KLAIPEDA UNIVERSITY

Faculty of Marine Technology and Natural Sciences

Department of Informatics and Statistics

Encrypted Chat Application Using React-Native

Bachelor's Final Thesis

Author:	JNIN18AK student Kirill, Durov
Supervisor:	doc. dr. Julius, Venskus

Natural Sciences, Informatics

Klaipeda 2022

BAKALAURO (MAGISTRO) BAIGIAMOJO DARBO LYDRAŠČIO FORMA

Pildo bakalauro baigiamojo darbo autorius

.... ……………………..

(bakalauro/magistro baigiamojo darbo autoriaus vardas, pavardė)

Baigiamojo darbo tema:
(bakalauro/magistro baigiamojo darbo pavadinimas lietuvių kalba)

Patvirtinu, kad bakalauro baigiamasis darbas parašytas savarankiškai, nepažeidžiant kitiems asmenims priklausančių autorių teisių, visas baigiamasis bakalauro/magistro darbas ar jo dalis nebuvo panaudotas Klaipėdos universitete ir kitose aukštosiose mokyklose.

____

(bakalauro/ magistro baigiamojo darbo autoriaus ir parašas)

Sutinku, kad bakalauro baigiamasis darbas būtų naudojamas neatlygintinai 5 m. Klaipėdos universiteto studijų procese.

____

(bakalauro/ magistro baigiamojo darbo autoriaus ir parašas)

Pildo bakalauro baigiamojo darbo vadovas

Bakalauro/magistro baigiamąjį darbą ginti..

....(įrašyti - **leidžiu** arba **neleidžiu**)

....2022 ....

(data) (bakalauro/magistro baigiamojo darbo vadovo vardas, pavardė ir parašas)

Pildo katedros, kuruojančios studijų programa, administratorius (sekretorius)

Baigimasis darbas įregistruotas katedroje 2022 ....

(data)

Laima Brazdeikienė....................

(katedros sekretorės vardas, pavardė ir parašas)

Pildo katedros, kuruojančios studijų programą, vedėjas

Bakalauro/magistro baigiamąjį darbą ginti...

(įrašyti - **leidžiu** arba **neleidžiu**)

2022........................ doc. dr. Mindaugas Kurmis...

(data) (katedros vedėjo vardas, pavardė ir parašas)

Recenzentu(-ais) skiriu ....…………………………

....

(įrašyti recenzento(ų) vardą, pavardę)

......2022................................ doc. dr. Mindaugas Kurmis..

(data) (katedros vedėjo vardas, pavardė ir parašas)

Acknowledgements

I would like to express my most sincere gratitude to my supervisor, doc. Dr. Julius Venskus, for his constant support, help, and the knowledge I have gained from him. Without his help, I am sure, the quality of the final thesis would have been much lower. He was able to solve my most difficult problems and answer any questions I had throughout the entire project. Julius Venskus provided me with many resources to expand my knowledge, which played a key role in the writing of this work. Thanks to him my development knowledge has broadened greatly, and the skills I gained while writing this project provided me with clarity and understanding of many algorithms, as well as a deeper understanding of architecture and dependencies between Front-end and Back-end. His contribution to this work is invaluable.

SUMMARY

Nowadays, modern society is unimaginable without means of communication. Everyone always has a cell phone, tablet, smart watch, etc.

People spend most of their time using their gadgets. What they do is reading news, watch interesting videos or funny pictures. Also people communicate with each other through social networks and messengers. Messenger is a software application allowing two users to exchange text messages or any other alternative information in real time.

Undoubtedly, a huge advantage of messengers is the storage of messages, and at any time it is possible to find the necessary information. This also leads to irrelevance of telephone calls, and it is likely that they will be replaced by messengers in the near future.

The relevance of work is in encryption, which uses private and public keys. In this regard, no one from the outside will not be able to obtain information transmitted to the server or the client.

Contents

List of Figures

List of Tables

Acronyms

AOT	Ahead-of-Time.
API	Application Program Interface.
AUTH	Authentication.
CLI	Command Line Interface.
CSS	Cascading Style Sheets.
DOM	Document Object Model.
ES6	European Commission Management.
GDPR	General Data Protection Regulation.
HIG	Human Interface Guidelines.
HTML	Hyper Text Markup Language.
ID	Identifier.
IDE	Integrated Development Environment.
JIT	Just-in-time compilation.
JS	Java Script.
JSON	JavaScript Object Notation.
JSX	Java Script XML.
NoSQL	Non-Structured Query Language.
NPM	Node Package Manager.
PC	Personal Computer.
SRC	Source.
UI	User Interface.
UML	Unified Model Language.
VSC	Visual Studio Code.
XML	Extensible Markup Language.

Introduction

In the modern age of informatization of society and the close implementation of information technology in human social life, one of the first places is the need to protect information and sensitive user data. Today, such data falling into the hands of a cybercriminal can cause enormous financial and moral damage to the user, and for the corporate segment the loss of this information can have a detrimental (if not critical) impact on the financial and economic performance of the enterprise.

On the other hand, with constantly increasing speed of access to public networks, the problem of speed of information processing by services is not in last place.

The general increase of attention to security issues, along with competition among messengers, which is taking place against the backdrop of the scandals related to the sale of user data by Facebook and the blocking of the Telegram messenger in a number of countries, used by the Telegram's developer to promote the idea that its system is secured, forcing developers to implement additional security measures, such as end-to-end encryption. protection, such as end-to-end encryption of transmitted data.

"Secret" chats are also supported in Telegram, but in normal Cloud chat mode, which is used by default, the data is "visible" to Telegram servers in the public view. The messenger's PR campaign was built on the assurances of its owners that neither secret services, nor intelligence agencies would be able to read private messages, and was held for an English-speaking audience under the slogan "Taking back our right to privacy". According to Pavel Durov, "From day one day, we did not give governments or third parties a single byte of private data." Thus, the Telegram messenger's security is ensured not by technical measures, but rather by the company's policies. However, in August 2018, Telegram was forced to make changes to the The messenger's privacy policy as part of the new European GDPR (General Data Protection Regulation), by adding a clause about the possibility of disclosing the IP address and phone number of the subscriber on the basis of a court decision, confirming that the subscriber is suspected of terrorism.

This reflects another trend in the security of messengers due to increased regulation by the state, seeking to de-anonymize, if it's possible. communications in case they need to be monitored. In this line of thought is The announcement in August 2018 that WhatsApp was abandoning the the endpoint encryption technology currently in use. A new version of the app, which implements weakened encryption to as required by U.S. authorities, will be released in 2019.

Messengers use their own transmission protocols data transfer protocols, with message encryption usually done at the application layer, and then the encrypted data is embedded in the transport protocol. As a transport protocol, it is usually (but not necessarily) SSL/TLS is usually (but not necessarily) used as the transport protocol, which has become the de facto standard for all secure web communications. However, The implementation of the latter on mobile platforms is far from being perfect.

To protect transmitted data, messengers, as a rule, use end-to-end (E2E, end-to-end or end-to-end) encryption. That means that cryptographic keys are generated and stored on endpoint device (the user's device) and not on the servers of the instant messaging system. Therefore, neither one but the recipient, nor even system's server, will not be able to read the contents of encrypted messages. On the other hand, the correspondence will not be available to the the subscriber himself, if he switches to another device. Therefore synchronization of devices or recovery in case of loss device (i.e. gain access to the correspondence archive) together with the using endpoint encryption is not possible without without escrowing the private key outside the device. As I see it, this is what ideal privacy and security of correspondence looks like.

In this thesis the Chat Application was implemented to be used with the help of popular frontend technology, React Native and Firebase hosting services, which offers cost free backend for the chat message and data. An instant messaging application uses cryptographic algorithms for encrypting correspondence and secure communication protocol between client and server.

The aim of the thesis is to design and create Encrypted Chat Application Using React-Native framework for mobile handheld devices.

For this aim, the following objectives should be achieved:

1. Review, analyze, summarize, and select technology stack for handheld mobile application development.
2. Design Encrypted Chat Application Using selected framework for mobile handheld devices.
3. Create and validate Encrypted Chat Application for mobile handheld devices.

1 Review of mobile application development frameworks

In this chapter, all technologies of the Chat App are mentioned with the background.

1.1 Chat Background

In this section, the background history of the chat is discussed.

1.1.1 SMS

Text message is the short electronic message which is called as SMS. It allows user to send and receive small text messages with little cost. It is a more secure and reliable way to send and receive short message that consists of around 160 characters. For SMS does not require an App and the internet and it is available on all mobiles while MMS (multimedia services) require the internet or a WIFI service. MMS is the enhanced form of the SMS, which enables users to send and receive video and pictures images. SMS is being used for the last 20 years on GSM standard as communication technology enhanced to 3G, 4G and 5G SMS developed to MMS which allows the user to send and receive video and pictures messages. While Chat App is a free of cost, it occurs via a desktop or a mobile app.[1]

1.1.2 Instant Messaging

Instant messaging platform was created in the 1960s by MIT allowing up to 30 users to log in at a time. Instant messaging is a one-to-one conversation-based platform which allows user to connect with another person´s device for sharing the text and picture image.[2] In other words, instant messaging is a sort of real-time online chat which is enabled via a special software application. Since technology has improved the instant messaging has gained popularity and it is used in every life. The downside of instant messenger or IM is that the other user must have the same device or software program for the communication.

1.1.3 Chat App

The word communication has been derived from the Greek word, “Communicate”, meaning the exchange or sharing of ideas or expression. Chat was created at the University of Illinois in 1973. Chat App is a platform that enable messaging either one-to-one or to group members usually occurring in a chat room where multiple people can join and shares their interests.[3] In other words, Chat App is also called “Real- Time” chat because it is on-going message service which processes the hundreds of messages between the sender and the receiver without any delay.

Some examples of popular Chat Apps are WhatsApp, Viber, LINE, Skype, WeChat, IM, tt„ C, Tango, N, kik, TALK, talk and ebuddy.

Chat App itself is different than the old tradition instant messaging applications because it requires only a desktop and a laptop, for example, Yahoo messenger and Hotmail messenger, while Chat App is enabled via mobile apps on smart phones such as Android, IOS as well as a desktop.[4]

Benefits of Chat

Benefits of chat includes many companies and schools rely on live chat today because it has the following benefits.

1. **Faster support**: It is easy to approach students, colleagues and customers by live chat. Generally, it takes less time to solve the customer issue when using live chat.

2. **Real-time text examine**: It is one of the benefits between customer and agent. A customer sends the request for a solution and an agent views and think about the solution of the issue and replies to the customer immediately.

3. **Instant customer feedback**: Customer can rate the company service right after using the chat service.

4. **No waiting queues**: Customers can reach the company agent immediately.

5. **On a browser site**: The customer and agent conversation take place on the website. During a chat, the agent can see where the customer is currently looking the issue so the agent can easily guide the user by copying the link or screenshot image.

6. **Purchasing solution**: On online shopping point of view, is one of the important advantages of live chat. A customer can ask the agent about the product cost in the middle of online shopping.

7. **File transfer**: A customer can provide file, document and images of the questioned issue upon asking by the company agent.

8. **File transfer**: A customer can provide file, document and images of the questioned issue upon asking by the company agent.

9. **Mobile Messaging Integrations**: Now, companies add Live Chat messenger like Viber, WhatsApp and Facebook to a company website. Customers are connected with the company´s central live chat system via advance live software programs.

10. **24/7 Support Service**: A company is available for the customers 24/7 without any time zone restrictions. Customers can talk to a chatbot even out of office hours.

11. **Less costs**: Live chat is cheaper than the traditional phone call for the customer to access the company agent for questioning and solving issues.

Disadvantages of Chat

Disadvantages of Chat are:

1. **Internet Access**: The user needs constant access to the internet during a chat conversation with colleague or company´s help desk. Also, users have internet access difficulties in developing countries.

2. **Communication difficulties**: Some users have difficulties in chat communication with the help desk, particularly elder and disable people.

3. **Virus Risk**: Companies can have virus risks attached files, pictures or images when transferring them from the user to company during chat.

4. **User Unfriendly**: Live chat is unfriendly for the novice computer users.

5. **Unfriendly**: It is observed unfriendly on mobile platforms.

1.2 Front-End Framework

In order to reach more users, I decided to develop an app that runs both on Android and iOS.[5] One of the solutions would have been to follow the classic native approach, creating two different apps: one written in Java or Kotlin (Android) and the other in Swift or Objective-C (iOS). This approach has its advantages, such as the use of official tools developed for that platform, allowing them to be definitely supported as long as the operating system itself remains in development. In addition, this approach enforces Human Interface Guidelines (HIG) and allows the user to have a familiar experience with the application's interaction (native UX/UI). However, this strategy is pretty time-consuming, since you have to write the same code twice to implement the same functions.

1.2.1 Cross-Platform development

Cross-platform development, on the other hand, solves this problem by providing powerful tools that use a single technology stack to develop applications for both Android and iOS. Cross-platform development is an approach that allows the development of a single code base for multiple platforms.[6]

This approach has several advantages:

1. **Ease of learning** Developers use a single technology stack and do not need to learn two different technologies. Although each platform has its own features and limitations, code can be written in one language and the GUI can be developed using one tool.

2. **Reusable code** Part of the code base can be reused from one platform to another. The code is common to both applications, such as business logic, data access, and networking, can be separated from the user interface design.

3. **Ease of maintenance** Maintenance is not done on each platform separately. Because common code is written once, there is no opportunity to introduce inconsistencies between platforms.

A common potential issue with cross-platform development tools is that the developer cannot use familiar libraries. Indeed, some libraries, such as those for networking, become common to both Android and iOS. And a possible problem is that some features available only in Android or iOS libraries may have no alternatives in cross-platform development tools. In addition, there are a number of disadvantages that are described below, such as a worse user experience, difficult integration of local preferences and notifications, inconsistent user interface, less flexibility, etc. These disadvantages vary across cross-platform development tools.

Therefore, I have analyzed several cross-platform development programs and frameworks to compare them and choose the one that best meets my needs for the task at hand. In the following sections, I will choose and explain the best tools or frameworks for cross-platform mobile development, both because of the number of features they offer and their ease of use: Xamarin, Ionic and React Native.[7][8] I will compare them according to several criteria (language stack, compilation method, performance, ease of deployment, library binding level, GUI, background code execution, and price). I will then explain why I chose React Native development tool.

1.2.2 Xamarin

Xamarin is a cross-platform development tool for creating native apps for Android and iOS, as well as Windows and Mac apps using a single common C Sharp codebase and native libraries wrapped in a .NET layer. Xamarin has two approaches to building apps: Xamarin.Android/iOS and Xamarin.Forms. The first approach allows the developer to design layout files (AXML) for Android and storyboards for iOS using Visual Studio tools. All native frameworks are readily available. The graphical user interface is specific to each platform, while code based on business logic can be common to Android and iOS projects. The second approach allows the developer to design a user interface for two platforms at once.[9]

1. **Language Stack** As mentioned above, Xamarin uses C Sharp and the .NET framework for all mobile platforms. A developer can translate any code written in Java, Swift, and Objective- C into C Sharp. A developer can also use .NET libraries to replace the native open source libraries available for Android and iOS.

2. **Compilation** C Sharp compiles into some bytecode rather than the native code of the target processor. C Sharp can be compiled either JIT (Just-in-time) or AOT (Ahead-of-time). Both AOT and JIT can be used in Android. However, JIT compilation does not apply to iOS. AOT-compiled code should run faster. Also, in Xamarin, code can be compiled to 64-bit target code. Generally, 64-bit code runs faster and takes up more memory.

3. **Performance** Applications created using the Xamarin approach behave as native. In fact, their cross-platform capabilities focus mainly on sharing business logic rather than codebase. They use native UI controls and leverage platform-specific hardware acceleration. Xamarin.Forms, in contrast, focuses on broad code sharing with less platform-specific behavior. With this approach, almost all source code can be reused. This reduces code performance compared to the first approach.

4. **Ease of deployment** When a developer writes new code, the faster they can observe the result, the more comfortable they are to develop. In Xamarin, the developer writes the code, builds it, deploys it on a simulator, and then observes the result. Rapid Deploymentis also available on Android in debug mode, which restarts the entire app, but without re-deploying the entire APK.

5. **Library binding** Xamarin is very good at binding libraries and has pretty complete binding of standard Android and iOS libraries. The developer can use the libraries as if they were writing in their native language (i.e., Java or Objective-C).

6. **Graphical User Interface** The Xamarin approach is to use a platform-specific user interface. Indeed, a developer can create a UI with native UI layouts and controls. This approach provides a native look and feel, but is time-consuming. Xamarin.Forms, by contrast, automatically maps each page and its controls to platform-specific UI elements at runtime. This second approach greatly speeds up the development process, but it loses the native look and feel.

7. **Background execution** Xamarin's background execution allows the developer to use a familiar set of platform native functions to execute code in the background. The developer has to write different code because of all

the limitations of the platform for both Android and iOS. However, he is not limited to using only the more limited features of the platform.[10]

8. **Pricing** Xamarin is an open source product. It is available for free for non-corporate projects with up to five developers. Professional and Enterprise licenses with more features are also available. If the development requires more features, then you will have to pay to extend the license. The price varies depending on the requirements for the implementation of a particular task.

1.2.3 Ionic

Ionic is a framework for creating mobile, web, and desktop applications (i.e., hybrid applications) with a common code base and open web standards, using HTML5 and AngularJS. Unlike Xamarin, Ionic does not use native widgets. Instead, it simply displays a web page written in HTML that mimics the design of native widgets.[11]

1. **Language Stack** The Ionic language stack uses web technologies such as HTML5, CSS, and JavaScript to write and run applications. It also uses the Apache Cordova framework to access the capabilities of each device, such as sensors, data, network status, etc. Ionic's main programming language is TypeScript (or plain JavaScript), which improves code quality. Indeed, this language helps to detect and eliminate errors when typing code.

2. **Compilation** The JavaScript compilation is also compiled into bytecode rather than the native code of the target processor. Ionic uses JIT compilation for Android and WKWebView (the platform browser) by default for iOS. While JIT compilation usually cannot be run on iOS, WKWebView provides JIT conversion of JavaScript code to machine code, which improves rendering performance. In addition, the current version of Apache Cordova supports 64-bit mode on iOS, but not Android.

3. **Performance** Ionic uses web technologies rather than native components to render the application and tries to recreate native behavior. This approach significantly reduces speed, so Ionic cannot achieve performance results comparable to the other two cross-platforms on complex and saturated applications.

4. **Ease of deployment** With Ionic, the testing process is very fast. Indeed, the use of web technologies such as HTML and CSS allows the developer to run the mobile app directly in the browser. This means that if the developer changes some web files, only the necessary build steps are performed, and the app is automatically updated as the code changes.

5. **Library binding** Unlike Xamarin, in Ionic you can't just bind any arbitrary code the developer wants. Indeed, in order to create a plugin, he needs to write native adapter code. Moreover, Ionic has its own layers of abstraction between the standard library and the JavaScript application, which was not the case in Xamarin. This means that some OS features are missing.

6. **Graphical User Interface** Ionic's graphical user interface does not use native widgets at all. The UI is created in HTML and CSS, and Ionic provides a set of Angular components to mimic native widgets and give the application a native look.

7. **Background execution** Ionic has a Cordova plugin for background execution, which provides no means of actually doing background work, but only features such as preventing the application from interrupting if it's in the background.

8. **Pricing** Ionic is also an open-source product. You can use a companion platform called Ionic Pro, which offers additional features such as days of error history tracking and collaboration tools.

1.2.4 React Native

React Native is a framework developed by Facebook to create apps for various platforms, such as Android and iOS, using JavaScript and React.JS. However, React Native works more like Xamarin. Indeed, it uses a special template language to create a GUI, and then creates native widgets. [12][13]

1. **Language stack** The React Native language stack combines the advantages of JavaScript and React.JS (a web framework). The developer can also write modules in Java, Objective-C or Swift when he needs to.

2. **Compilation** The React Native compilation uses JIT compilation for Android, but interprets JavaScript code for iOS. Also, 64-bit mode is not currently supported on Android, but it should be on iOS since Apple enforces it.

3. **Performance** The performance of React Native is close to native. Indeed, code components directly access the native API. The common code base, that is, about 80 percent of the JavaScript code, can be used on the two platforms. The rest are native modules for Android and iOS, written in Java and Objective-C to optimize performance for complex operations.

4. **Ease of deployment** React Native's ease of deployment allows the developer to use either the classic approach, live reloading or hot reloading. The

difference between live and hot reloading is that hot reloading does not involve completely rebooting the application. Indeed, this approach replaces only the changed parts of the application and preserves the state of the components.

5. **Binding Libraries** As with Ionic, React Native cannot simply bind arbitrary code that the developer wants, and has its own layers of abstraction between the standard library and the JavaScript application, resulting in partial support for OS functions.

6. **Graphical User Interface** React Native modules interact with native Android and iOS UI controllers. However, not all native widgets are available. Even when native widgets are used, the UI elements don't look native.

7. **Background execution** React Native only supports background code execution on Android, but not on iOS.

8. **Pricing** React Native is a fully open-source product, and its libraries can be used for free.

Table 1: Comparison of cross-platform mobile development tools

	Xamarin		**Ionic**	**React Native**
Language stack.	C Sharp		JavaScript, Type-Script, HTML, CSS	JavaScript, Java,Objectice-C, Swift
Compilation				
Android	JIT or AOT		JIT	JIT
Apple	AOT		JIT (WKWeb-View)	Interpreter
64-bit mode				
Android	Supported		Not supported	Not supported
Apple	Supported		Supported	Supported
Deployment	Fast deployment (Android) and classical approach (iOS)		Deployment in browser	Live and hot reloading
UI widgets	Native		HTML, CSS	Native (not all available)
Library binding	Good at bindings and full binding of standard libraries		Moderate at bindings and partial binding of standard libraries	Moderate at bindings and partial binding of standard libraries
Background execution	Usual means		Plugin with few features	Supported only on Android
Pricing	Open source but eventual additional costs		Open source but eventual additional costs	Open source
	.Android/iOS	**.Forms**		
Performance	Close to native	Moderate	Moderate	Close to native
UI look-and- feel	Close to native	Moderate	Close to native	Low
Code reuse	Business logic, net- working	Almost the entire code	Almost the entire code	Up to 80% of JavaScript code

1.2.5 Final Choice

The table compares different cross-platform software or frameworks for mobile development according to the criteria defined above. Based on the information above, I decided to use React Native, and now I will explain why I chose it.

React Native uses modern and trendy features, such as hot reloading and instant updates. Also, the main language is JavaScript, which I studied at university for quite a long period of time. This approach will help us learn the technology faster and implement the necessary components for the task at hand. At the same time, it remains close to native, although the user interface is quite low level.

Unlike the other tools mentioned above, Xamarin is primarily focuses on sharing business logic, networking and data access, rather than the codebase. This means that the GUI of each platform (i.e. Android and iOS) has to be developed independently, which can take a long time in the development process. Limited access to open-source libraries also plays valuable role. Choosing React Native we are provided with the full functionality for free, which cannot be said about the others. Since the framework is fairly young, it is actively added to the libraries and is often updated. If we will choose Ionic, applications will look almost like native apps, but creating user interfaces in HTML and CSS can be painful and significantly reduce performance.

1.3 Database

Each modern web application uses one or more databases to store information. Databases provide tools for organizing, adding, searching, updating, deleting, and performing calculations on data. In most cases, web application servers communicate directly with job servers. In addition, each internal service may have a corresponding database isolated from the rest of the application.

There are two types of databases: **SQL** and **NoSQL**.

SQL stands for "Structured Query Language". It was invented in the 1970s. SQL databases store data in tables that are linked by common keys. Such keys are usually represented by integers.

NoSQL stands for "not only SQL" and is a newer set of database technologies. It was designed to handle the very large amounts of information that can be generated by large-scale Web applications. Most SQL variants don't scale horizontally very well and can scale vertically only up to a certain limit.

The design module will store a very large number of items in the "messages" section. In addition, to put the application in perspective, there is a chance of storing a large number of user profiles containing username, email and images encoded in base64. Moreover, all data are not deeply related to each other. Therefore for this project we will NoSQL database will be used.[14]

1.3.1 Comparison of database solutions

Three different databases were considered: DynamoDB from Amazon, Firebase Realtime Database from Google and MongoDB from MongoDB Inc.

All of them have advantages and disadvantages. The first two were released in 2012 and the last one was released in 2009.

DynamoDB is a hosted, scalable database service from Amazon whose data is stored in the Amazon cloud. [15]

Firebase is a cloud-based, real-time document storage service. iOS, Android and JavaScript clients use a single real-time database instance and automatically receive updates with the latest data. [16]

MongoDB is one of the most popular document stores available both as a fully managed cloud service and for deployment on a self-managed infrastructure.[17]

All databases have primary database models of document repositories. In addition, DynamoDB has a key-value store.

DynamoDB and Firebase have commercial licenses, while MongoDB is an open-source project. In addition, the first two databases are only available as cloud servers. MongoDB can be run locally on Linux, Mac OS X, Solaris and Windows.

DynamoDB supports the following programming languages: .Net, ColdFusion, Erlang, Groovy, Java, JavaScript, Perl, PHP, Python, Ruby.

Firebase supports Java, JavaScript, Objective-C.

MongoDB has the most supported programming languages: Actionscript, C, C #, C++, Clojure, ColdFusion, D, Dart, Delphi info, Erlang, Go, Groovy, Haskell, Java, JavaScript, Lisp, Lua, MatLab, Perl, PHP, PowerShell, Prolog, Python, R, Ruby, Rust, Scala, Smalltalk, Swift.[18]

DymanoDB provides access rights for users, and roles can be defined through AWS Identity and Access Management (IAM). Firebase provides access control based on authentication and database rules, and MongoDB provides greater access rights for users and roles.

Table 2: Comparison of the databases

Name	Amazon DynamoDB	Firebase Realtime Database	MongoDB
SQL	No	No	Read-only SQL queries via the MongoDB Connector for BI
License	Commercial	Commercial	Open source
Cloud-based only	Yes	Yes	No
Data scheme	Schema-free	Schema-free	Schema-free
Server operating systems	Hosted	Hosted	Linux, Mac OS X, Solaris, Windows
APIs and other access methods	RESTful HTTP API	Android iOS JavaScript API RESTful HTTP API	proprietary protocol using JSON
Server-side scripts	No	Limited functionality with using 'rules'	JavaScript
Implementation language	No	No	C++
Triggers	Yes	Callbacks are triggered when data changes	Yes
MapReduce	No	No	Yes
Consistency concepts	Eventual Consistency, Immediate Consistency	Eventual Consistency, Immediate Consistency	Eventual Consistency, Immediate Consistency
Foreign keys	No	No	No
Transaction concepts	ACID	Yes	Multi-document ACID Transactions with snapshot isolation

1.4 Firebase Cloud Platform

Firebase is a database, not a backend. It is currently possible to use Firebase without the server part, but that can cause a lot of problems in the future. Firebase is a NoSQL database with all the advantages and disadvantages. This is why Firebase does not support complex SQL queries. NoSQL databases provide a mechanism for storing and retrieving data which is different from the table relationships used in relational databases. To choose a solution for storing data, you need to analyze the data and its structure. Firebase has the following disadvantages:

– Firebase scope is much smaller than a NoSQL solution;

– Firebase severely limits you in getting data and, if necessary, writing data to several places at once;

– Not all data structures are convenient to work with in firebase;

– It does not allow you to make connections to data elements;

The solution allows you to start development quickly and also has many advantages:

– Firebase has many options, such as Realtime and Firestore, which are both cloud-based;

– NoSQL databases that are flexible and scalable in size;

– All data is stored in JSON format and is synchronized for all connected clients in real time;

– Realtime also gives developers offline access and real-time updates, allowing them to work on responsive applications without an Internet connection;

– The service lets you deploy web applications in seconds;

– It provides static hosting activities, using a secure connection using SSL and CDN network infrastructure;

It's great for beginners who want to see if the platform is good enough for their product, and don't want to pay for all services in advance. For those who want to estimate the total price to pay for an individual plan, there is a price calculator that will make this process easier.

Firebase has well-prepared technical documentation that makes the services offered easier to work with and more accessible to developers. The developer can find all the necessary information about integrations, accessibility and supported technologies. Moreover, there are about 1.5 million applications based on Firebase. This means that the community around the product and the number of resources will only benefit developers who are trying to find the answer to any problem.

Finally, the ease of integration and quick customization. The out-of-the-box APIs that platform, allow you to add new functionality in just a few clicks. Firebase requires little or no technical knowledge to get started on your product. A simple user interface allows you to implement features such as authentication in your application without major problems. With Firebase, there's no need for complicated configurations, so almost anyone can set up an application - web or mobile.[19]

1.5 Expo SDK

Expo is a framework and a platform for universal React applications. It is a set of tools and services built around React Native and native platforms that help you develop, build, deploy, and quickly iterate on iOS, Android, and web apps from the same JavaScript/TypeScript codebase. It's possible to build most apps without ever needing to write native code, provided that you have a comprehensive set of APIs exposed to JavaScript.

This is important because with React Native you can always drop down to native code.[20]

1.5.1 Expo advantages

1. A convenient utility Expo CLI that opens in a browser and helps to check the app status, devices it runs on, scan QR code or send the link via email to open the app in Expo client, switch the production / development mode and publish your app on Expo server.

2. Expo client is an app that is installed from Google Play and Apple Store on your phone. It allows opening projects during development without build via XCode or Android Studio. With Expo client, you can send your app to others for review, which is very useful when testing as you can see all changes in code in Expo client without creating apk or ipa files.

3. Expo SDK offers the collection of ready solutions, such as working with the device accelerometer, camera, notifications, geolocation, etc. You can see them in the section SDK API Reference. Expo team regularly updates SDK with new solutions.

4. Over the Air - very handy Expo feature for updating your app over the air without repeated deployment on Google Play or Apple Store. When users open your app, it will automatically update new changes in JavaScript code. However, such things as app icon, its name and other settings are not updated via OTA. You can read about it in more detail in the Limitations section.

5. You can develop apps for ios without macOS with ios device and test them with Expo client.

6. Automatic management of certificates and app signatures.

1.5.2 Expo disadvantages

1. You can't add native modules written in Objective-C, Swift, Java, Kotlin.

2. You can't use packages with native languages that require linking.

3. The app has a big size as it is built with all Expo SDK solutions, even those you don't use. An application with Hello World weighs 25 MB.

4. Often everything works well in Expo client during testing, but certain problems may occur in a standalone app.

But nevertheless its still an amazing SDK for building simple apps such as my Chat App where you don't need to interact that much with the devise. Otherwise Expo won't fit you and could cause some issues. For my project Expo was more than enough.

1.6 Visual Studio Code

Visual Studio Code is the part of visual studio family which is developed by Microsoft in November 2015. It is based-on Electron framework which is used for Node.js (node java script). It is written in TypeScript, JavaScript and CSS.

1.6.1 Features

Features of Visual Studio Code are as follows:

1. It is open-source and Freeware text editor for private and commercial purposes.

2. It is cross-platform source code editor debugger.

3. It supports many different programming languages while only proper installation of the extension is required for React Native, Java, JavaScript, C++, C Sharp, Python etc.

4. GitHub is built-in.

5. Products availability as it explains the definition and show opening and closing of the brackets etc.

6. It provided unique customizations.

7. Provides Visual Studio Keymap extension for using Key binding.

8. It provides in the portable mode that means it keeps data and settings in the same location of installation is possible even on a USB drive.

9. It is available in many different language services.

10. It is also available in Remote Development mode.

The choice of Expo was not difficult, because the choice in this area, one might say, was not provided. Nevertheless Using Expo when developing mobile React Native mobile application development, it saves you from solving some technical issues, and also allows you to use additional functionality during development and after the creation of the application.

Below I will highlight the key points that for me played a key role in choosing this platform:

- No native development tools (such as Xcode for iOS and Android Studio for Android) will be required to compile JS files into executable application files (.apk/.ipa). The compilation of JavaScript code into executable files will take place on the Expo server. Once completed, the installation file can be downloaded from the server to your computer. This advantage is very important, at least because Xcode, which is necessary to build applications for IOS, can only be installed on MacOS devices. Although the development was done on MacOS, installing Xcode takes a lot of time and free space on the hard drive, which was a disadvantage in my case;
- Expo provides a convenient interface for debugging code during code development. All errors and logs can be displayed in the console of the Expo client application or in the browser console when debugging through it. Also, functionality is provided to run the application under development Application can be run on mobile device emulators or on a real mobile devices when connected to the same wifi network;
- Expo provides its own interface for interacting with device features: camera, contacts, local data storage, etc. Therefore, there is no need to load various separate libraries for these tasks;
- Another feature of the Expo is updating application installed on the device in real time: after build and publish the new version, all applications previously installed users will be updated. This is a very powerful feature, it It solves one of the major problems in mobile development - compatibility between versions. Without the ability to update applications in real without the ability to update applications in real time, when new versions are released for it, you always need to implement compatibility with old ones, since users don't always immediately update installed apps right away.

1.6.2 Virtual DOM and Real DOM

DOM stands for Document Object Model, which is programming language platform interface for HTML and XML documents. It is a tree-like structure which enables dynamically accessing and managing the elements. It has three main types:

1. **HTML DOM**: Standard model for HTML documents.
2. **XML DOM**: Standard model for XML documents.
3. **Core DOM**: Standard model for all types of documents. In addition to, DOM have two types Real DOM and Virtual DOM. React Native uses Virtual DOM.

Real DOM

- Its manipulation or updates is slower and costly.
- It is heavy weight code.
- It can update the HTML in the browser.

Virtual DOM

- It is not created by React, but it is used as for free.
- It is a copy of Real DOM.
- Its manipulation or updates is faster and easier.
- It is light weight code.
- It cannot update the HTML directly in the browser.

1.7 ES6

The European Computer Manufacturers Association (ECMA) created ECMAS to standardize JavaScript in 2015. ECMAScript (ES6) is a script language. ES6 is generally used for the client-side scripting also server-side application, especially with Node.js.

ES6 offers you the possibility to write the code more readable and in a more advance way. In addition, ES6 introduces many advance features like arrows, classes, template strings and class destruction.

1.8 Encryption

An important part of the application is encryption. In this context encryption is the main part of the client side of the application.

Encryption is a redundant transformation of information in order to hide it from unauthorized persons, as well as to give authorized users access to data. The main purpose of encryption is to keep transmitted information confidential.

Encryption is divided into two types - symmetric and asymmetric. The first uses only one key which encrypts and decrypts information. With the second type, there are 2 keys - the public key, which is used for encryption and transmitted through an unprotected channel and the secret key, which is used to decrypt and its value knows only the decryptor. [21]

The RSA (Rivest-Shamir-Adleman) algorithm, the most widely used asymmetric algorithm embedded in the SSL/TLS protocol used to secure communications over a computer network, was used to encrypt the user's incoming and outgoing messages. It is a public-key cryptographic algorithm that relies on the computational complexity of factorizing large integers that are the product of two large prime numbers. Multiplying two large prime numbers is easy, but the difficulty of determining the of the initial numbers from the total factors public key cryptography.

For encryption, a simple exponentiation operation modulo N is used. For decryption it is necessary to calculate the Euler function of the number N, for this purpose it is necessary to know the expansion of the number N into prime factors (this is the factorization problem). In the RSA cryptographic system, the public and private keys consist of a pair of integers. The private key is kept secret, while the public key is shared with another participant, or published somewhere.

The time it takes to factor the product of two sufficiently large prime numbers is considered too much for most attackers. The cryptosystem was the first suitable for both encryption and digital signature. This algorithm is used in a large number of cryptographic applications. [22]

1.9 base64 (reversible coding algorithm)

base64 is an encoding scheme by which an arbitrary sequence of bytes is converted into a sequence of printed ASCII characters. Only uppercase and lowercase Latin characters are used - symbols (A-Z, a-z), numbers (0-9), and the symbols "+" and "/", with "=" as a special suffix code. [21]

In order to convert data to base64, the first byte is placed in the highest eight bits of the 24-bit buffer, the next in the middle eight, and the third in the lowest significant eight bits. If less than three bytes are encoded, the corresponding bits of the buffer are set to zero. Then every six bits of the buffer, starting from the highest, are used as indices of the string "ABCDEFGHIJKLMNOPQRSTUVWXYZ-abcdefghijklmnopqrstu vwxyz0123456789+/" and its characters, to which the indices point, are placed in the output string. If only one or two bytes are encoded, the result is only the first two or three characters of the string, and the output string is complemented by two or one "=" characters. This prevents additional bits from being added to the reconstructed data. The process is repeated over the remaining input data.

In my application, I will use base64 to store the user account pictures. The main purpose of this approach is a way to make sure that no data is lost or altered during the transfer.

2 Design of Encrypted Chat Application

This section details the most relevant parts of application development, purpose and algorithms.

2.1 Main idea and purpose of application

The main idea of the Encrypted Chat Application is pretty basic - privacy. Nowadays, privacy is the most discussing issue as everyone wants to be private and untraceable. From one point of view we can say that it's impossible to stay anonymous in 21st century and it's true. But on the other hand you can take some measures to be private from some undesirable persons, such as hackers who can be "Man In The Middle" and intercept some of your data. To prevent such "MITM" attacks our chat application equiped with "RSA" encryption algorythm which prevents anyone from monitoring your text conversations. Implemented end-to-end encryption make you conversation private and inaccessible for anyone except you and your interlocutor.

2.2 Stakeholders, main users

This application comes in handy for anyone who is worried about data security when communicating. For example, you can create chats between company employees to protect business or customer data from leaks. You can use chat rooms for personal communication with friends or relatives. You can have important conversations with partners.

2.3 Functional requirements

Table 3: Functional requirements

Name	Description	Priority	Actor
User Registration	Functionality for user to create account using email address	High	User
Login	Functionality for user to get access	High	User
Logout	Functionality for user to close session	High	User
User List	Functionality to see list of registered users	High	User
Send Message	Functionality that send message. User should be able to send instant message to any registered user from list	High	User
History	To make able to see past messages	High	User

2.4 Non-Functional requirements

Table 4: Non-Functional requirements

Name	Description	Priority	Actor
Privacy	Messages shared between users should be encrypted to maintain privacy	High	Admin
Robustness	To make able to deal with errors	High	Admin
Performance	Application performance must be better. Application must be lightweight and must send messages instantly	High	Admin
Usability	To make able to use even for newbie	High	Admin
Reliability	To be trustworthy to users	High	Admin
Support-ability	To be capable of being supportive	High	Admin
Portability	Application must be able to run in many different system	High	Admin

2.5 High level architecture

2.5.1 Data Flow Diagram

It is a notation designed to model information systems in terms of data storage, processing and transmission.

This diagram is made in the form of a peculiar tree. As you can see in the diagram, the entire scenario starts with the user. He is given two possibilities:

- "Login" - if the user already has an account.
- "Register" - if the user does not have an account.

Depending on the scenario selected, certain processes will be started and certain data will be requested from the server.

But regardless of the above scenarios, the user will be redirected to the chat screen, where he can start a dialog. From the current situation, the development goes to the method of deauthentication.

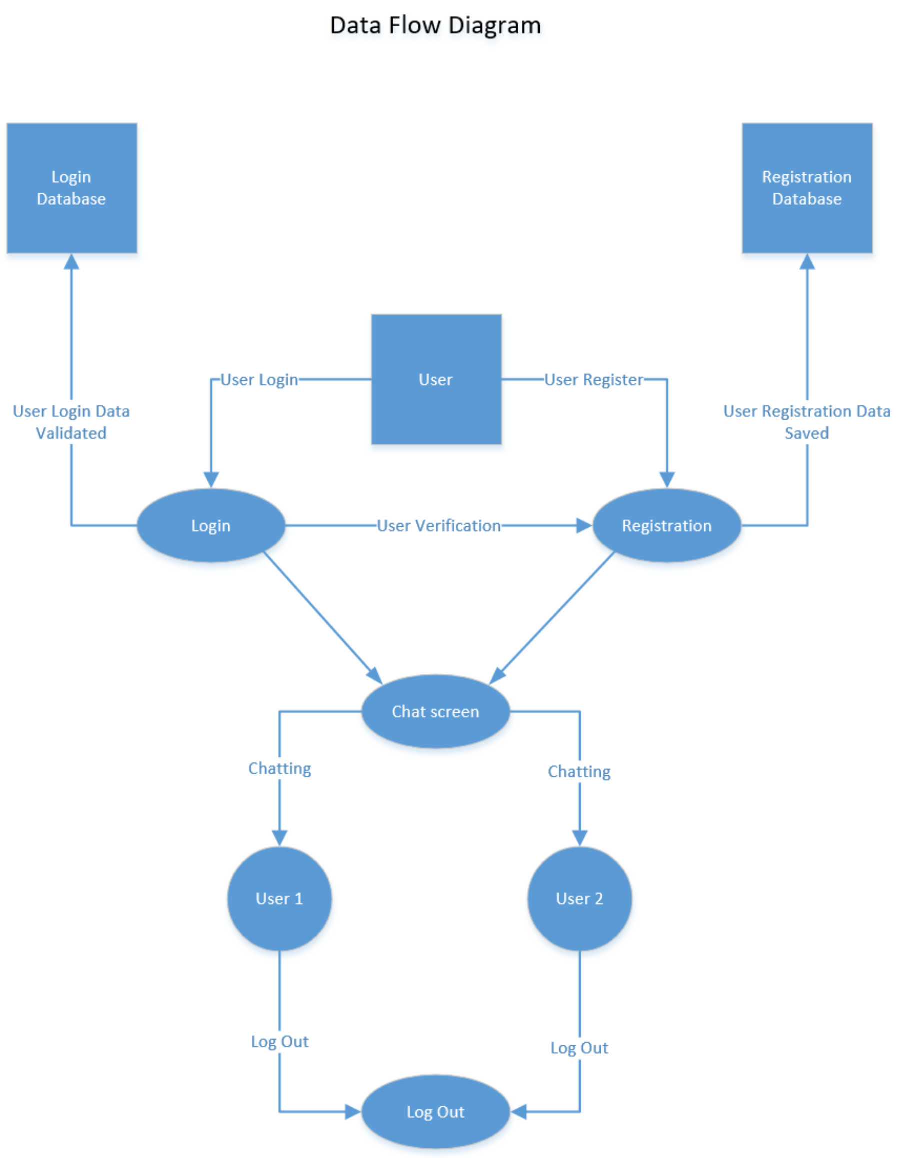


Figure 1: Data Flow Diagram[**Data_Flow**]

2.5.2 Deployment Diagram

The Deployment Diagram contains graphic representations of processors, devices, processes and the links between them.

The diagram depicts 3 operating systems on which the application can run. Within the operating system device, the application is described with the most important artifacts

The adjacent large block contains Google Server components, which include, Firebase Realtime database, Firebase console and Google Cloud Server.

This diagram shows the application elements involved when running on one or more devices.

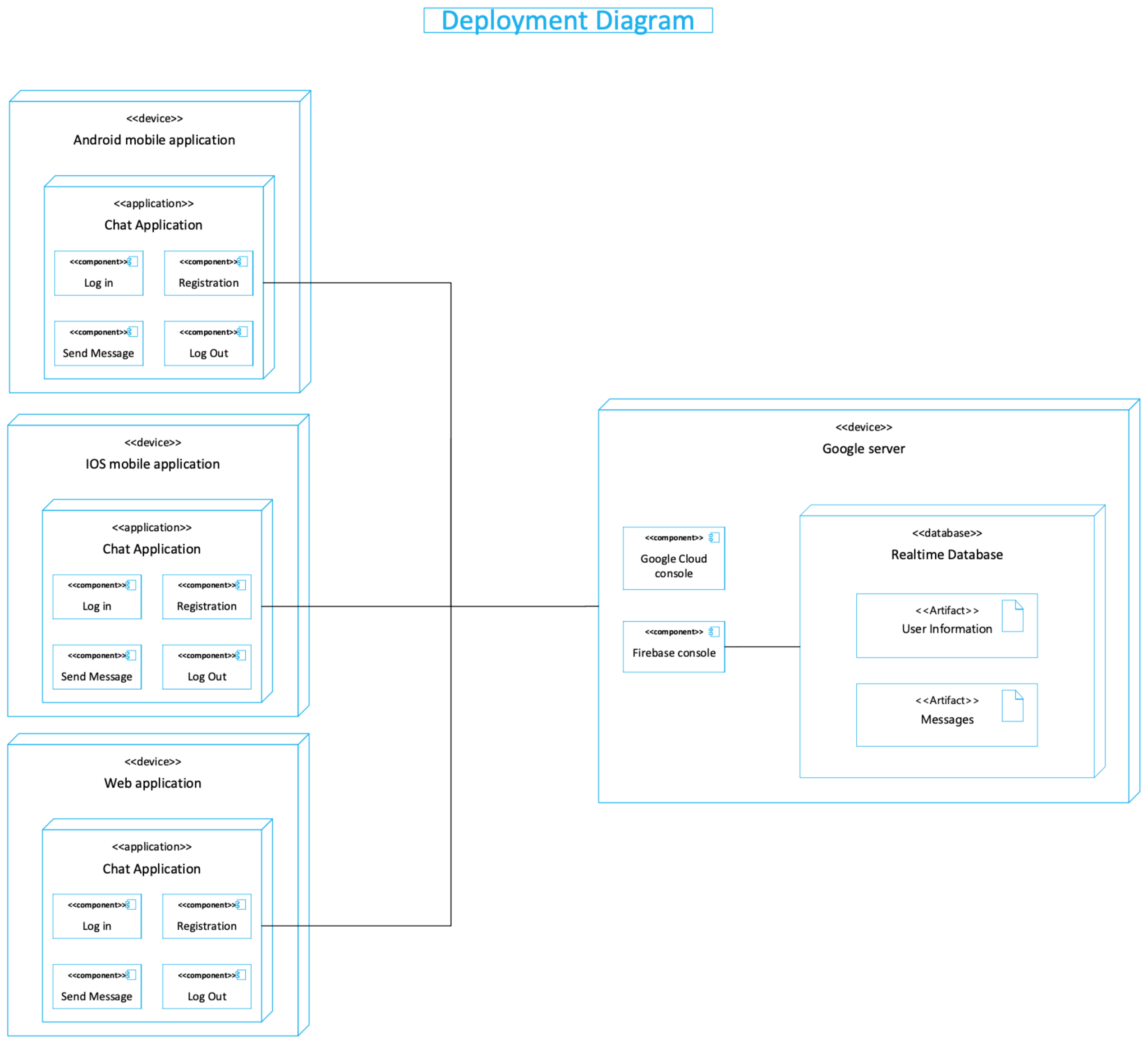


Figure 2: Data Flow Diagram[**Deployment_Diagram**]

2.6 Low level design

A sequence diagram is one of the varieties of interaction diagrams and is designed to simulate the interaction of System objects over time, as well as the exchange of messages between them.

In the diagram, you can see that the user opens the application first. After opening the application, he gets to a screen where he is asked to log in to a previously created account or to register. The diagram shows only the authorization scenario, as the communication between the systems will be the same. Once the authorization request has been made, the application sends a request to Firebase to allow this action. If the validation is successful, the server responds positively and sends the appropriate message to the device.

Next, the user can write a letter, by sending which will be generated the appropriate request and transmitted a certain set of data. The message is then written to the Realtime database with all the accompanying data that have been transmitted to the server during sending and returns a corresponding signal to display the message in the chat window. On the same principle, the user receives incoming messages.

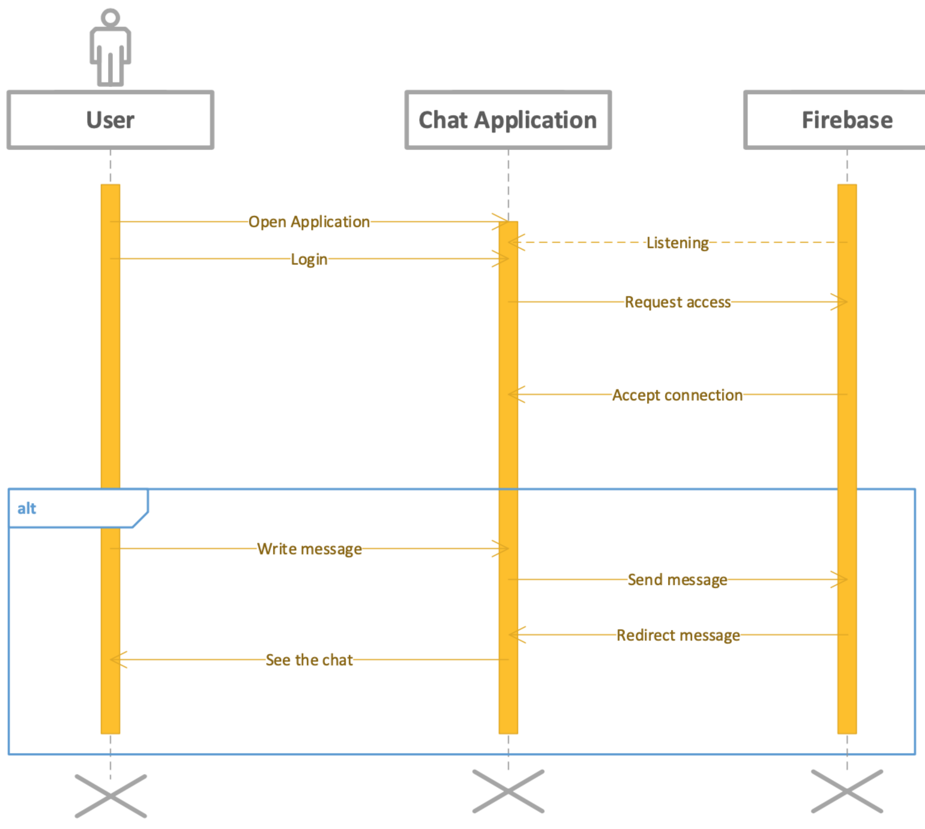


Figure 3: Data Flow Diagram[**Sequence_Diagram**]

2.7 Use cases

Use Case Diagram is one of the diagrams of UML (Unified Model Language) which demonstrates the interaction of the users (called Actors) with each other within the system. In other words, a use case diagram represents the activity of a system between actors, elements and their roles. The use case diagram consists of a system which is a whole scenario in which all events and flows of the different elements are interacted with each other. Actors can be users or internal elements in the system and the used cases that made connection with elements in the system that is always represented by an oval shape. [23]

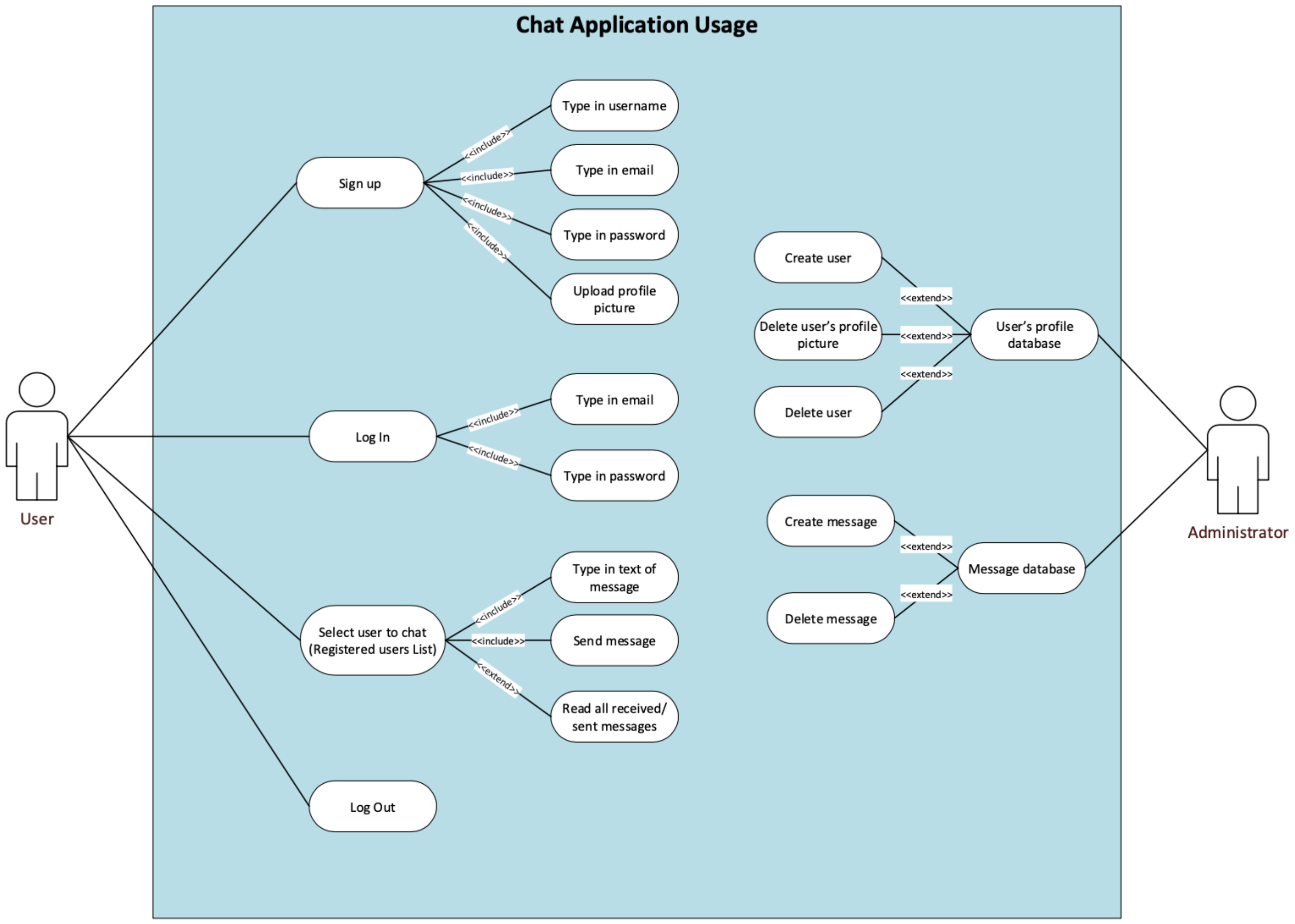


Figure 4: Use Case Diagram[**Use_Case_D**]

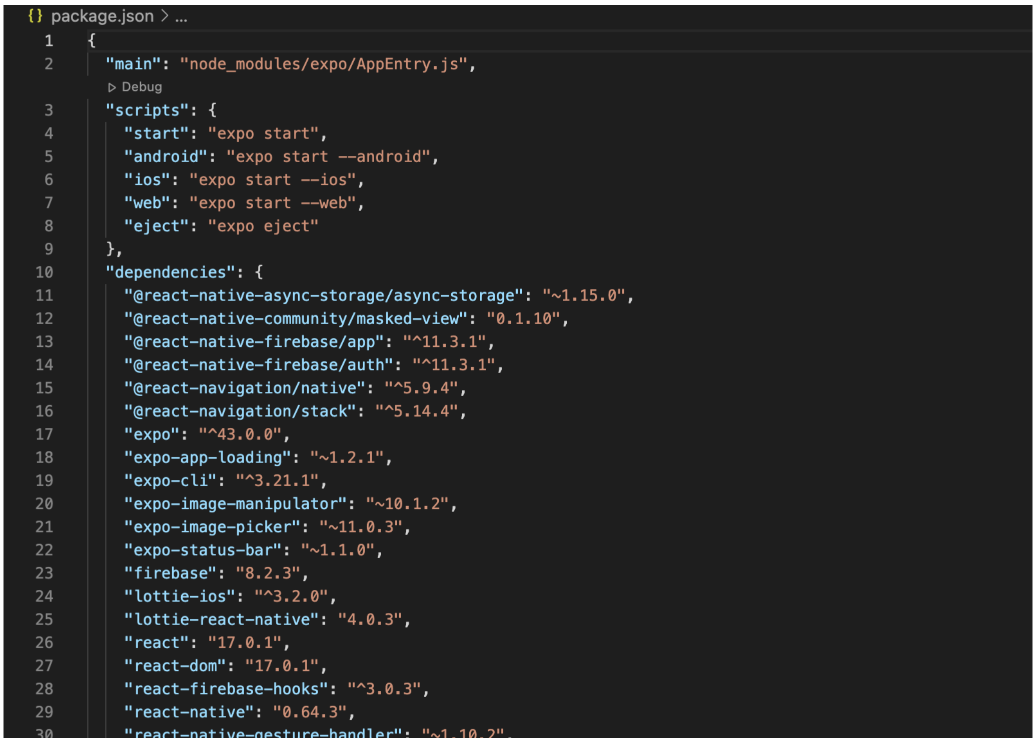
3 Creation and validation of Encrypted Chat Application

3.1 Chat application implementation

This section will describe both the basic steps for developing an application with code inserts for a deeper understanding. The main focus will be on the most important elements of the application, which constitute the very essence and idea of this project.

3.1.1 Environment Setup

The first step in creating a server is to install and run on a local host on a dedicated port. First you need to initialize "npm", through which the necessary packages are installed. During initialization a "package.json" (Figure 5) file is created, which describes the version, packages and other parameters.



```
{} package.json > ...
{
  "main": "node_modules/expo/AppEntry.js",
  ▷ Debug
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web",
    "eject": "expo eject"
  },
  "dependencies": {
    "@react-native-async-storage/async-storage": "~1.15.0",
    "@react-native-community/masked-view": "0.1.10",
    "@react-native-firebase/app": "^11.3.1",
    "@react-native-firebase/auth": "^11.3.1",
    "@react-navigation/native": "^5.9.4",
    "@react-navigation/stack": "^5.14.4",
    "expo": "^43.0.0",
    "expo-app-loading": "~1.2.1",
    "expo-cli": "^3.21.1",
    "expo-image-manipulator": "~10.1.2",
    "expo-image-picker": "~11.0.3",
    "expo-status-bar": "~1.1.0",
    "firebase": "8.2.3",
    "lottie-ios": "^3.2.0",
    "lottie-react-native": "4.0.3",
    "react": "17.0.1",
    "react-dom": "17.0.1",
    "react-firebase-hooks": "^3.0.3",
    "react-native": "0.64.3",
    "react-native-gesture-handler": "~1.10.2",
```

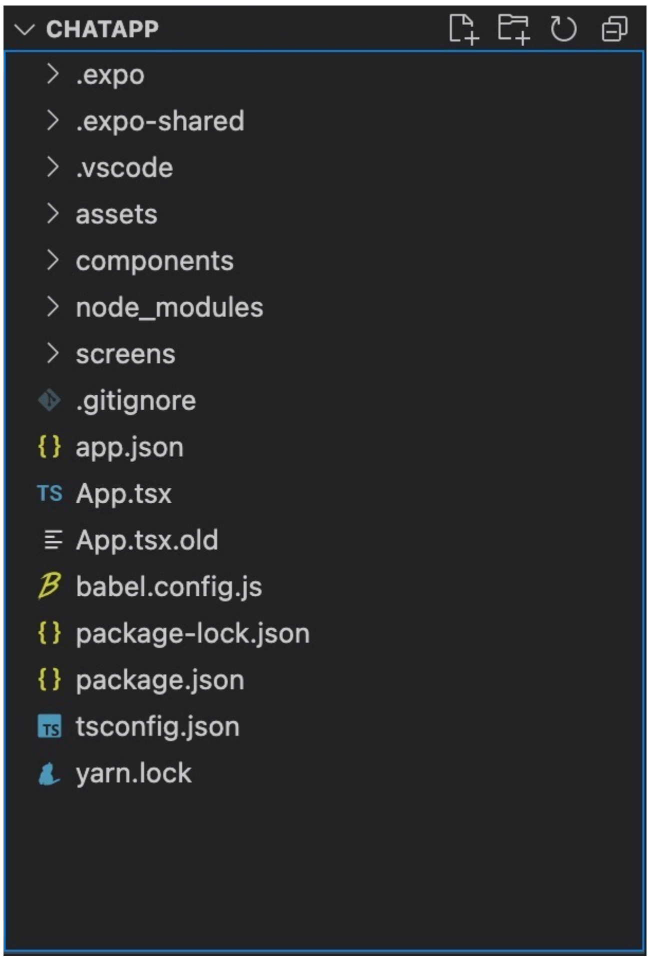
Figure 5: Package.json file[**Package.json file**]

After successfully installing the "NPM", you can proceed to install the "Expo SDK". To achieve a satisfactory result we also need to install "Node.JS". The installation of expo is done using the above initialized "NPM". After that it will be possible to create a project with configuration for TypeScript, because expo provides such a possibility. As a result, an empty project will be created with the existing file App.tsx and basic node-modules which are more than enough for the first test run of the application. The compilation is done again with expo. The commands used to install and run will be listed below.

```
sudo apt install nodejs
npm install -g expo-cli
expo init PROJECTNAME  #replace PROJECTNAME with your desired project name
expo start
```

3.1.2 Chat application structure

The basic structure of the application is as follows:



```
CHATAPP
  > .expo
  > .expo-shared
  > .vscode
  > assets
  > components
  > node_modules
  > screens
  .gitignore
  app.json
  App.tsx
  App.tsx.old
  babel.config.js
  package-lock.json
  package.json
  tsconfig.json
  yarn.lock
```

Figure 6: Chat Application Structure[**Chat Application Structure**]

The most prominent folders are:

- screens: This folder contains all the component screens that contain the state the user goes to.
- components: This folder contains all components that the user interacts with through screens.
- assets: Contains all static application assets: Favicon, images, etc.

3.1.3 Initializing Firebase in the application

After installing all the necessary components, we need to synchronize the application with Firebase. To do this there is a special package, which must be installed using "NMP". We will not delve into the installation, but stop already on the connection of the database in the application.

To do this, we need to create a new file in a separate folder or outside of it, it depends on the chosen development structure of the application. In my case, the file "firebase.tsx" is located in the folder path components/backend. This file must contain the following information:

– unique digits of ApplicationKey;

– authenticationDomain;

– databaseURL;

– projectID;

– storageBucket (data storage);

– messagingSenderId;

– ApplicationID;

– databaseURL;

Firebase itself is initialized with the firebase.initializeAp function, as implemented in my application.

The firebase.tsx file code of my application will be attached below:

```
import firebase from "firebase";

const firebaseConfig = {
  apiKey: "api key",
  authDomain: "chatappreactnative-dee56.firebaseapp.com",
  projectId: "chatappreactnative-dee56",
  storageBucket: "chatappreactnative-dee56.appspot.com",
  messagingSenderId: "360035375164",
  appId: "1:360035375164:web:7c04efbbcba275d1d64b39",
  databaseURL: "https://chatappreactnative-dee56-default-rtdb.firebaseio.com
   /"
}

export default firebase.initializeApp(firebaseConfig);
```

Listing 1: firebase.tsx

3.1.4 Registration

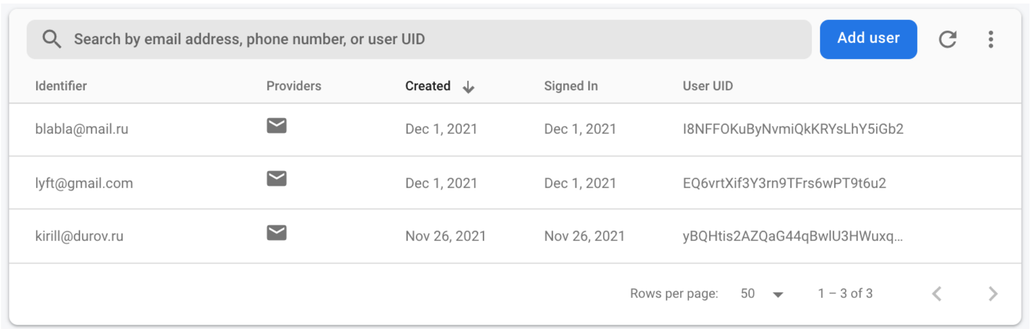
Registration is a basic component that any new user needs in order to first register and then log in to the Chat Application.

In the Chat App, the Register component requires you to enter your email, username, password, and upload a photo of the user. By default, an image will be uploaded to indicate there is no photo of the user.

The firebaseReg method uses the "Firebase API" to register the user. The "createUserWithEmailAndPassword" command gets two input parameters: email and password. These are written to a special "Authentication" section, which can be found in the Firebase Console.

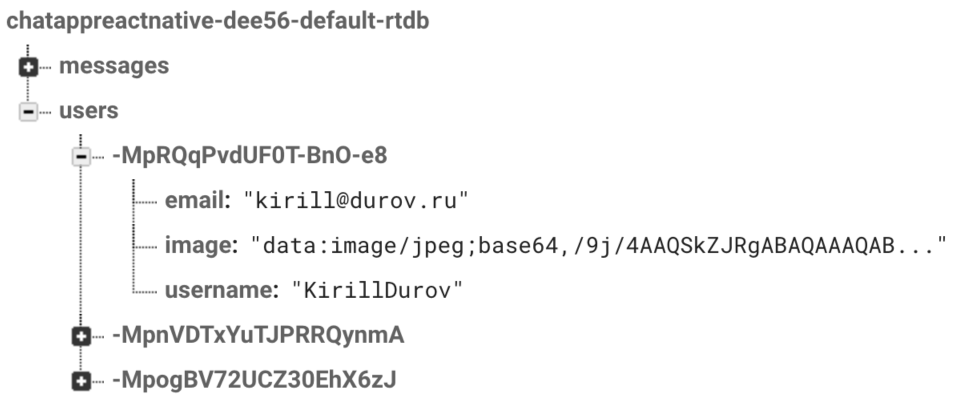
Also, during registration, the username, email and profile picture in base64 format are sent to the Realtime Database for further integration and loading into the chat window.

I would like to point out that Firebase Authentication uses an internally modified version of scrypt to hash passwords for accounts. Even if an account is loaded with a password that uses a different algorithm, Firebase Auth re-hashes the password the first time the account is successfully logged in. Accounts loaded from Firebase Authentication will only contain the password hash if it is available for that version of scrypt, otherwise the password hash will be empty.[24]



Identifier	Providers	Created	Signed In	User UID
blabla@mail.ru	✉	Dec 1, 2021	Dec 1, 2021	I8NFFOKuByNvmiQkKRYsLhY5iGb2
lyft@gmail.com	✉	Dec 1, 2021	Dec 1, 2021	EQ6vrtXif3Y3rn9TFrs6wPT9t6u2
kirill@durov.ru	✉	Nov 26, 2021	Nov 26, 2021	yBQHtis2AZQaG44qBwlU3HWuxq...

Figure 7: Firebase console Authentication tab[**Firebase console Auth**]



```
chatappreactnative-dee56-default-rtdb
  messages
  users
    -MpRQqPvdUF0T-BnO-e8
      email: "kirill@durov.ru"
      image: "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQAB..."
      username: "KirillDurov"
    -MpnVDTxYuTJPRRQynmA
    -MpogBV72UCZ30EhX6zJ
```

Figure 8: Realtime database user storage[**Realtime database user**]

The authorization method implemented and described above is presented in the code format below:

```
const firebaseReg = () => {
        firebase.auth()
            .createUserWithEmailAndPassword(email, password)
            .then((authUser) => {
                firebase.database().ref("users/").push({
                    email: email,
                    username: username,
                    image: image,
                });
            })
            .catch((error) => alert(error.message));
    };
```

Listing 2: registration.tsx

3.1.5 Login

The authentication method in the application looks almost the same as the user registration method. The "signInWithEmailAndPassword" method is called, which also receives 2 input parameters: email and password, and then the user is authenticated in the database. The code is presented below:

```
 const firebaseLogin = () => {
        firebase.auth()
        firebase.auth().signInWithEmailAndPassword(email, password)
            .then(() => {

            })
            .catch((error) => alert(error));
    }
```

Listing 3: login.tsx

In this section we would like to pay more attention to the implemented listener, whose task is to avoid re-authentication.

When a user has successfully logged in and closed the application without calling the "log out" method, no authentication will be required when the application is reopened, because the listener will understand that the user is still logged in and does not want to log out.

This method is implemented using the useEffect hook, as well as using the "Firebase API" metadata, namely firebase.auth().onAuthStateChanged.

The "navigation.replace" module is responsible for redirecting to the chat screen after the listener is triggered. This module is one of the main ones in my application because its task is to navigate between the screens. The screens, in turn, are described as a stack. An additional and rather convenient function of this module is the transfer of data between the screens. This allows you to access the database only once, making the work of the application a bit easier. For components such as profile picture and username, this is quite a handy solution, because these components are static from the moment of registration.

The implementation of this function in my application is shown below:

```
useEffect(() => {
       const unsubscribe = firebase.auth().onAuthStateChanged((authUser) =>
    {
           if (authUser) {
               var ref = firebase.database().ref("users/");
               ref.orderByChild('email').equalTo(authUser.email).on("value"
   , function (snapshot) {

                   var myobj = snapshot.val();
                   var keysArray = Object.keys(myobj);
                   var mydata = myobj[keysArray[0]];

                   navigation.replace("Chat", {
                       image: mydata.image,
                       username: mydata.username,
                   });

               });

           }
       });

       return unsubscribe;
   }, [])
```

Listing 4: login.tsx/listener

3.1.6 Send Message

In order to explain how sending a message works, I need to touch on another component, namely "ChatItem". This component represents the construction of the structure of the message, both outgoing and incoming. The location of the user name, picture, sending time, animation of sending and receiving messages, as well as the text position in the message - is the responsibility of the component "ChatItem". This component is implemented using interfaces: The first describes:

– id: a unique value of the message generated randomly;

– text: the content of the message;

– image: photo of the user;

– timeStamp: time of sending;

– by: by whom the message was sent (username);

The second interface takes the values of the first interface and supplements them with user data from the database.

For a clearer understanding of the above explanation, here is the code:

```
interface ChatItem {
  id: string;
  text: string;
  image: string;
  timeStamp: number;
  by: string;
}

interface Props {
  chatItem: ChatItem;
  username: string;
}


const RenderChatItem = ({ chatItem, username }: Props) => {
  let unknownAvatarImage = "base64 coding for default image";
  let avatarImage = chatItem.image ?? unknownAvatarImage;


  // Adds animation to the new Chat Item
  let [animatedValue] = useState(new Animated.Value(0));
  useEffect(() => {
    Animated.timing(animatedValue, {
      toValue: 1,
      duration: 400,
      easing: (number) => Easing.ease(number),
      useNativeDriver: true,
    }).start();
  });
  return (
    // Styling for incoming and outgoing messages. Currently not in use, it
   does only animation for messages
    <Animated.View
      style={[
        Styles.flatListItem,
        { backgroundColor: username == chatItem.by ? "#ECECEC" : "#6db7f7"
   },
        { alignSelf: username == chatItem.by ? "flex-end" : "flex-start" },
        { opacity: animatedValue },
        { transform: [{ scale: animatedValue }] },
      ]}
    >
       {/* Adding profile image to the message*/}
      <View style={Styles.chatItemHeader}>
        <Image
          source={{
            uri: "data:image/jpeg;base64," + avatarImage,
            // uri: avatarImage,
          }}
          style={Styles.avatarSmall}
        />
        <Text style={Styles.smallItalicText}>
          {chatItem.by} at {new Date(chatItem.timeStamp).toLocaleTimeString
   ()}
        </Text>
      </View>
      <Text style={Styles.chatText}>{chatItem.text}</Text>
    </Animated.View>

  );
};

```

```
export { ChatItem, RenderChatItem };
```

Listing 5: chatitem.tsx

At this point, you can begin to parse the method of sending the message. The method of sending the message includes the generation of a random message identifier, as well as the generation of the sending time.

When the user wrote the message, the "ChatItem" method is called, which receives all the necessary parameters listed in the previous section for further processing. After receiving all the necessary data, the "firebase.database().ref("messages/" method from "Firebase API" is called, in order to transfer all the information to the database. As a parameter for the ".ref" method, the directory in which the files are to be stored is specified. If the directory passed as parameter is not found, it will be created automatically.

After all the data has been successfully sent, the input field of the message becomes empty.

```
onPress={() => {

            var id = Math.random().toString(36).substring(7);
            var dateTime = Date.now();

            setChatItemList([
              ...chatItemList,
              {
                id: id,
                text: chatInput,
                image: aImage,
                timeStamp: dateTime,
                by: username,
              },
            ]);
            firebase.database().ref("messages/").push({
              content: chatInput,
              dateTime: dateTime,
              image: aImage,
              username: username,
              id: id,
            });

            setChatInput("");
          }
          }
```

Listing 6: chat.tsx/sendMessage

3.1.7 Receiving messages

To receive messages and display them on the screen, we need to access the database, as well as call the ChatItem method, which will display messages in the correct form.

To solve this problem, we created a separate method which, via API, accesses the directory in the Realtime database, specifically the "messages" section. Then the id of the messages is compared for the correct display of incoming and outgoing messages. At the very end of this process, there is a call to "ChatItem" and the data obtained from the directory Realtime database is passed as parameters to the ChatItem component.

UseEffect hook was used to call this method.

```
const getData = () => {
  var ref = firebase.database().ref("messages/");
  ref.orderByKey().limitToLast(10).on("value", function (chatItem2) {
   chatItem2.forEach((item) => {
    setChatItemList((chatItemList) => {
     if (chatItemList.find((i) => i.id == item.child("id").val())) return chatItemList;
     return [
      ...chatItemList,
      {
       id: item.child("id").val(),
       text: item.child("content").val(),
       image: item.child("image").val(),
       timeStamp: item.child("dateTime").val(),
       by: item.child("username").val(),

      },
     ]
    });

   })
   return
  });
 }

 useEffect(() => {
  getData();
 }, [])
```

Listing 7: chat.tsx/recieveMessage

For the subsequent display of messages on the screen we have written a method "renderItem", which transforms the received data from the above described method "getData" into a readable and user-friendly form.

The "renderItem" method is then called in the <FlatList> component, which displays the data converted by the "renderItem" method in a list and arranges it by date and time of sending/receiving.

The code implementation of the two mentioned methods is shown below:

```
const renderItem: ListRenderItem<ChatItem> = ({ item }) => (
    <RenderChatItem chatItem={item} username={username}></RenderChatItem>
  );

---------------------------------------------------------------
  <FlatList
        inverted
        data={chatItemList.sort((a, b) => b.timeStamp - a.timeStamp)}
        keyExtractor={(item) => item.id}
        renderItem={renderItem}
      ></FlatList>
```

Listing 8: chat.tsx/renderItem

3.2 Testing

In this chapter, test cases of the application are examined.

3.3 System Testing Cases

3.3.1 Sign Up new User

- **Test Step**: Go to the Chat App page and Sign Up with a valid email and a six-character password then click "Sign Up".
- **Expected Result**: It should allow the user and enter the chat panel.
- **Actual Outcome**: A new user is entered in the chat panel.

3.3.2 Check New Created User in Firebase database

- **Test Step**: Go to the Firebase console project with Gmail and password. Check the Firebase´s Authentication users.
- **Expected Result**: Newly Created User email, created date and unique uid in Firebase.
- **Actual Outcome**: A new Created User is available in Firebase Authentication user list.

3.3.3 Login new User without Sign Up

- **Test Step**: Go to the Chat App page and Login with a valid email and a six-character password and click "Sign Up".
- **Expected Result**: It should not allow the user to enter the chat panel.
- **Actual Outcome**: User is not allowed to enter chat panel.

3.3.4 User messages with username and time

- **Test Step**: Logged in user writes the message in Chat panel.
- **Expected Result**: Users writes any message and receiver views the message with sender´s username and time.
- **Actual Outcome**: Sender´s username with time is viewed by receiver.

3.3.5 Logged in User chat data in Firebase

- **Test Step**: Go to the Firebase console project with Gmail and password. Check the Firebase´s real-time database data.
- **Expected Result**:User´s chat messages data available in the real-time database.
- **Actual Outcome**: Chat messages data of user are available in Firebase database.

3.3.6 Sign Up new User with less than six-character password

- **Test Step**: Go to the Chat App page and Sign Up with a valid email and less than six-character password and click "Sign Up".
- **Expected Result**: Chat App does not allow user to sign up.
- **Actual Outcome**: Sign up is not successful.

3.3.7 Sign Up new User with mobile number

- **Test Step**: Go to the Chat App page and Login with mobile number and six-character password and click "Sign Up".
- **Expected Result**: User should not able to sign up because it is defined in Firebase Authentication user can only sign up with a valid email and password.
- **Actual Outcome**: Sign up is not successful.

3.3.8 Sign Up new User with existing email in database

- **Test Step**: Go to the Chat App page and SignUp with already registered email and click "Sign Up".
- **Expected Result**: User should not able to sign up because such email already exist in Firebase storage.
- **Actual Outcome**: Sign up is not successful.

3.3.9 Delete the user chat data in Firebase database

- **Test Step**: Go to the Firebase console project with Gmail and password. Check the Firebase´s real-time database data. Delete selected message.
- **Expected Result**: Administrator allows to delete user chat messages or data that will be also deleted of the selected message of the chat panel.
- **Actual Outcome**: The message(s) or data is lost by deleting in the Firebase real-time database.

3.3.10 Logged in user Refreshes chat automatically

- **Test Step**: Logged in user writes the message in Chat panel.
- **Expected Result**: Chat panel windows Refreshes automatically and is also observed by user.
- **Actual Outcome**: Window is Refreshed in the chat panel automatically.

3.4 Conclusions of section

TypeScript has greatly simplified web application development for everyone, while maintaining a leadership position in the face of a growing number of technologies and programming languages. The underlying JavaScript simplified development and time to learn, since JavaScript was learned before at university. With its frameworks running on TypeScript, developers can easily create hybrid applications for various platforms, including iOS, Android and Web versions that can be used on personal computers as well as on various tablets or smartphones with the aforementioned operating systems. One of the main reasons to use TypeScript for app development is that it allows pages to be very attractive, present spectacular graphics, validate data, create cookies, respond to events, and much more. These advantages clearly define its strength in the development community and how it simplifies the web application development process.

React Native had both a hardware advantage over the other frameworks mentioned, and a personal one, as this framework was used to write a term paper and basic knowledge was already present. Manipulation of the browser DOM is not inherently slow, it is their drawing the user interface, which is costly. The virtual DOM in React Native helps minimize these paint events by ensuring that only the elements that need to be changed in the DOM are manipulated, and that these updates are sent in batches. These update packages prevent unnecessary "slow" paint events and make applications developed with React Native more efficient.

The Expo SDK has proven itself as a toolkit that provides access to system functions without any problems. I would like to return to the idea that factors such as:

- No need to install Xcode and Android Studio, which speeds up the development process;
- Portability, which is achieved by running the code in any native environment containing the Expo SDK;
- Providing UI components to handle a variety of use cases that will cover almost all applications, but will not be built into the React Native core;
- Debugging during development;
- Hot Reloading;

The above criteria proved to me practically that I was not wrong in choosing the SDK to implement my project.

Firebase has revolutionized the way developers test and use hosting as a service compared to its competitors. In my case, it's a much simpler and easier way to effectively host applications without going into the details and intricacies of hosting setup. Simple both in installation and in use API proved to be amazing in the long run, as there were no difficulties in using it, which is not to say about the construction of the logic of the code because of which API can work incorrectly. Because of its popularity and active development, the documentation for Firebase is more than detailed and comprehensive. All acquaintance with this technology was held with the direct use of official documentation, which for the implementation of my project was beneficial

General Conclusions

In the course of the thesis, existing analogues were analyzed and toolkits were selected. The comparison was made according to the selected criteria, which, when examined in detail, allowed us to choose the most appropriate tools for development. After choosing necessary tools for implementation, a cross-platform messenger for popular operating systems and web versions was developed.

React Native, Firebase and Expo were chosen as the main tools because they are promising, convenient and widely used technologies. Expo - rather simple and easy to understand and well-established in the market of mobile and web-applications development. Thanks to a large range of functionality for development on React Native, further acquaintance with the framework was flawless. React Native is a young and rapidly evolving framework for cross-platform application development, which throughout the project development process has proven itself as a potential and comfortable framework for writing cross-platform applications.

Firebase, on the other hand, is certainly terrific in execution, implementation, and operation. The ease of integration, the extensive functionality within my project and the inexhaustible amount of documentation created the most pleasant atmosphere to work with this technology. Experience with Firebase throughout the project proved the accuracy of choice when comparing this product with the aforementioned counterparts.

The implemented software product supports modern versions of Android 12 and iOS 15 and below. As a result, we have a chat room allowing users to exchange messages in real time without any problems. At the moment the application is raw and needs further modification. Many features conceived and not covered in this work still need development, such as: sending media files, sending voice messages.

References

[1] Sovetov B.Y., Cekhanovskiĭ V.V. , Chertovskiĭ V.D. “Intellektualnye sistemy i tekhnologii”. In: (2013).

[2] Sovetov B.Y., Vodyaho A.I., Dubeneckij V.A., Cekhanovskij V.V. “Arhitektura informacionnyh sistem”. In: (2012).

[3] Stephen Martin. “Messengers: Who We Listen To, Who We Don’t, and Why”. In: (Oct 15, 2019).

[4] Sharonin P.N., Kozlova E.V. “Rol’ messendzherov v sovremennom mediaprostranstve”. In: (2017).

[5] Sovetov B.Y., Yakovlev A.Y. “Modelirovanie sistem”. In: (2013).

[6] “Benefits of Cross-Platform Development”. In: (June 8, 2018). URL: `https://%20itexico.com/blog/benefits-of-cross-platform-development`.

[7] “Xamarin vs Ionic vs React Native: differences under the hood”. In: (May 9, 2018). URL: `https://cruxlab.com/blog/reactnative-vs-xamarin/`.

[8] “Pricing – Amazon Web Services”. In: (May 14, 2018). URL: `https://www.altexsoft.com/blog/engineering/xamarin-vs-react-%20native-vs-ionic-cross-platform-mobile-frameworks-comparison/`.

[9] “Part 2 - Architecture - Xamarin”. In: (June 6, 2018). URL: `https://%20docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-%20cross-platform-applications/architecture`

[10] “Sharing Code Overview - Xamarin”. In: (June 6, 2018). URL: `https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/code-sharing`.

[11] “Ionic Documentation”. In: (June 6, 2018). URL: `https://ionicframework.com/docs/`.

[12] Robin Wieruch. “React Native Explained A complete guide to modern web and mobile development with React.js: Step-by-Step Guide to React”. In: (Jun 28, 2021).

[13] Petru Alin Gheorghe. “Lightning Fast Mobile App Development with Galio Build stylish cross-platform mobile apps with Galio and React Native”. In: (Sep 9, 2021).

[14] Aaron Ploetz, Devram Kandhare, Sudarshan Kadambi, Xun (Brian) Wu. “Seven NoSQL Databases in a Week Get up and running with the fundamentals and functionalities of seven of the most popular NoSQL databases”. In: (2018).

[15] Gerardus Blokdyk. “Amazon DynamoDB Details Standard Requirements”. In: (2018).

[16] Robin Wieruch. “The Road to Firebase Your journey to master Firebase in JavaScript”. In: (2021).

[17] Eric Bush. “Node.js, MongoDB, React, React Native Full-Stack Fundamentals and Beyond”. In: (Oct 5, 2018).

[18] Eric Bush. “NoSQL with MongoDB in 24 Hours, Sams Teach Yourself”. In: (Aug 21, 2014).

[19] Robin Wieruch. “The Road to React with Firebase: Your journey to master advanced React for business web applications”. In: (2021).

[20] “Expo Documentation”. In: (2019). URL: https://docs.expo.dev/.

[21] Ryabko B. Ya., Fionov A. N. “Kriptograficheskie metody zashchity informacii: Uchebnoe posobie dlya vuzov”. In: (2015).

[22] Song Yan. “Kriptoanaliz RSA”. In: (2011).

[23] Fauler Martin. “UML. Osnovy. Kratkoe rukovodstvo po standartnomu yazyku ob"ektnogo modelirovaniya”. In: (2018).

[24] “Firebase Authentication Password Hashing”. In: (2019). URL: https://firebaseopensource.com/projects/firebase/scrypt/.