thus bringing it closer to wider audience.

* Socket.IO

After confirming to use React Native, the next technology that was confirmed was to use WebSocket protocol for real-time communication. This protocol provides a full-duplex communication over a TCP connection, i.e., allowing a bidirectional communication between the client application and the web browser. This allows user of this app to have true real time conversation rather than the traditional way, where the client application sends request to the server for new messages over an interval and the server replying to it depending on the presence of new message.

The library used in this project to implement WebSocket protocol is Socket.IO. It is a JavaScript library. At first, Socketio library of python language was considered to handle the real-time conversation but due to the lack of online support for this library in comparison to the JavaScript Socket.IO library, the later was finally used.

* Express.js

For server sider development Express.js is used. It is a free and open source web application framework for Node.js. It is based in JavaScript. It provides a rich set of features to develop web and mobile application. Initially, Flask micro framework of python language was considered for server-side development, but after changing WebSocket library from Socketio in python to Socket.IO in JavaScript the language to develop the server side was shifted from Python to JavaScript. It provides easier integration and faster development as the entire project is built in only one language-JavaScript.

* Mongoose

For storing the data of the users and the messages sent and received a local database is used. The type of database used is MongoDB which is a type of NoSQL database. The main advantages of this database are

* High Scalability: NoSQL uses horizontal scalability. It is adding more machines to handle large volume of data whereas Vertical scalability is adding more resources to one machine to handle the large volume of data.
* High Availability: NoSQL database has auto replication feature making it highly available.

This project requires a database that can store large volumes of data and for long time. Thus, NoSQL database is the best option. Object Data Modelling library for MongoDB and Node.js, Mongoose was used in this project. This library provides data relationship management, schema validation and relates JavaScript object to MongoDB object. This allowed faster and easier development.

The other type of database available is SQL (Structured Query Language) database. It deals with Relational Database. SQL follows vertical scaling hence when it is used for large volumes of data, the system gets slower.

* Firebase

Firebase is Google’s own mobile app development platform that allows business to rapidly grow along with providing features such as Cloud Firestore, ML Kit etc that help building the app. In this project Firebase is used for authentication and notification.

Firebase authentication provides a number of authenticating options such as email, phone number, social media login etc. As this project is a messaging app, these options provide a number of opportunities for future implementations like adding the user phone number. Moreover, for easier user authentication, Firebase can also be used to provide social media login such as Google login and Facebook Login.

For notification service in the app, which allows the user to be informed of a new message, Firebase Cloud Messaging (FCM) service is used. Sending a notification to an Android app can only be done via FCM. Moreover, it is also a cross platform messaging platform that allows sending notification to iOS. In addition to FCM being a cross-platform messaging service, this service allows testing of notification to client applications by sending notifications from the Firebase console. Thus, making testing of background notification service in the app very simple and easy.

* Translator API Integration

For translating a message from one language to another, a translator API is utilized. The main APIs available for Translation are:

* Google Translate API:

The Google Translate API is a translator API provided by Google. It allows dynamic translation between thousands of language pairs. It is easy to integrate and provides <500,000 free character translation per month. Originally Google API used Statistical Machine Translation but in 2016 it transitioned into Neural Machine Translation. After the usage of the free characters, translating 500,001-1billion character cost $20 per million character and using Language Detection cost $20 per million character. It has 104 supported language.

* Microsoft Translator API:

Microsoft Translator API is provided by Microsoft Azure which is Microsoft’s cloud computing platform. This API provides 2million characters per month for free and it includes language detection as well. It uses Neural Machine Translation. It provides a many payment options such as pay as you go where it costs £7.454 per million characters. It supports a total of 63 languages.

* Amazon Translate API:

Amazon Translate API is a service provided by Amazon Web Service which is cloud computing platform provided by Amazon. It provides language identification along with translation. It provides 2million characters per month free in its free tier and costs $15 per million character. It also uses Neural Machine Translation. It supports a total of 25 languages.

Out of the 3 translate APIs listed above, Microsoft Translator API is used in this project. It provides a good range of languages for translation and 2M characters per month for free along with language detection. It also allows easy integration to JavaScript server along with extensive official documentation availability. In addition to these features, signing up to this service do not require registering any financial cards.

## Implementation

The Language Dependent Messaging App is a complicated app developed using a number of tools and libraries. This section explains the implementation of the app and how it was developed. For simplicity the section will follow the navigation flow of the app shown in the figure below. It will first illustrate the general flow of the app and the UI of the entire app. Then it will explain the development of each part with first describing the functional part and then the UI of each part. The fulfilment of each functional requirement will be highlighted at the same time.

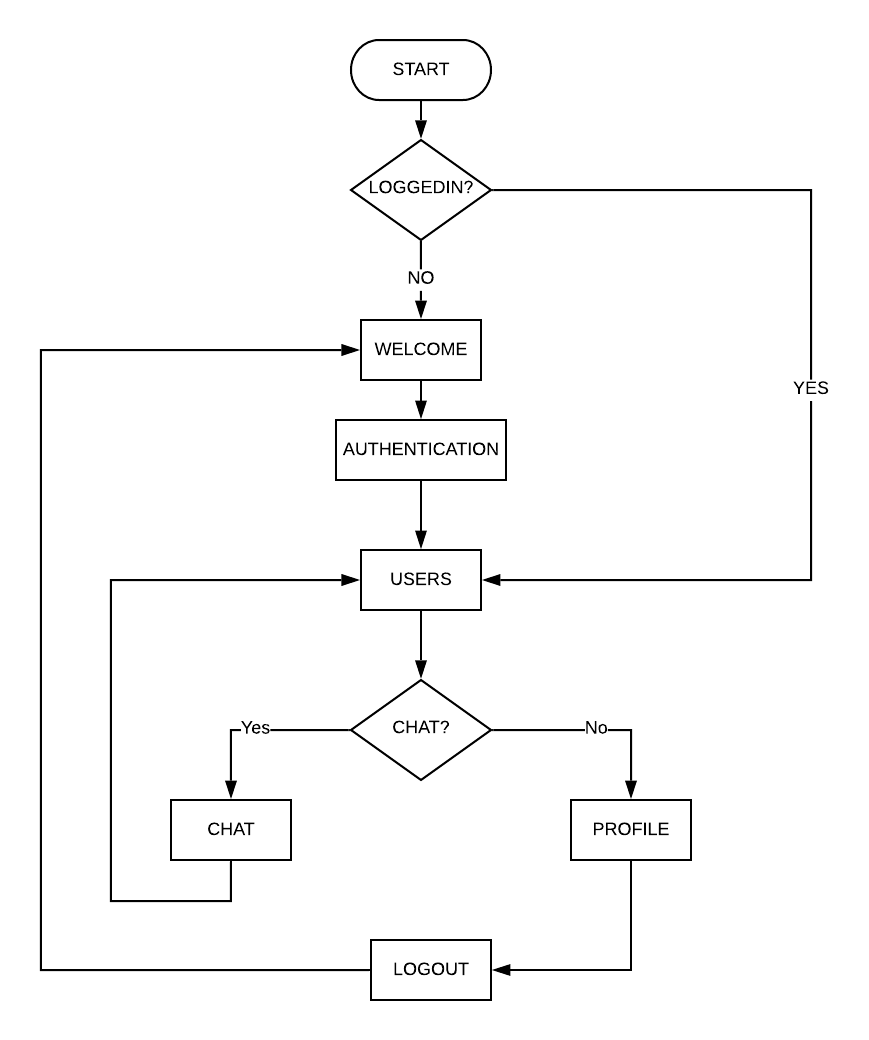


Figure: Navigation Flow Chart in the App

As it can be observed in the above figure, when a user opens the app, it is first checked if the user is authenticated or not. If the user is authenticated then they are taken directly to the *Users* screen. If not then the user is directed to the *Welcome* screen where the user has the option to

The following sections elaborates the App UI, Authentication, Users Screen, Chat Screen and the Profile Screen.

### App User Interface

This section discusses the UI design that is common to the entire app. Designing the UI of the App started by first building the wireframes (added in Appendix A) to visualise the tentative design of the UI in each screen. These wireframes were built using Mockflow, an online wireframe creating tool. These wireframes were then sent to users who volunteered to take part in a study where the users were asked if the UI was satisfactory and if there were some missing features (questionnaire added in Appendix B). A total of 17 users took part in this study. Some of the feedbacks received are:

* As it can be seen from the wireframe pictures, they were built in black and white as the colour of the app were not decided then. Hence, the main feedback I received on the Welcome Screen was to implement graphics.
* In this project there are no options of Google or Social media login, and this was another feedback received. As the addition of these options were not the main aim of the project, they were not included.
* In the login screen, the feedback that most users provided was to add the option of *Forgot Password* and in the Sing up screen the feedback received was to provide the feature of adding username. Even though these features were a part of the requirements, they were not included in the wireframes. However, they are now implemented in the project.
* The option to mask and unmask the password field was one of the feedbacks received. This has been implemented in the final project.
* Another feedback received was to implement the option to send media such as image, documents along with text. This feature is out of scope for this project.

The UI finally was built following standard Material Design specifications, first invented by Google in 2014, together with the user feedbacks. Material Design is based on grid layouts, shadows, lighting, padding etc. In this project the tools used to build the UI were React Native components and Native Base components. Both these tools follow Material Design.

The colours chosen for the App are “#160000”as the Primary Colour, “#ffd500” as the Secondary Colour and “#c6c6c6” as the colour for disabled buttons. The main reason behind selection of colours were to select a combination that is inclusive of users suffering from colour blindness. Hence, colours such as red, green and blue were avoided. Moreover, in order to make the app most inclusive, every icon used is associated with a text that let users know about the function of the app.

The design of the App is maintained consistent throughout by maintaining the width, height, colour of similar items. A consistent Header is kept across the app in the Primary Colour which provides the user of the information in which screen they are in and options such as Settings and going back to the previous screen.

In a React Native project, when a component is loaded the methods which are called and the order in which they are called are:

constructor() > static getDerivedStateFromProps() > render() > ComponentDidMount()

This lifecycle is used while developing the project and rendering the UI of each screen.

### Authentication

This section talks about the authentication process in detail. It starts with the following figure which shows the low-level activity diagram for the Authentication process. UML activity diagrams are used to depict the flow of control from one activity to another. It is used to show conditional flow, concurrency and loops. It is then followed by the functional explanation of Login and Signup process and then the UI.

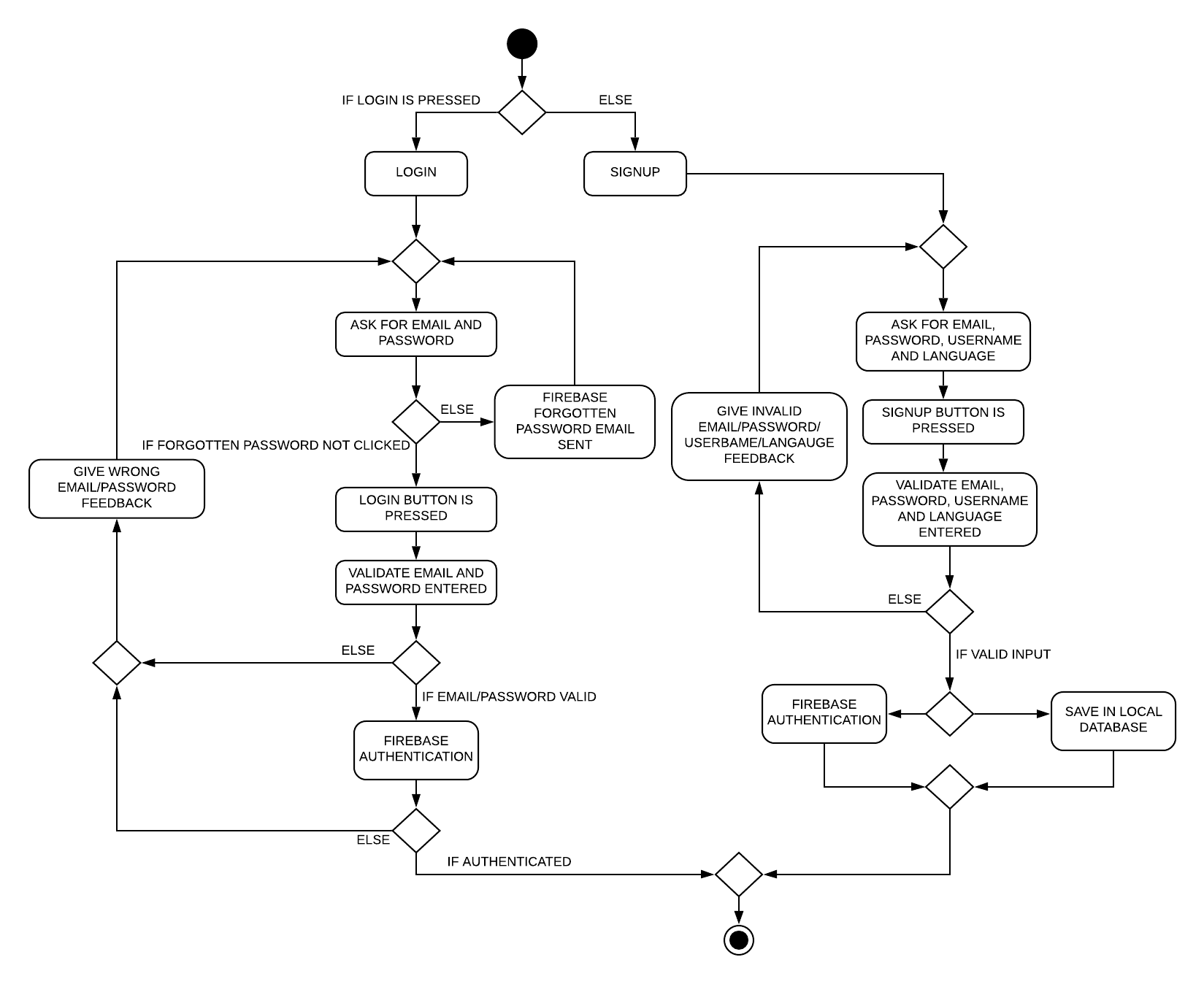


Figure: Low-level activity diagram for user authentication

The following paragraphs will explain the Login activity first and then the Sign-up activity.

If the user, clicks the *Login* Button in the *Welcome* screen then they are navigated to the *Login* screen. There they are provided with the option to enter either email and password to login or to enter email and click *Forgotten Password*. In both the cases the user input is validated locally if they have entered the correct format or even if they have entered the values or not. If the user input is invalid or if the user has missed the email or the password then the user is informed of the error in the form of an alert box. If the user has entered correct email and have pressed the forgotten password link then a link to reset the password is sent to their email address via Firebase. On the other hand, if the user has given valid details and pressed *Login* then the email and the password entered is sent to Firebase Authentication via a request. Firebase handles the authentication and returns the response. In case of any error such as wrong password or unregistered email, the user is informed and asked to try again. Else, the user is navigated to the *Users* screen.

If the user has clicked on the *Sign-up* button then they are moved to the *Sign-up* screen. Here they are asked to enter their email, username, password, confirm password and choose the language they prefer to message in. The options available to user to select the languages are obtained by simply making a GET request to the translator API used and the languages returned in the response are used as the options. The user inputs here are also validated and if any of the inputs are invalid then the user is informed of it via alert box. If the inputs are valid inputs and the user clicks the Sign-up button, this triggers two activities:

1. It informs firebase of a new user and its email and password via a request.
2. It sends the user’s email, username and language to the server to store it in the database.

After the completion of the above activities, the user is then navigated to the *Users* screen.

After the user logs in or signs up, the authenticated state of the app is stored in React-Native’s local store called Async Storage. It is stored so that if the user reopens the app, they won’t need to reauthenticate unless they logout or uninstall the app. Hence, when the user reopens the app, they are directly navigated to the Users Screen.

The UI of the *Login* and *Sign-up* screen is a basic form using the Native Base components and the fixed Primary and Secondary Colours. The Button used in both the screen is a “TouchableOpacity” component of React Native styled as a button instead of a traditional “Button” component as providing custom style to the Button component in React Native is very complicated and the simplest way to add a button was via the TouchableOpacity component.

The functional requirements that this section fulfils are the “A user must be able to Sign-up to, Login to and Logout from the app”, “A user must be able to choose the language they prefer to communicate in”, “A user should be able to set a nick name for themselves that will be available to other users” and “A user should be able to change their password if they forget it”. The Logout functionality is fulfilled in the Profile Screen. This functionality is detailed in the Profile Screen section.

### Users Screen

This section explains the User screen in the app and its functionalities. The following diagram shows the low-level sequence diagram for the requirement: “A user must be able to view all the users who signed up to the app and select one to message”. For this activity, UML sequence diagram is selected as it best shows the flow of messages from one object to another. The following diagram shows the flow of messages from the User screen in the mobile app to the server while requesting all the users in a list.

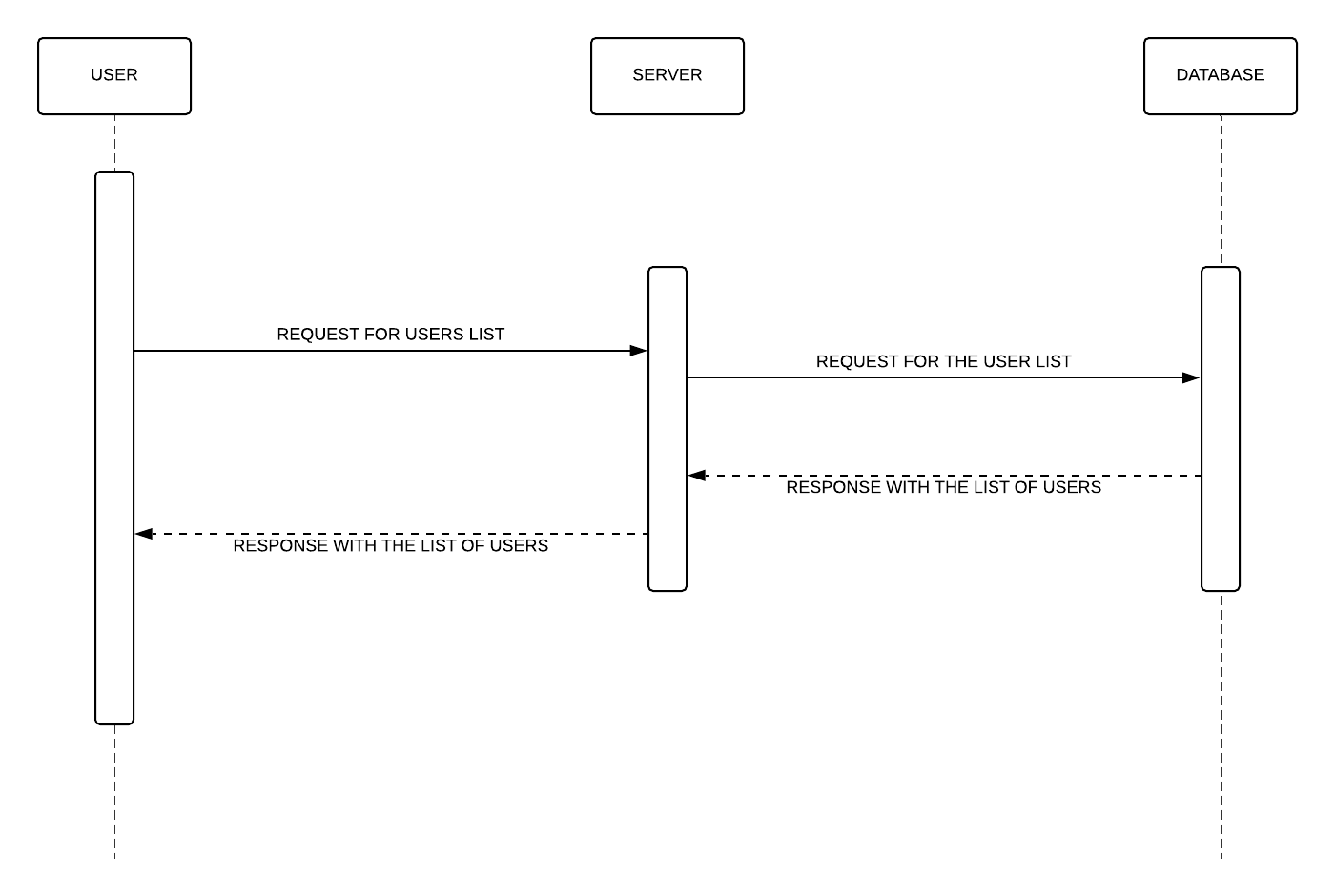


Figure: Sequential diagram of getUsers().

When an authenticated user opens the app or after an unauthenticated user is authenticated then the user is navigated to the Users Screen. In this screen all the users who have signed up to the app is shown to the user. The user can select any of the users from the list and start a conversation. As soon the User Screen is loaded then the componentDidMount() method of the React is called as it can be seen from the React Native lifecycle explained above. In this method the request to the server to get all the signed in users of the app is formed and made. The response returned from the server is in the form of a list. Each item in the list is exhibited such that they are clickable and on clicking them the user is navigated to the chat screen where the user can check the chat history with the receiver and also send and receive message. From the user screen the user can also move to the Profile screen where the user have the option to edit their details. The Users screen also has a search bar where the user can search for the user they want to message to. This search bar was implemented keeping into mind that as more and more user signs up to the app, the users list can get pretty long and thus make it difficult for the user to find a particular user.

The UI of the Users screen is built using List component from the Native Base with each user as a list item. The main reason why Native Base was used for the List is because it provides inbuilt template for list items where a thumbnail of the user picture is shown in the left, the username of the user and their email is shown in the centre and a right-arrow icon is shown in the right which informs the user that the list items are clickable. Although there is a space to show the picture of the users, for this project a fixed picture of a minion is chosen as a placeholder. To be able to show pictures, the database will be needed to be modified to be able to store images and this feature was out of scope for this project. Furthermore, the way via which the user can navigate to the Profile Screen is a Settings icon in the right-hand side of the Header. On clicking this button, the user is moved to the Profile Screen.

In this screen, the notification listeners are added but they are applicable to the entire app as long as the user is authenticated. The app can listen for notifications when the app is in foreground or background and act on it accordingly. The following figure shows the sequence diagram for the flow of the notification message.

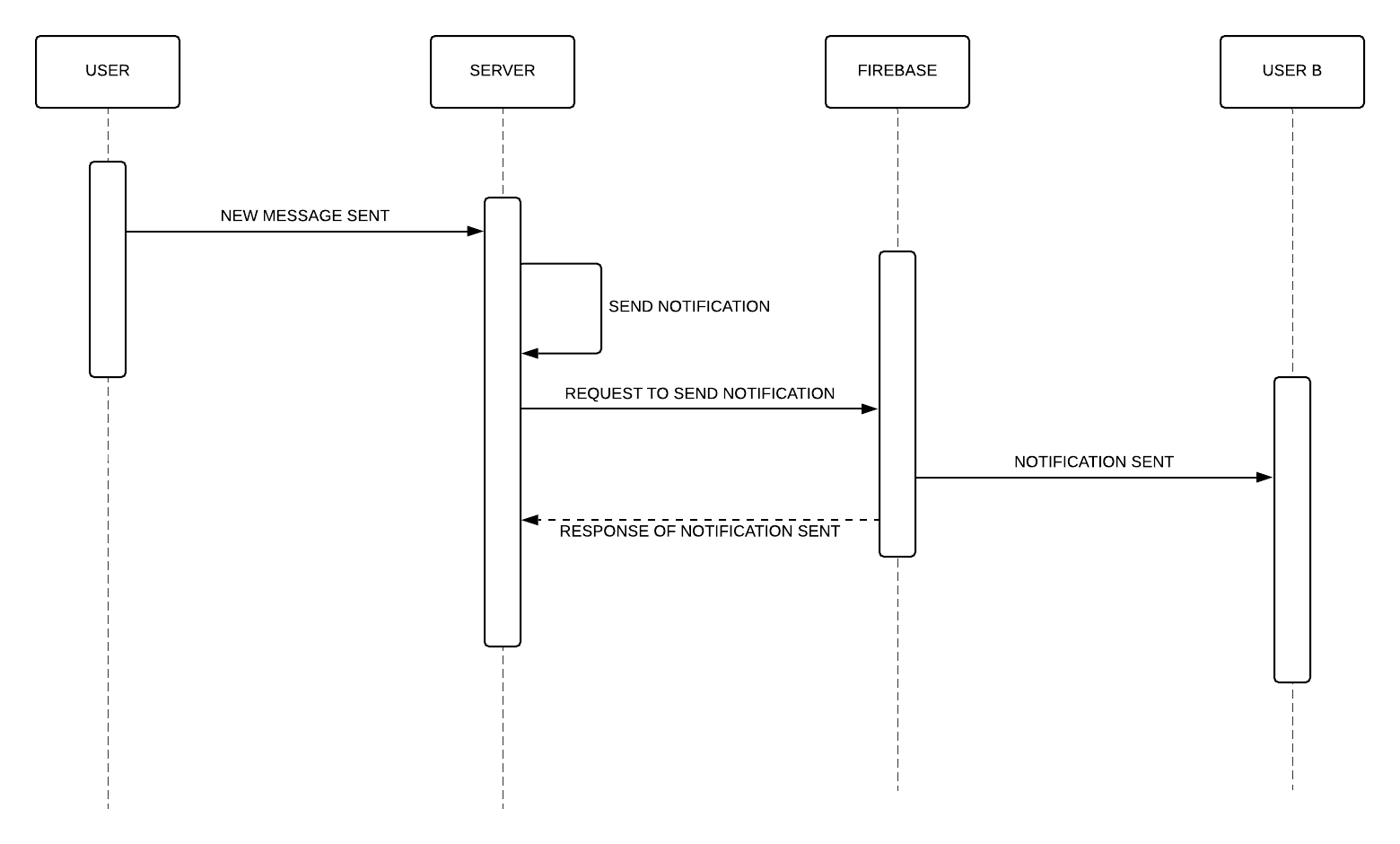


Figure: UML Sequence Diagram for sending notification

Notifications in this project is handled by Firebase. When each user logs in or signs up, they are given a unique topic name. Then each user is registered to Firebase with that topic name. This was Firebase is able to distinguish the user and send targeted notification. Once the user logs out, they are also unsubscribed from that topic name.

In the above figure there are two users, User A and User B. User B has their app running in the background. When user A sends a new message to the server, the server then sends a notification request to Firebase with the user A email id, the message and the topic name of the receiver. Firebase then sends notification to the user with the user who is subscribed to the topic name received.

In this section it can be seen that the requirements “A user must be able to view all the users who signed up to the app and select one to message” and “When there is a new message and the app is not in the foreground then the user must be sent notification about the message” are fulfilled.

### Chat Screen

A user is navigated to the chat screen when a user selects the receiver they want to message from the Users Screen. Hence, in this screen we have the email of the user and the email of the receiver as the data thus request is made the server for the chat history. The following UML sequence diagram shows the flow of messages between the user and the server in this request.

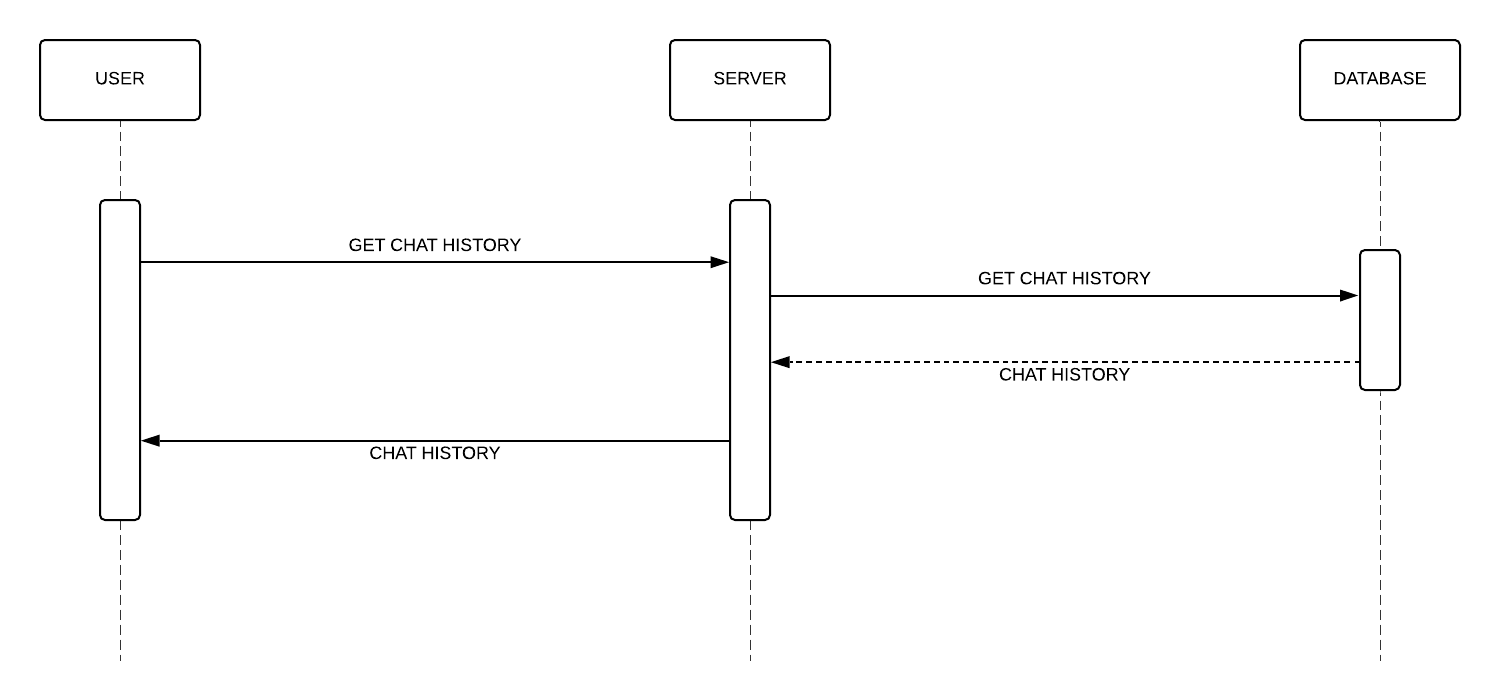


Figure: UML sequence diagram of the getMessage() request

When the Chat Screen is loaded then a request to get the chat history between the user and the receiver is made in the ComponentDidMount() method. The server on receiving the request then makes a subsequent request to the database and returns the history as a list of objects. Each chat object is consisting of the sender, the receiver, the translated text, the original text and the timestamp. The chat history is shown to the user as a list in the chronological order.

After requesting for the chat history, a WebSocket is also opened in this chat screen to allow the user to have real time conversation if the receiver also has an open WebSocket. This WebSocket is closed when the user moves back to the User Screen.

In this screen the user can also send and receive messages in the language of their choice. While creating a message in the preferred language depends on the keyboard settings of the mobile hardware used, receiving a message in the preferred language is handled by this project. The following figure shows the message flow for message translation in UML sequence diagram.

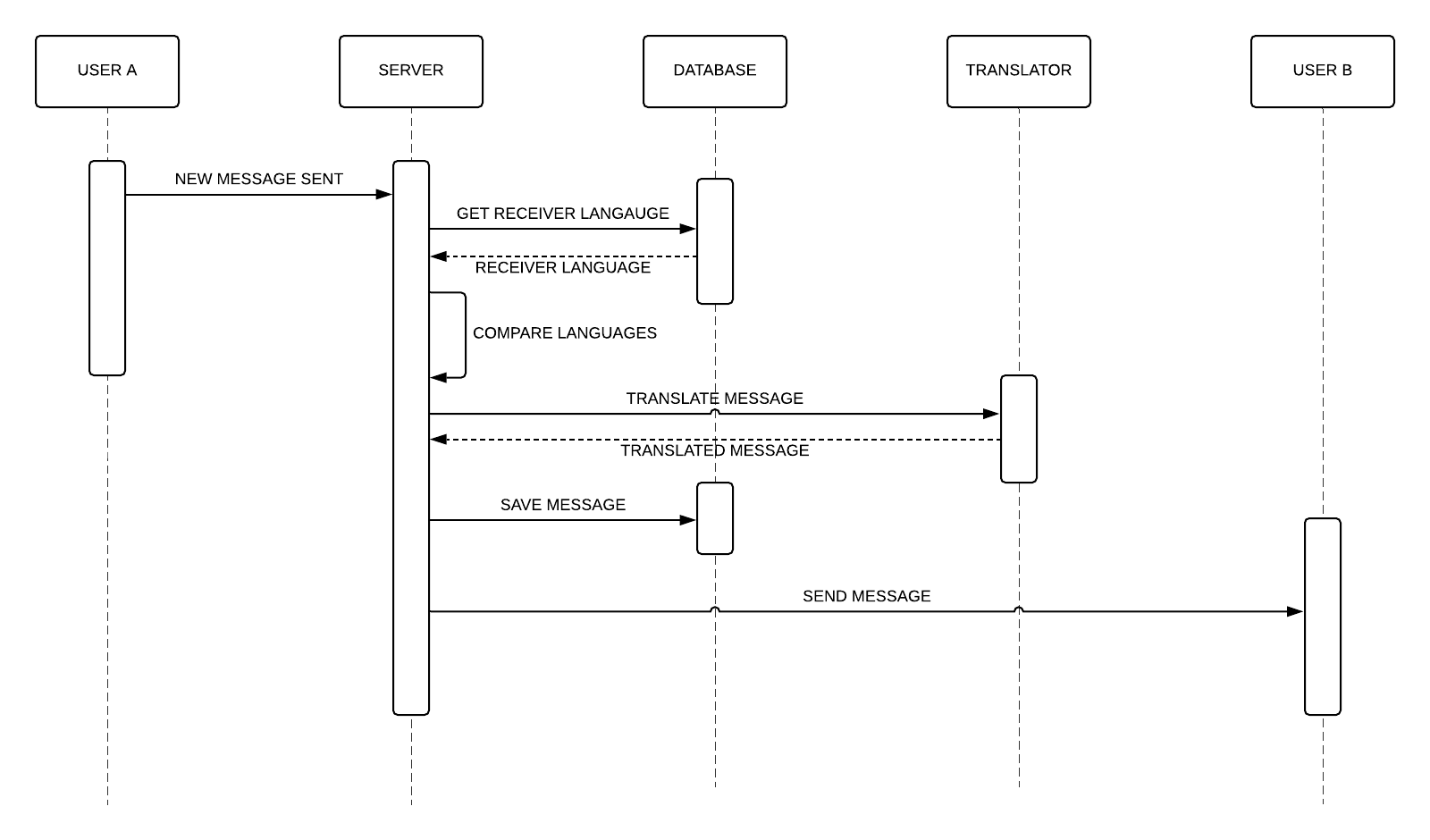


Figure: message flow during message translation

The above figure shows the message flow when a new message is sent by user A to user B. When the server receives the new message, it fetches the language preferred by the receiver. If the language of the sender and the receiver is the same then the server saves the message in the database and sends it to the receiver, i.e., User B. If the languages are not the same then the server sends the message to the translator API for translation along with the targeted language. This translation is done by a simple https request. On receiving the translated message, the server saves the message in the server and sends it to User B. The translator API in this project is Microsoft Translator API. However, the architecture of the server is made such that the API can be changed to any other API if need be by simply changing the https request.

The UI of the chat screen is done such that the received messages are stored in the left-hand side of the screen and the messages sent are stored in the right-hand side of the screen. If the user wish to view the original message sent, then they can simply long press the message and the translated message will be replaced by the original message. The text area where messages are written along with the send button is in the bottom of the screen. There is a back button in the left-hand side of the Header which allows the user to navigate back to the Users Screen.

In this screen the functional requirements fulfilled are the “A user must be able to view all the messages sent and received from a selected user”, “A user must be able to send and receive messages in the language they choose” and “A user should be able to view the original message sent if the received message is a translated one”.

### Profile Screen

The user of the App navigates into the Profile Screen from the Users Screen when they press the Settings button in the right-hand side of the header. In this screen the user can see its profile picture which is currently a fixed minion picture, thus providing the space for future implementations and change their email, username, preferred language and password. In this screen the user can also logout from the app. Changing the email, language and password updates the user data in the local database, while changing the password changes the user data in the Firebase database. Similar to *Forgot Password* when a user clicks the *Change Password* Firebase sends a change password link to the user’s email address via which the user can change their password. The languages provided in this screen for the user to select from is also the language list received from the Translator API via a GET request. When a user presses logout, the user is first unsubscribed to the topic in Firebase and then the user is logged out. They are then navigated to the Welcome Screen.

In this screen the functional requirement “A user should be able to reset their email, password and language” is fulfilled.

## Result