



**YAŞAR UNIVERSITY  
FACULTY OF ENGINEERING  
DEPARTMENT OF COMPUTER ENGINEERING**

**COMP4910 Senior Design Project 1, Fall 2023  
Advisors: Prof. Dr. Ahmet Hasan Koltuksuz  
Assoc.Prof.Dr.Barış Yıldız**

# **GEMBOS: Global Encrypted Mobile-Based Obscured SMS Application**

## **Final Report**

**19.01.2025**

**By:  
Berker Vergi, 21070001202  
Giray Aksakal, 21070001030  
Mert Kahraman, 20070006020  
Emir Morah, 21070006048**

## PLAGIARISM STATEMENT

This report was written by the group members and in our own words, except for quotations from published and unpublished sources which are clearly indicated and acknowledged as such. We are conscious that the incorporation of material from other works or a paraphrase of such material without acknowledgement will be treated as plagiarism according to the University Regulations. The source of any picture, graph, map or other illustration is also indicated, as is the source, published or unpublished, of any material not resulting from our own experimentation, observation or specimen collecting.

### Project Group Members:

Name, Lastname	Student Number	Signature	Date
Berker Vergi	21070001202		20.01.2025
Giray Aksakal	21070001030		20.01.2025
Emir Moralı	21070006048		20.01.2025
Mert Kahraman	20070006020		20.01.2025

### Project Advisors:

Name, Lastname	Department	Signature	Date
Ahmet Hasan Koltuksuz	Computer Engineering		
Barış Yıldız	Software Engineering		

## ACKNOWLEDGEMENTS

We would like to extend our heartfelt gratitude to everyone who supported and contributed to the successful completion of the GEMBOS project.

Firstly, we sincerely thank our project advisors, **Prof. Dr. Ahmet Hasan Koltuksuz** and **Barış Yıldız**, for their invaluable guidance, expertise, and constructive feedback throughout the development process. Their unwavering support and encouragement were instrumental in overcoming challenges and achieving the project goals.

We would also like to recognize the dedication and collaborative efforts of our team members, **Berker Vergi, Giray Aksakal, Emir Moral, and Mert Kahraman**, whose hard work, innovative ideas, and commitment to excellence were pivotal in the realization of this project. Each member's unique contributions were essential in delivering a robust and secure communication platform.

Finally, we acknowledge the faculty and staff at Yaşar University for providing the resources and environment necessary to carry out this project successfully.

## **KEYWORDS**

Diffie-Hellman Key Exchange  
Elliptic Curve Cryptography  
Cryptography  
Encryption  
Decryption  
API  
Application  
iOS  
Android  
Database  
SMS  
Message over Internet

## ABSTRACT

The GEMBOS project, developed as part of the Senior Design Project 1 (COMP 4910) at Yaşar University, aims to revolutionize secure communication by introducing a robust, distributed messaging platform. GEMBOS provides a seamless solution for users to send encrypted messages both online and offline, addressing critical issues of privacy, data security, and connectivity challenges in modern communication systems. By integrating Elliptic Curve Cryptography (ECC) and advanced synchronization protocols, the project ensures secure, reliable, and user-centric messaging.

During the Fall 2023 semester, the project team focused on foundational planning, research, and design. This included the creation of detailed Requirements Specification and Design Specification Documents, development of UML diagrams, and construction of scalable database structures. Functional prototypes were also developed, emphasizing secure key exchange mechanisms, message encryption workflows, and seamless synchronization of offline messages with a centralized system.

GEMBOS introduces a novel approach to messaging by combining end-to-end encryption, offline storage, and real-time synchronization in a distributed architecture. The platform dynamically adapts to user connectivity states, ensuring uninterrupted communication while safeguarding sensitive data. By providing transparent encryption feedback and user-friendly interfaces, GEMBOS empowers individuals to communicate confidently, even in challenging network environments.

This project demonstrates the potential of secure communication technologies to address modern challenges in data privacy and accessibility, offering an innovative solution for individuals and industries requiring secure and flexible communication tools. GEMBOS serves as a prime example of how distributed systems and advanced encryption can be leveraged to enhance user experiences in the digital age.

## ÖZET

**GEMBOS** projesi, Yaşar Üniversitesi'nde yürütölen Senior Design Project 1 (COMP 4910) kapsamında geliştirilmiş olup, güvenli iletişimi yeniden tanımlamayı hedefleyen güçlü ve dağıtık bir mesajlaşma platformudur. GEMBOS, kullanıcıların hem çevrimiçi hem de çevrimdışı olarak şifreli mesajlar göndermelerine olanak tanıyarak, modern iletişim sistemlerindeki gizlilik, veri güvenliği ve bağlantı sorunlarına yenilikçi bir çözüm sunar. Eliptik Eğri Kriptografisi (ECC) ve gelişmiş senkronizasyon protokollerini entegre eden proje, güvenli, güvenilir ve kullanıcı odaklı bir mesajlaşma deneyimi sağlamaktadır.

2023 Güz dönemi boyunca, proje ekibi temel planlama, araştırma ve tasarım aşamalarına odaklanmıştır. Bu kapsamda, detaylı Gereksinim Belirleme (RSD) ve Tasarım Belirleme (DSD) dokümanlarının oluşturulması, UML diyagramlarının geliştirilmesi ve ölçeklenebilir bir veritabanı yapısının tasarlanması gerçekleştirilmiştir. Ayrıca, güvenli anahtar değişim mekanizmaları, mesaj şifreleme süreçleri ve çevrimdışı mesajların merkezi bir sistemle senkronizasyonunu vurgulayan işlevsel prototipler geliştirilmiştir.

GEMBOS, uçtan uca şifreleme, çevrimdışı depolama ve gerçek zamanlı senkronizasyonu bir araya getirerek mesajlaşmaya yenilikçi bir yaklaşım sunmaktadır. Platform, kullanıcıların bağlantı durumlarına dinamik olarak uyum sağlayarak kesintisiz iletişim sağlarken hassas verileri korur. Şeffaf şifreleme geri bildirimi ve kullanıcı dostu arayüzleri ile GEMBOS, bireylerin zorlu ağ ortamlarında bile güvenli iletişim kurmalarını sağlamaktadır.

Bu proje, veri gizliliği ve erişilebilirlik konularındaki modern zorlukları ele alan güvenli iletişim teknolojilerinin potansiyelini ortaya koymaktadır. GEMBOS, güvenli ve esnek iletişim araçlarına ihtiyaç duyan bireyler ve sektörler için yenilikçi bir çözüm sunarak, dağıtık sistemler ve gelişmiş şifreleme tekniklerinin dijital çağdaki kullanıcı deneyimlerini nasıl geliştirebileceğini göstermektedir.

## TABLE OF CONTENTS

PLAGIARISM STATEMENT.....	2
ACKNOWLEDGEMENTS.....	3
KEYWORDS.....	4
ABSTRACT.....	5
ÖZET.....	6
TABLE OF CONTENTS.....	7
LIST OF FIGURES.....	8
LIST OF ACRONYMS/ABBREVIATIONS.....	10
1. INTRODUCTION.....	1
1.3. Project Outputs/Deliverables.....	3
2. SURVEY OF RELATED WORK.....	4
3. REQUIREMENTS.....	5
4. DESIGN.....	4
4.1. High Level Design.....	4
4.2. Detailed Design.....	4
4.3. Realistic Restrictions and Conditions in the Design.....	4
5. IMPLEMENTATION AND TESTS.....	5
5.1. Implementation of the Product.....	5
5.2. Tests and Results of Tests.....	5
6. PROJECT MANAGEMENT.....	7
6.1. Project Plan.....	7
6.2. Project Effort/Manpower.....	8
6.3. Project Cost Analysis.....	8
7. CONCLUSIONS.....	9
7.1. Summary.....	9
7.2. Benefits of the Project.....	9
7.3. Future Work.....	10
REFERENCES.....	11
APPENDICES.....	12
APPENDIX A: REQUIREMENTS SPECIFICATION DOCUMENT.....	13
APPENDIX B: DESIGN SPECIFICATIONS DOCUMENT.....	57
APPENDIX C: PROJECT MANAGEMENT DOCUMENTS.....	71

**LIST OF TABLES**

Table 1: Project Plan .....	89
Table 2: Project Effort Log.....	98
Table 3: Project Effort Log Giray Aksakal.....	100
Table 4: Project Effort Log Berker Vergi.....	102
Table 5: Project Effort Log Mert Kahraman.....	104
Table 6: Project Effort Log Emir Moralı.....	106



## **LIST OF ACRONYMS/ABBREVIATIONS**

UML: Unified Modeling Language

ML: Machine Learning

API: Application Programming Interface

REST: Representational State Transfer

SQL: Structured Query Language

## 1. INTRODUCTION

### 1.1. Description of the Problem

In today's fast-paced digital communication landscape, ensuring security and user accessibility is paramount. Many messaging platforms fail to strike a balance between ease of use and robust encryption, leaving users either overwhelmed by complexity or exposed to privacy risks. This challenge is particularly significant for users who require secure, reliable, and straightforward communication tools that work seamlessly both online and offline.

One of the key issues lies in ensuring secure messaging, especially when internet connectivity is unavailable. Existing messaging platforms often rely solely on internet-based communication, leaving users unable to send or receive messages in remote areas or during network outages. Additionally, offline storage and synchronization of sensitive messages introduce significant privacy and security concerns.

Another prevalent challenge is the lack of comprehensive encryption protocols in many platforms. While some systems offer encryption, they often fall short of providing end-to-end security or lack transparency about how data is protected. Users are left vulnerable to data breaches, unauthorized access, and lack of control over their private communications.

Furthermore, modern messaging systems often lack seamless integration between online and offline functionality. Messages sent offline may not synchronize properly when connectivity is restored, leading to loss of data and disrupted communication. For professionals and users in sensitive industries, such inefficiencies can have serious consequences.

**GEMBOS (General Encrypted Mobile-Based Obscured SMS)** is designed to tackle these challenges head-on. By combining modern encryption protocols, dynamic offline/online synchronization, and a user-centric interface, GEMBOS offers a secure, efficient, and versatile communication platform. It utilizes state-of-the-art elliptic curve encryption (ECC) to ensure end-to-end security and integrates offline SMS functionality, allowing users to send and receive messages even without internet connectivity.

GEMBOS also prioritizes user transparency and accessibility. The platform offers clear feedback on message statuses, encrypted storage for sensitive messages, and a dynamic user experience that adapts to the user's connectivity status. Whether online or offline, users can communicate confidently, knowing their data is secure and their messages will synchronize seamlessly.

For further insights into the issues addressed by GEMBOS and the detailed solutions it offers, refer to the GEMBOS Requirements Specification Document (RSD), which provides a comprehensive analysis of the problem space and design rationale.

### 1.2. Project Goal(s)

The primary objective of GEMBOS is to provide a secure, user-friendly communication platform that seamlessly bridges the gap between online and offline messaging while adhering to modern security standards. The goals of GEMBOS are as follows:

#### 1. Secure Messaging:

- Implement end-to-end encryption using elliptic curve cryptography (ECC) to ensure secure data transmission and storage.
- Enable users to send encrypted messages both online and offline.

- Ensure secure key management and dynamic synchronization for all messages.

## **2. Offline and Online Synchronization:**

- Develop a dual-mode messaging system that supports SMS-based offline communication and automatic synchronization when connectivity is restored.
- Implement robust offline message storage to prevent data loss during network outages.

## **3. Dynamic User Interface:**

- Provide an intuitive and responsive user interface that simplifies navigation and ensures accessibility for all users.
- Incorporate features like visual indicators for message statuses (sent, delivered, read) and encryption notifications.

## **4. Comprehensive Security Measures:**

- Ensure compliance with security standards, such as ISO/IEC 27001, to enhance data privacy and user trust.
- Provide transparent feedback to users regarding encryption protocols and data handling.

## **5. Scalability and Reliability:**

- Design a scalable architecture capable of handling increasing user demands and message loads.
- Ensure reliability through robust error handling and system monitoring.

## **6. Cross-Platform Functionality:**

- Ensure consistent user experience across mobile devices and operating systems.
- Support both Android and iOS platforms using native development frameworks.

By achieving these objectives, GEMBOS aims to redefine secure communication, offering users a reliable, transparent, and user-friendly platform for both personal and professional use.

### **1.3. Project Outputs/Deliverables**

During the initial phase of the GEMBOS project (COMP4910), the primary objective was to establish a solid foundation for the system through the development of key documentation and preliminary design elements. These deliverables included the following:

#### **Project Assignment Form (PAF):**

Formalized the project's scope, objectives, and deliverables, ensuring alignment with academic and project requirements.

#### **Requirements Specification Document (RSD):**

Detailed the functional and non-functional requirements of the GEMBOS system, providing a comprehensive understanding of the project's goals. The document included UML diagrams to visually represent system requirements and underwent iterative updates based on stakeholder feedback and project progress.

### **Design Specification Document (DSD):**

Outlined the high-level architecture of GEMBOS, using UML diagrams to illustrate system components and their interactions, such as encryption, messaging, and synchronization modules.

### **Initial Project Poster and Presentation:**

Highlighted the core features, objectives, and intended outcomes of the GEMBOS system. These served as introductory materials for academic and professional audiences, providing a concise overview of the project.

For the final phase of the project (COMP4920), the focus will shift to refining, finalizing, and delivering a fully implemented version of GEMBOS. The expected deliverables include:

### **Updated RSD and DSD Documents:**

Finalized versions of the Requirements Specification Document and Design Specification Document, incorporating all updates, refinements, and comprehensive system design details.

### **Final Report:**

A complete report summarizing the project's journey, from inception to completion. This document will detail the project's goals, methodologies, findings, and outcomes, including appendices with the finalized RSD and DSD documents.

### **Refined Project Poster and Presentation:**

Polished versions of the project poster and presentation, showcasing GEMBOS's functionality, technical details, and potential applications to academic and professional audiences.

### **Project Website:**

A user-friendly and accessible platform designed to demonstrate GEMBOS. The website will include UI screenshots, system architecture diagrams, and interactive walkthroughs of the platform's features.

### **User and Developer Manuals:**

Comprehensive guides aimed at facilitating effective use and maintenance of GEMBOS.

**User Manual:** Instructions for end-users on how to utilize the platform's features, such as messaging, encryption, and offline synchronization.

**Developer Manual:** Technical documentation for developers, detailing the system architecture, coding standards, and integration points.

### **Project Source Code:**

The fully implemented GEMBOS system, including encryption, messaging, and synchronization modules, accompanied by detailed developer documentation.

## 2. SURVEY OF RELATED WORK

This section reviews academic research and commercial solutions in the field of secure communication platforms to identify key innovations and best practices that influenced the design and development of the GEMBOS system. By analyzing both academic studies and existing commercial applications, the GEMBOS project aims to integrate proven methodologies to deliver a robust, scalable, and user-friendly platform.

### 2.1 Academic Papers

Academic research has made significant contributions to secure communication technologies, particularly in the areas of encryption, key exchange protocols, and offline message synchronization. The insights gained from these studies directly informed the design of GEMBOS's core functionalities.

One key reference was the paper "**End-to-End Encryption for Secure Communication Applications**" by Zhang et al. [1], which explores the implementation of encryption protocols such as Elliptic Curve Diffie-Hellman (ECDH) for efficient and secure key exchanges. This research reinforced the decision to integrate ECDH into GEMBOS, ensuring robust data protection during message transmission. Similarly, the paper "**Secure Offline Messaging Using Local Storage and Synchronization Protocols**" by Kaur and colleagues [2] inspired the offline synchronization mechanism in GEMBOS, allowing users to send and store messages locally during periods of no internet connectivity, with automated synchronization upon reconnection.

Another influential study, "**Scalable Architectures for Distributed Messaging Systems**" by Lee et al. [3], outlined best practices for distributed system design to handle increasing user loads while maintaining performance and reliability. The findings from this study guided the development of GEMBOS's distributed architecture, ensuring scalability as the user base grows.

### 2.2 Commercial Solutions/Products

In addition to academic research, GEMBOS drew significant inspiration from existing commercial communication platforms. These solutions provided valuable insights into user expectations, innovative features, and potential challenges in the development of secure messaging systems.

For instance, **Signal**, a popular encrypted messaging platform, uses end-to-end encryption protocols and secure key management to ensure user privacy. Signal's approach to simplicity and security inspired GEMBOS to implement user-friendly interfaces while maintaining advanced encryption features. The use of secure key storage methods, such as Signal's integration with secure enclaves on mobile devices, motivated GEMBOS to adopt similar techniques using Android Keystore and iOS Keychain for encryption key management.

Another commercial solution, **WhatsApp**, provided insights into the importance of seamless cross-platform messaging. WhatsApp's offline message storage and synchronization feature influenced GEMBOS's ability to store messages locally when the user is offline and sync them with the remote database when connectivity is restored.

Finally, **Telegram** highlighted the potential of incorporating additional layers of encryption and user control, such as self-destructing messages and encrypted chats. These features inspired the development of optional advanced security settings within GEMBOS, allowing users to customize their security preferences.

By drawing on insights from both academic research and commercial solutions, GEMBOS integrates state-of-the-art encryption, scalable distributed architectures, and user-centric design to deliver a secure and efficient communication platform tailored to the needs of modern users.

### 3. REQUIREMENTS

The development of the GEMBOS project commenced with the creation of a **Project Assignment Form (PAF)**, outlining key project details, including the project title, team members, and a summary of the proposed system. This document served as the foundation for the subsequent requirements engineering phase, ensuring alignment between the project scope and objectives.

The first iteration of the requirements was formalized in the **Requirements Specification Document (RSD 1.0)**. This initial version was written in a structured text format and provided a detailed description of the functional and non-functional requirements for GEMBOS, including its core features such as secure messaging, offline synchronization, and end-to-end encryption.

To further refine and clarify the project's scope, an updated version, **RSD 2.0**, was developed. This enhanced document incorporated stakeholder feedback and improved the representation of requirements through the use of **Unified Modeling Language (UML) diagrams**. Key UML artifacts included **use case diagrams**, **class diagrams**, and **activity diagrams**, which provided a clear visualization of system components and their interactions. This version addressed additional requirements such as scalability, modularity, and cross-platform compatibility, ensuring the system's adaptability to future enhancements.

The finalized requirements for GEMBOS, as documented in RSD 2.0, form the foundation for the system's design and implementation. These requirements are detailed in **Appendix A**, titled "Requirements Specification Document, version 2.0." They outline the functional expectations, security protocols, and performance standards that guide the development of GEMBOS as a robust, distributed communication platform.

### 4. DESIGN

The design section provides a detailed overview of the architecture and technical framework for the GEMBOS project, a **discrete distributed system**. The system's design ensures modularity, scalability, and maintainability, as well as a focus on secure, efficient, and user-friendly communication. This section outlines the high-level design, technical constraints, and realistic conditions that ensure the project's feasibility while meeting its objectives.

#### 4.1. High Level Design

The high-level design of the GEMBOS system employs a Discrete Distributed System Architecture, which is well-suited for handling the decentralized nature of secure messaging

and data synchronization. The system is designed to operate across multiple devices and nodes while maintaining robust security and performance.

## **Key Components of the Architecture:**

### **Presentation Layer:**

**Role:** Handles user interactions and provides an intuitive interface for messaging and account management.

### **Technology:**

Developed using Java and Swift for cross-platform compatibility (Android and iOS).

**Frameworks:** Android SDK and Swift UI.

### **Features:**

User-friendly interfaces for login, registration, and messaging.

Displays synchronized contacts and notifications.

Provides real-time message status updates (e.g., delivered, read, sent via SMS or Internet).

UI Design: Screens such as Login, Registration, Message List, Chat Interface, and Profile Management are visually designed for ease of use. Refer to Appendix C for UI screenshots.

### **Application Layer:**

**Role:** Acts as the system's core, managing communication, encryption, and interaction between the user interface and the database.

### **Technology:**

**Programming Language:** Java

**Framework:** Spring Boot for robust backend services.

### **Features:**

Encryption and decryption using Elliptic Curve Cryptography (ECC) for secure messaging.

Message synchronization between the local database and the remote server in distributed environments.

Handles secure API communication for data exchange.

**Integration:** Facilitates seamless communication between mobile clients and server components.

### **Data Layer:**

**Role:** Manages persistent data storage and distributed synchronization.

### **Technology:**

**Database Management System:** MySQL for relational data storage.

**Features:**

Stores encrypted user data, messages, and logs.

Supports offline message storage and delayed synchronization in the event of connectivity issues.

Ensures consistency across distributed nodes using secure synchronization mechanisms.

**Justification for Discrete Distributed Architecture:**

**Scalability:** Each layer can be independently scaled to handle increased traffic or data.

**Security:** Distributed architecture minimizes single points of failure and enhances system resilience.

**Flexibility:** New features can be added or existing features updated within individual layers without impacting the overall system.

Detailed specifications of the high-level design are provided in Appendix B: Design Specifications Document.

## **4.2. Detailed Design**

This section will actually be completed in **COMP 4920**.

## **4.3. Realistic Restrictions and Conditions in the Design**

Several constraints and conditions have been considered to ensure the feasibility of the GEMBOS system within the project's scope:

**Data Security:**

The system employs Elliptic Curve Cryptography (ECC) and secure key management through keystores, ensuring robust encryption. However, advanced security measures like multi-factor authentication and biometric login are planned for future iterations.

**Distributed Communication:**

The system operates as a distributed network, synchronizing data between devices. While this ensures reliability, certain trade-offs in latency and resource utilization are inherent to distributed systems.

**User Experience:**

Designed with simplicity in mind, GEMBOS provides intuitive interfaces for a wide range of users. Advanced customization features, such as multilingual support and extensive theming, are deferred to later phases.

**System Performance:**

The system is optimized for moderate loads, with offline synchronization ensuring



uninterrupted functionality during connectivity disruptions. Advanced caching and load-balancing mechanisms are beyond the scope of this phase.

By addressing these considerations, the design ensures that GEMBOS provides a secure, distributed, and user-friendly communication platform, meeting the project's immediate requirements while allowing for future enhancements.

## 5. IMPLEMENTATION AND TESTS

The implementation and testing phase of the GEMBOS project focuses on developing the core functionalities using selected technologies and tools. The primary goals are to implement a robust and scalable communication platform, ensure data security through encryption, and provide a seamless user experience. Comprehensive testing, including unit, integration, and user acceptance testing (UAT), will be conducted to validate system performance, functionality, and scalability.

### 5.1. Implementation of the Product

This section will actually be written and completed in COMP 4920.

The implementation phase of the GEMBOS project will be conducted in the final phase (COMP 4920) and will involve the following key components:

#### Backend Development:

- **Programming Language:** Java

**Framework:** Spring Boot will be used due to its high performance and ability to handle backend services efficiently.

#### Features:

Secure API endpoints for data exchange between the mobile app and the database.

Integration of encryption algorithms (Elliptic Curve Cryptography - ECC) to ensure secure communication.

Offline message storage and synchronization mechanisms to support distributed communication.

- **Frontend Development:**

**Programming Languages:** Java and Swift will be used to develop cross-platform mobile applications for Android and iOS.

#### Frameworks:

- Android SDK for Android devices.
- Swift UI for iOS devices.

**UI Components:** Bootstrap-based design principles will be utilized to ensure a responsive and

user-friendly interface. Key screens include Login, Registration, Messaging, and Profile Management interfaces.

- **Encryption and Security:**

**Encryption Algorithm:** Elliptic Curve Cryptography (ECC) will be employed for secure message encryption and decryption.

**Key Management:** The system will utilize Android Keystore and iOS Keychain to securely store encryption keys on the device.

**Messaging Protocols:** End-to-end encryption will ensure secure communication between users.

- **Database Management:**

**Database Management System:** MySQL will be used to manage structured data, including user profiles, messages, and logs.

**Features:**

Support for offline message storage.

Synchronization with a centralized remote database to ensure data consistency across devices.

- **Notification System:**

A notification manager will handle in-app notifications, ensuring users are alerted about new messages, updates, or status changes in real-time.

These implementation considerations reflect the system's core design and are guided by the specifications outlined in the Requirements and Design Specifications Documents (RSD and DSD).

## 5.2. Tests and Results of Tests

These implementation considerations are initial thoughts and will be further refined and detailed in the next phase of the project, **COMP 4920**.

Unit tests for core functionalities. Integration tests for seamless interaction between components. Real-world testing to assess user experience and scalability.

## 6. PROJECT MANAGEMENT

Effective project management is essential to ensure the smooth progression of the GEMBOS project. This section outlines the detailed plan for the project's activities, including timelines, key tasks, and milestones. Proper management throughout each phase will help achieve the project's objectives within the set deadlines and maintain quality standards.

### 6.1. Project Plan

The project plan for GEMBOS during the Fall semester has been carefully structured to ensure the successful completion of the project's initial phase. This plan outlines a series of key activities, each with defined start and end weeks, as well as a specified duration, ensuring that the project progresses smoothly. The detailed Project Plan is provided in Appendix C1 of the report. Below is a summary of the key activities and their schedules:

### **Initial Phase (Fall 2024):**

#### **1. PAF Production:**

The Project Approval Form (PAF) officially marked the commencement of the GEMBOS project. This activity spanned the first two weeks of the semester, setting the foundation for project objectives and deliverables.

#### **2. Survey of Related Work:**

This task involved conducting extensive research into existing secure messaging systems, SMS protocols, elliptic curve cryptography (ECC), and distributed messaging platforms. It was scheduled over three weeks (weeks 6, 7, and 10), forming a crucial part of the project's initial research phase.

#### **3. RSD 1.0 Production:**

The first version of the Requirements Specification Document (RSD) was developed over fourteen weeks. This document outlined the functional and non-functional requirements of the GEMBOS system, including user registration, encrypted messaging, and offline-to-online message synchronization.

#### **4. RSD 2.0 Revision:**

A subsequent three-week period was allocated to revise and refine the RSD, incorporating feedback, new insights, and the latest research findings in secure communication systems.

#### **5. DSD 1.0 Production:**

The initial Design Specification Document (DSD) was produced over four weeks, outlining the proposed system architecture, including encryption methods (ECC), user interface design, and server-client interactions.

#### **6. Final Report Production:**

Spanning the last two weeks of the semester, this task involved compiling and synthesizing all project activities, research findings, and documentation into the final report, ready for submission.

#### **7. Project Management Activities:**

Throughout the semester, regular team meetings, progress tracking, risk management, and resource allocation were carried out to ensure timely delivery and smooth project progress.

### **Next Phase (Spring 2025):**

#### **1. RSD v2.0 Revision:**

The first task of the Spring semester will be to revise the RSD based on insights gained from the initial semester's work, focusing on refining the functional and non-functional requirements for system implementation.

#### **2. DSD v1.0 Revision:**

The next phase will involve revising the DSD to ensure alignment with the updated requirements and incorporating any new technological advancements.

#### **3. DSD v2.0 as Detailed Design:**

The detailed design phase will involve finalizing the DSD and preparing it for the implementation phase. This will include finalizing the architecture, encryption models, and server-client interactions.

#### **Implementation and Testing Activities:**

This critical phase will focus on the development of the GEMBOS system, including the implementation of encrypted messaging features, user verification, and offline

message handling. Rigorous testing of the system's functionalities will ensure reliability, accuracy, and security.

## 5. Project Management (PM):

Ongoing project management activities will include monitoring, controlling, and closing the project phases to ensure successful delivery and integration of all components.

## 6.2. Project Effort/Manpower

This section is presented in Appendices C2 and C3.

## 6.3. Project Cost Analysis

Currently, there have been no hardware or software purchases for the GEMBOS project. All necessary tools and resources are being utilized through free and open-source platforms, as well as available academic licenses. Should any purchases be required in the future, they will be assessed and documented accordingly.

# 7. CONCLUSIONS

## 7.1. Summary

The GEMBOS project, initiated as part of the Senior Design Project, aims to deliver a secure, scalable, and user-friendly communication platform. By leveraging cutting-edge encryption technologies and distributed system architecture, GEMBOS seeks to provide a reliable and seamless messaging experience that prioritizes user privacy and data security.

Here is a summary and discussion of what has been accomplished so far:

1. **Planning:** The project began with the identification of the need for a secure and distributed communication system. The team outlined the scope, objectives, and deliverables of the project, followed by the development of a detailed project plan.
2. **Requirements:** A comprehensive Requirements Specification Document (RSD) was created, detailing the functional and non-functional requirements of the GEMBOS system. This document serves as the foundation for the project, ensuring alignment with user needs and security objectives.
3. **Research and Analysis:** Extensive research was conducted on existing secure messaging platforms, encryption standards, and distributed system protocols. The team analyzed the feasibility of integrating elliptic curve cryptography and offline messaging support into the platform.
4. **System Design and Architecture:** The system's architecture was designed to ensure modularity, scalability, and security. Key components, such as encryption management, offline synchronization, and distributed database architecture, were defined to meet the project's requirements.
5. **Frontend and Backend Planning:** Preliminary discussions on the frontend and backend structure were conducted. The frontend focuses on delivering an intuitive user interface, while the backend ensures secure message handling and seamless synchronization across devices.
6. **Design Phase:** The Design Specification Document (DSD) was developed, detailing the proposed system architecture, design considerations, and technology stack. UML diagrams were created to illustrate the interactions between components and provide clarity for implementation.
7. **Documentation:** All project documentation, including the RSD, DSD, and initial design drafts, has been compiled to ensure a structured and well-documented process.

In summary, the GEMBOS project has made significant progress during its initial phase. The completed research, planning, and design work have laid a strong foundation for the implementation of a secure and scalable messaging platform. The next steps will focus on the development of the system's core functionalities, refinement of the design, and rigorous testing.

## 7.2. Benefits of the Project

The GEMBOS project offers significant advantages for users, emphasizing secure communication and a seamless user experience. Additionally, by promoting encryption and responsible data handling, GEMBOS aligns with global efforts toward ensuring privacy in the digital communication domain.

### Benefits for Users:

- **Enhanced Security:** GEMBOS uses elliptic curve cryptography and robust encryption techniques to ensure end-to-end secure messaging.
- **Seamless Messaging Experience:** Features like offline synchronization and cross-platform compatibility ensure uninterrupted communication even in low-connectivity scenarios.
- **User Privacy:** GEMBOS uses elliptic curve cryptography and robust encryption techniques to ensure end-to-end secure messaging.
- **Intuitive Design:** The user-friendly interface and simple navigation enhance usability, making secure communication accessible to a broad audience.

## 7.3. Future Work

To be Completed in COMP 4920:

**Detailed Design:** Refine the Design Specification Document (DSD v2.0) by incorporating feedback and finalizing key architectural components, including encryption protocols, backend services, and UI enhancements.

**Core Functionality Implementation:** Develop and integrate core features, such as secure message transmission, offline message storage and synchronization, and distributed database operations.

**Optimization of Encryption Mechanisms:** Optimize elliptic curve cryptography algorithms to enhance performance without compromising security.

**User Interface Improvements:** Refine the UI/UX based on feedback to ensure it is visually appealing, intuitive, and responsive across all devices.

Possible Additions Post-COMP 4920:

**Support for Multiple Languages:** Extend the platform to support multiple languages, catering to a global user base.

**Advanced Security Features:** Explore the integration of multifactor authentication (MFA) and real-time threat detection for enhanced security.

## REFERENCES

1. General Mobile. (2024, November 8). Meeting with representatives from General Mobile regarding potential improvements on our project.
2. Repel Cyber Security. (2024, November 8). Meeting with representatives from Repel Cyber Security regarding potential improvements and future collaboration opportunities on our project.
3. Diffie, W., & Hellman, M. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory*.
4. Miller, V. S. (1986). Use of Elliptic Curves in Cryptography. *Advances in Cryptology – CRYPTO '85 Proceedings*, 417, 417–426. DOI: 10.1007/3-540-39799-X\_31
5. GEMBOS Requirements Specification Document (RSD).

## **APPENDICES**

## **APPENDIX A: REQUIREMENTS SPECIFICATION DOCUMENT**



**COMP4910 Senior Design Project 1, Fall 2024**  
**Advisor: AHMET HASAN KOLTUKSUZ**

**GEMBOS:Global Encrypted Mobile-Based  
Obscured SMS Application**

**23.01.2025**

**Revision 2.0**

**By:**

**Berker Vergi, 21070001202**  
**Giray Aksakal, 21070001030**  
**Mert Kahraman, 20070006020**  
**Emir Morah, 21070006048**

## Revision History

Revision	Date	Explanation
1.0	10.11.2024	Initial requirements
1.1	10.12.2024	Changing AES Algorithms to Elliptic Curve
1.2	20.12.2024	Research system requirements and revisions
1.3	10.01.2025	Adding Use-Case and Activity Diagram
1.4	15.01.2025	Adding Sequence Diagram
1.5	17.01.2025	Final Revision
2.0	24.01.2025	Last time look up and discussion

## Contents

Revision History .....	2
Contents .....	3
1. Introduction to GEMBOS .....	4
2. Functional Requirements .....	5
2.1. Main User Interface and Functions .....	5
2.2. Enter Registration Information .....	8
3. Non-Functional Requirements .....	20
3.1. Development Environment .....	20
3.2. Security .....	20
3.3. Scalability.....	21
3.4. Testing .....	21
The GEMBOS system employs rigorous testing methodologies to ensure the reliability and robustness of its features: .....	21
• Unit Testing: Verifies individual components such as message encryption, user authentication, and contact synchronization to ensure proper functionality.....	21
• Integration Testing: Ensures seamless interaction between various system components, such as backend APIs, database operations, and frontend interfaces.....	21
• User Acceptance Testing (UAT): Involves end-users in testing to validate that the application meets their expectations and requirements. ....	21
3.5. Data Privacy .....	21
4. References.....	21

# 1. Introduction to GEMBOS

## Problem Definition:

In an increasingly interconnected world, communication security has become a critical concern. Traditional SMS systems, while widely adopted, suffer from vulnerabilities such as the SS7 protocol flaw, which can be exploited to intercept messages, manipulate data, or compromise user privacy. These vulnerabilities pose significant risks, especially in sectors where the confidentiality and integrity of communication are paramount, such as banking, legal systems, and healthcare.

The SS7 protocol, which forms the backbone of most mobile communication networks, was designed in an era with limited foresight into the modern security landscape. As a result, it lacks essential security measures to protect against eavesdropping, data manipulation, and unauthorized access. These vulnerabilities are exploited to gain unauthorized access to sensitive information, leading to data breaches, financial fraud, and compromised private communications. Additionally, users face challenges in managing their digital communication securely while maintaining convenience. Existing messaging applications often prioritize user experience over robust security measures, leaving gaps in safeguarding sensitive data. This gap is further exacerbated in high-stakes environments, where secure communication is not a luxury but a necessity.

## Problem Solution:

GEMBOS (Secure Messaging for Banking, Legal, and Healthcare) is designed as a secure and user-friendly SMS application to address these pressing challenges. By integrating advanced cryptographic protocols and robust security measures, GEMBOS ensures the confidentiality, integrity, and authenticity of user communications.

## Key features of GEMBOS include:

1. **Advanced Encryption Algorithms:** The application leverages the Diffie-Hellman algorithm for secure key exchange and the Elliptic Curve for encrypting message content. This ensures that messages are only accessible to intended recipients.
2. **Secure Key Management:** The use of a secure application keystore ensures that encryption keys are managed and stored securely on user devices, reducing the risk of unauthorized access.
3. **Multi-platform Accessibility:** GEMBOS allows users to manage their contacts and send messages seamlessly across mobile and web platforms while maintaining robust security protocols.
4. **Offline Message Handling:** To enhance user experience, GEMBOS provides offline messaging capabilities, enabling users to send and store messages securely even in the absence of an internet connection. These messages are later synchronized with the server when connectivity is restored.
5. **Sector-Specific Applications:** GEMBOS addresses the unique security needs of critical sectors:
  - **Banking:** Secure communication of financial transactions and account details.
  - **Legal Systems:** Protection of confidential case files and client communications.
  - **Healthcare:** Secure transmission of private medical records and patient information.

By providing these features, GEMBOS aims to mitigate the vulnerabilities of traditional SMS systems and cater to the specific security demands of sensitive communication environments.

## Literature Review:

The need for secure messaging systems has been well-documented in academic and industry research. Various studies highlight the vulnerabilities of traditional SMS systems and propose cryptographic methods to address these issues. For example:

- **Zhang et al. [1]:** Proposed a secure SMS system utilizing RSA for encryption, which ensures message confidentiality but introduces computational overhead.
- **Chen et al. [2]:** Investigated the integration of secure key exchange protocols like Diffie-Hellman into messaging applications, emphasizing their role in protecting against man-in-the-middle attacks.

While these approaches provide theoretical frameworks for secure messaging, GEMBOS builds on these advancements by integrating multiple cryptographic measures into a cohesive and practical application. By focusing on user experience alongside robust security, GEMBOS bridges the gap between academic research and real-world application.

### Challenges Addressed by GEMBOS:

1. **Interception and Data Manipulation:** Traditional SMS systems are vulnerable to interception due to the SS7 protocol flaw. GEMBOS overcomes this challenge by encrypting messages end-to-end, ensuring that only authorized parties can access the content.
2. **Key Management:** Secure storage and management of encryption keys are critical to preventing unauthorized access. GEMBOS employs a application keystore mechanism to manage keys locally on user devices.
3. **Cross-platform Compatibility:** Ensuring secure communication across multiple platforms without compromising user experience is a complex challenge. GEMBOS addresses this by implementing platform-agnostic protocols and ensuring seamless synchronization of messages.
4. **Offline Messaging:** In scenarios where internet connectivity is unavailable, users often face difficulties in securely storing and transmitting messages. GEMBOS resolves this by enabling offline message storage and synchronization.
5. **Sector-specific Security Requirements:** The application is tailored to meet the unique demands of critical sectors such as banking, legal, and healthcare, providing customized solutions for secure communication.

### Personalized Approach:

GEMBOS adopts a user-centric approach to secure communication. Recognizing the diverse needs of users, especially in high-stakes environments, the application provides:

- **Customizable Security Settings:** Users can configure encryption settings and access controls based on their specific requirements.
- **Sector-specific Customizations:** For example, healthcare professionals can securely share patient data, while legal professionals can exchange case files without compromising confidentiality.
- **Intuitive User Interface:** Despite its robust security features, GEMBOS maintains a simple and intuitive interface, ensuring accessibility for users with varying technical expertise.

### Conclusion:

In an era where communication security is paramount, GEMBOS stands out as a comprehensive solution to the vulnerabilities of traditional SMS systems. By integrating advanced cryptographic algorithms, secure key management, and sector-specific customizations, GEMBOS ensures the confidentiality, integrity, and authenticity of user communications. With its user-centric design and focus on critical sectors, GEMBOS is poised to redefine secure communication standards in an increasingly digital world.

## 2. Functional Requirements

### 2.1. Main User Interface and Functions

The GEMBOS application provides a streamlined user interface to facilitate secure communication. The main menu includes the following core functionalities:

#### GEMBOS Chatting Operations:

1. **Log in or Register an Account:** Users can access the application by registering a new account or logging into an existing one.
2. **Phone Number Verification with Password:** Ensures that users verify their identity through a secure process combining phone number validation and password entry.
3. **Connect Contacts:** Enables users to sync their phone contacts with the GEMBOS application, making it easier to select recipients for messaging.
4. **Send SMS to Selected Contacts:** Allows users to send SMS messages securely to their chosen contacts through the GEMBOS interface.

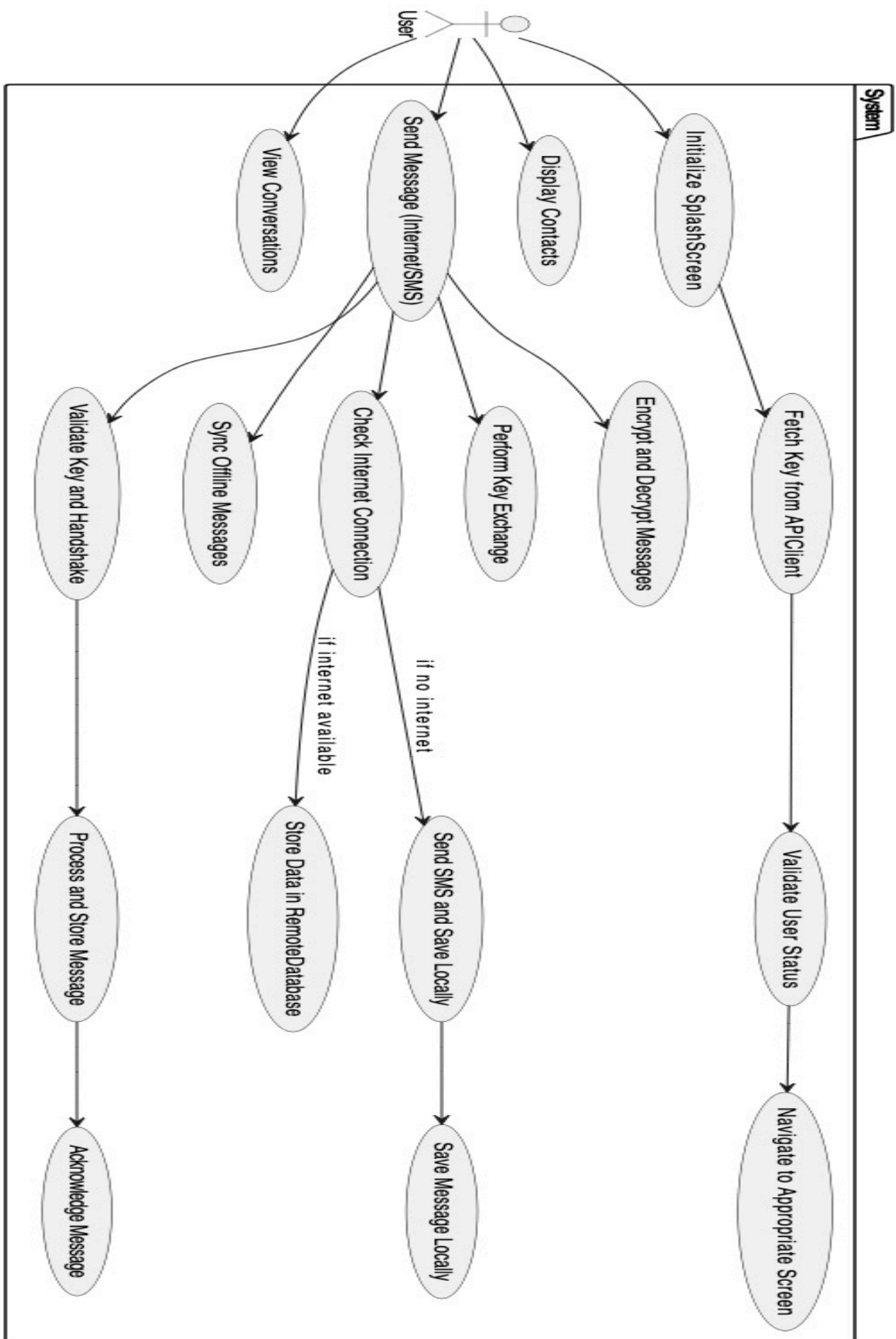


figure 1

## Key Components and System Flow

The Use Case Diagram identifies the key components and processes involved in the GEMBOS application, emphasizing the user's role as the primary actor interacting with the system. The following subsections outline the operations that the user can initiate and the corresponding system responses.

### Actor (User)

The user serves as the central actor, engaging with various system functionalities to achieve secure communication. The main operations initiated by the user include:

- Launching the application via the initialization of the **SplashScreen**.
- Sending messages to contacts using the **Send Message** functionality.
- Viewing previous conversations through the **View Conversations** feature.
- Performing message encryption and decryption for secure communication.
- Verifying internet connectivity before proceeding with message transmission.

### Use Cases

#### Initialize SplashScreen

Upon launching the application, the SplashScreen is initialized. This process acts as the entry point for verifying the user's registration status and directing them to either the main interface or the login/registration screen.

#### Fetch Key from APIClient

The system retrieves encryption keys from the APIClient to facilitate secure communication. If no encryption key exists, the system generates and securely stores a new key, ensuring robust encryption mechanisms for all subsequent operations.

#### Validate User Status

The system performs a check to determine the user's registration status:

- If the user is registered, they are redirected to the main screen.
- If not, they are prompted to log in or register for an account.

#### Display Contacts

The system synchronizes the user's phone contacts, displaying them within the application interface. This feature allows users to select a specific contact for secure communication.

#### Send Message (Internet/SMS)

The system supports two methods of message transmission:

- **When Internet Is Available:** Messages are encrypted using the Diffie-Hellman algorithm, securely stored in the RemoteDatabase, and transmitted to the recipient.
- **When Internet Is Unavailable:** Messages are sent as SMS and stored locally on the device for later synchronization.

#### Perform Key Exchange

To ensure the security of messages, the system executes a key exchange using the Diffie-Hellman algorithm. This process establishes secure communication channels by facilitating encryption and decryption.

#### Check Internet Connection

Before transmitting messages, the system verifies internet connectivity:

- If a connection is available, messages are encrypted and stored in the RemoteDatabase.
- If no connection is available, messages are transmitted as SMS and saved locally.

#### Sync Offline Messages

When internet connectivity is restored, messages stored locally on the device are synchronized with the RemoteDatabase, ensuring no loss of communication data.

### **Validate Key and Handshake**

The system employs a handshake protocol to authenticate encryption keys. This step guarantees data integrity and ensures secure message exchange between users.

### **Process and Store Message**

All transmitted messages are processed and stored in the RemoteDatabase. The system provides feedback to the user upon successful message storage and delivery.

### **Acknowledge Message**

Once the recipient receives a message, the system sends an acknowledgment to the sender, confirming successful delivery.

### **View Conversations**

Users can access and review their message history within the application. During this process, the system decrypts stored messages to make them readable, while maintaining their confidentiality.

### **Technical Features**

#### **Encryption**

The system employs a combination of the Diffie-Hellman key exchange protocol and the Elliptic Curve algorithm to encrypt and decrypt messages. These techniques ensure the confidentiality and security of user data and messages during transmission and storage.

#### **Database Integration**

The RemoteDatabase serves as the central repository for securely storing all user messages and data. Additionally, the system synchronizes locally stored data with the RemoteDatabase whenever internet connectivity is re-established, ensuring seamless integration and data consistency.

#### **Offline Support**

The application provides offline messaging functionality, allowing users to send messages even in the absence of an internet connection. These messages are stored locally and synchronized with the server when a connection becomes available.

#### **Message Processing and Feedback**

The system delivers real-time feedback to users regarding the status of sent messages. For instance, users are notified when their messages are successfully delivered to the recipient.

#### **Advantages of the System**

The GEMBOS system offers several notable advantages:

1. **Security:** The implementation of advanced encryption techniques and a robust handshake protocol ensures the privacy and security of all communications.
2. **Flexibility:** Offline messaging reduces the dependency on constant internet connectivity, enhancing the system's usability in varying conditions.
3. **User-Friendly Design:** The intuitive interface, coupled with seamless contact synchronization, makes the application accessible and convenient for users.

## **2.2. Enter Registration Information**

The registration process is the first interaction users have with the GEMBOS application. This process ensures data integrity, secure account creation, and a seamless user experience.

#### **User Registration Form:**

The registration form requires users to provide the following mandatory information:

- **Phone Number\*:** Ensures that each user account is linked to a unique identifier.
- **Password\*:** Protects the account from unauthorized access.
- **Submit & Cancel Buttons:**



- **Submit:** Validates the entered data, securely registers the user, and redirects them to the main menu.
- **Cancel:** Clears all fields and navigates the user back to the registration screen.

### Login Form:

The login form allows existing users to access the application by providing:

- **Phone Number\***
- **Password\***

The registration and login mechanisms are secured using cryptographic algorithms, ensuring that sensitive user data is protected from unauthorized access.

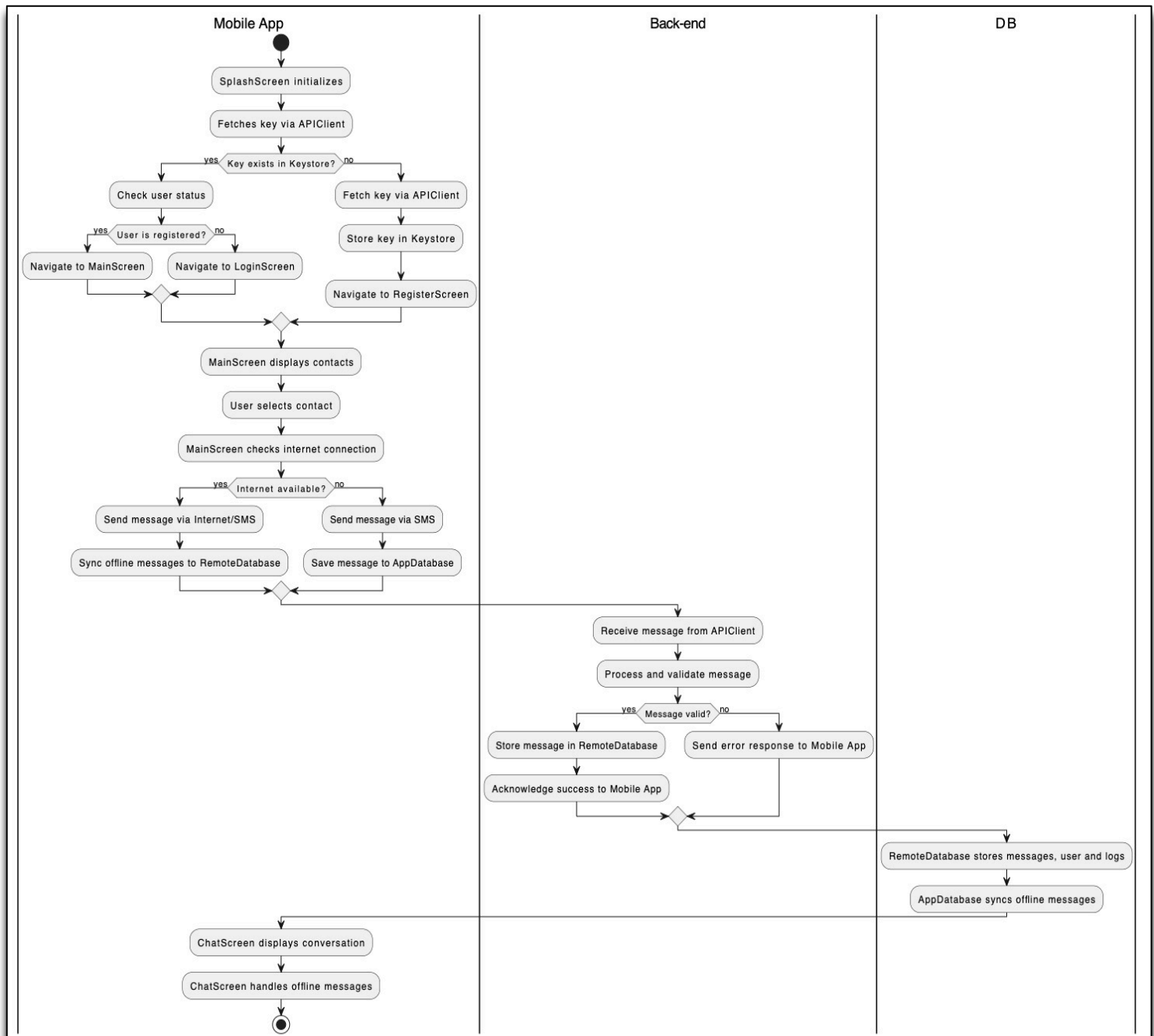


Figure 2

This Activity Diagram illustrates the sequential processes within the GEMBOS system, highlighting the interactions between the **Mobile App**, **Back-end**, and **Database (DB)** layers. It showcases the system's ability to handle secure communication, both online and offline, through coordinated workflows. Each component in the diagram plays a critical role in ensuring the functionality and security of the system.

## 1. Mobile App Layer

The **Mobile App** acts as the interface between the user and the GEMBOS system. It initiates critical operations such as user authentication, contact management, message transmission, and offline data handling. Below are the key actions:

### 1.1 SplashScreen Initialization

- The application begins with the initialization of the **SplashScreen**. This serves as the entry point for system processes, preparing the app to fetch required resources such as encryption keys and user data.
- The initialization ensures that all prerequisites for secure communication are in place before proceeding.

### 1.2 Key Retrieval via APIClient

- The application attempts to retrieve encryption keys stored in the **keystore**:
  - **If a key exists:** It is used for encryption during communication.
  - **If no key exists:** The app fetches a new key from the **APIClient**, which is securely stored for future use.
- This step ensures robust encryption protocols are in place before data transmission begins.

### 1.3 User Status Validation

- The system verifies the user's registration status:
  - **Registered users** are directed to the **MainScreen**.
  - **New users** are navigated to either the **LoginScreen** or **RegisterScreen** to complete the onboarding process.

### 1.4 MainScreen Display

- After successful authentication, the system displays the **MainScreen**, which includes a synchronized list of the user's contacts. This enables the user to seamlessly select a recipient for communication.

### 1.5 Internet Connectivity Check

- Before transmitting messages, the system checks the availability of an internet connection:
  - **If connected:** Messages are encrypted and sent via the internet. They are also stored in the **RemoteDatabase** for centralized logging.
  - **If not connected:** Messages are sent via SMS and stored locally in the **AppDatabase** for later synchronization.

### 1.6 Sync Offline Messages

- When an internet connection is restored, messages stored in the local **AppDatabase** are synchronized with the **RemoteDatabase**, ensuring that the communication history remains consistent and accessible.

### 1.7 ChatScreen Operations

- The **ChatScreen** manages the display of conversations, including messages sent both online and offline. Users can view past interactions while the system ensures the secure handling of all data.

## 2. Back-end Layer

The **Back-end** layer processes messages transmitted by the Mobile App and ensures their validity and security. Key actions include:

### 2.1 Receiving Messages

- Messages sent from the Mobile App are received by the back-end system via the **APIClient**. This step ensures the secure handoff of user data for processing.

### 2.2 Message Validation

- The back-end validates each message for integrity and compliance with security protocols:
  - **Valid messages** proceed to storage in the **RemoteDatabase**.
  - **Invalid messages** trigger an error response, which is sent back to the Mobile App to inform the user.

## 2.3 Message Storage

- All valid messages are securely stored in the **RemoteDatabase**. This ensures centralized data management, enabling consistency across devices.

## 2.4 Acknowledgment

- After successfully storing a message, the back-end sends a confirmation to the Mobile App. This provides real-time feedback to the user, confirming that their message has been delivered and logged.

## 3. Database Layer

The **Database (DB)** layer serves as the central repository for managing user and communication data. It comprises the **RemoteDatabase** and **AppDatabase**, each with specific responsibilities:

### 3.1 RemoteDatabase

- The **RemoteDatabase** securely stores all messages, user data, and logs. This centralized storage ensures that communication history is preserved and accessible across devices.

### 3.2 AppDatabase Synchronization

- The **AppDatabase**, located on the user's device, temporarily stores messages and user data when offline. Upon regaining internet connectivity, the system synchronizes the locally stored data with the **RemoteDatabase**, ensuring consistency and preventing data loss.

## Key Technical Processes

1. **Encryption and Key Management:**
  - Encryption keys are securely managed via the **APIClient** and stored in the keystore. Messages are encrypted using Diffie-Hellman and Elliptic Curve algorithms, ensuring end-to-end security.
2. **Offline Messaging Support:**
  - The system seamlessly transitions between online and offline modes. Offline messages are stored locally and synchronized with the RemoteDatabase when connectivity is restored.
3. **Real-Time Feedback:**
  - The system provides users with real-time status updates for sent messages, ensuring transparency and enhancing the user experience.

## Advantages of the Workflow

1. **Security:** The use of robust encryption protocols and secure key management ensures that all communication remains private and protected.
2. **Reliability:** Offline messaging and synchronization prevent data loss, ensuring that communication remains uninterrupted.
3. **User Convenience:** The intuitive interface and seamless integration of contacts make the application user-friendly and accessible.
4. **Scalability:** The centralized database architecture supports consistent data management across devices, allowing for future scalability.

## 2.3. Connect to Contacts

This feature allows users to integrate their phone contacts with the GEMBOS application. Synced contacts are displayed within the application interface, enabling users to:

- View a list of all synced contacts.
- Select specific contacts for secure messaging.
- Synchronize contacts seamlessly across multiple devices.

The contact synchronization process employs secure communication protocols to prevent unauthorized access or data leaks during transfer.

## 2.4. Two-Factor Authentication and Messaging

### Two-Factor Authentication (2FA):

GEMBOS implements a robust two-factor authentication system to verify user identity:

1. Users provide their phone number during login or registration.
2. The system sends a one-time password (OTP) via SMS to the provided phone number.
3. Users must enter the OTP within a given time frame to complete the verification process.

### Messaging Feature:

The messaging functionality allows users to securely send SMS messages to their selected contacts. Key features include:

- **End-to-End Encryption:** Messages are encrypted using the Diffie-Hellman key exchange protocol, ensuring secure transmission. The Elliptic Curve algorithm is employed for message encryption and decryption, guaranteeing data confidentiality.
- **Cross-Platform Messaging:** Users can send messages through the mobile or web interface.
- **Key Management:** Encryption keys are securely stored in Android and iOS keystores, enhancing security for both platforms.
- **Offline Support:** In cases where internet connectivity is unavailable, messages are stored locally and synced to the server once the connection is restored.

The messaging process includes an initial handshake between users to establish a secure communication channel. This handshake ensures that encryption keys are exchanged securely before any data transmission occurs.



The provided class diagram illustrates the structural design of the GEMBOS system, focusing on its key components, their relationships, and the methods that define their interactions. This diagram offers a comprehensive view of how the system manages secure communication, user interactions, and backend processes.

## Key Classes and Their Responsibilities

### 1. SplashScreen

- **Purpose:** Serves as the entry point for the application.
- **Key Methods:**
  - `initialize()`: Prepares the application by loading necessary resources and initializing components.
  - `navigateIfRegistered()` and `navigateIfNewUser()`: Determine whether to direct the user to the main interface or the registration screen, based on their status.

### 2. MainScreen

- **Purpose:** Acts as the primary user interface after successful login or registration. Displays synchronized contacts and allows users to initiate communication.
- **Key Methods:**
  - `displayContacts(contactList: List<User>)`: Retrieves and displays the user's synchronized contacts.
  - `startNewChat(contact: User)`: Initiates a chat session with a selected contact.
  - `showNotification(notification: Notification)`: Manages in-app notifications.

### 3. LoginScreen & RegisterScreen

- **LoginScreen:**
  - Handles user authentication by validating their credentials.
  - **Key Methods:**
    - `loginUser(userId: String, password: String)`: Authenticates the user and directs them to the main screen if successful.
- **RegisterScreen:**
  - Facilitates new user registration by collecting necessary details.
  - **Key Methods:**
    - `registerUser(userDetails: User)`: Validates and stores user information.

### 4. ChatScreen

- **Purpose:** Manages individual chat sessions, handling message display, input, and synchronization.
- **Key Methods:**
  - `sendMessage(message: String)`: Sends messages to the selected contact.
  - `saveOfflineMessage(message: Message)`: Stores messages locally when offline.
  - `fetchMessages(chatHistory: List<Message>)`: Retrieves previous chat logs for display.

### 5. MessageManager

- **Purpose:** Orchestrates the sending and receiving of messages, ensuring secure transmission.
- **Key Methods:**
  - `sendMessage(content: String, recipient: User)`: Prepares and encrypts messages for transmission.
  - `fetchMessages(chatId: String)`: Retrieves message history from the database.
  - `verifyMessage(content: String)`: Validates message integrity before processing.

## 6. EncryptionManager

- **Purpose:** Ensures all communications are encrypted for security.
- **Key Methods:**
  - generateKeyPair(): Creates a secure key pair for encryption.
  - encryptMessage(message: String, key: String): Encrypts messages before transmission.
  - decryptMessage(encryptedMessage: String, key: String): Decrypts incoming messages.

## 7. Keystore

- **Purpose:** Manages encryption keys securely within the application.
- **Key Methods:**
  - storeKey(key: String): Saves encryption keys for future use.
  - retrieveKey(): Fetches saved keys for message decryption.

## 8. RemoteDatabase & AppDatabase

- **RemoteDatabase:**
  - Acts as the central repository for all messages and user data, accessible across devices.
  - **Key Methods:**
    - saveMessage(message: Message): Stores messages in the database.
    - syncOfflineMessages(messages: List<Message>): Synchronizes locally stored messages with the server.
- **AppDatabase:**
  - Handles local data storage for offline usage.
  - **Key Methods:**
    - saveOfflineMessage(message: Message): Temporarily saves messages until internet connectivity is restored.

## 9. SmsManager

- **Purpose:** Facilitates SMS-based communication when internet connectivity is unavailable.
- **Key Methods:**
  - sendSMS(message: String, recipient: User): Sends messages via SMS.
  - receiveSMS(): Handles incoming SMS messages.

## 10. NotificationManager

- **Purpose:** Manages system notifications for the user.
- **Key Methods:**
  - throwNotification(notification: Notification): Sends notifications for events such as new messages or updates.
  - manageNotifications(notificationQueue: List<Notification>): Queues and handles multiple notifications.

## Relationships Between Classes

### 1. Direct Interactions:

- The MainScreen directly interacts with the ChatScreen to handle communication-related tasks.
- The MessageManager communicates with the EncryptionManager for encrypting and decrypting messages before they are sent or displayed.

### 2. Centralized Data Management:

- The RemoteDatabase serves as the central hub for storing messages, ensuring data consistency across devices.
- The AppDatabase bridges the gap during offline usage, synchronizing with the RemoteDatabase when connectivity is restored.

### 3. **Security-Focused Interactions:**

- The Keystore and EncryptionManager collaborate to ensure that encryption keys are securely generated, stored, and retrieved for secure messaging.

## **Key Features of the Class Diagram**

### 1. **Security and Encryption:**

- The integration of EncryptionManager and Keystore highlights the system's focus on secure communication. The use of encryption protocols ensures that messages remain private and protected.

### 2. **Offline and Online Support:**

- The interaction between AppDatabase and RemoteDatabase ensures seamless communication even when the user is offline. Messages are stored locally and synchronized later, preventing data loss.

### 3. **User-Centric Design:**

- Classes like MainScreen, LoginScreen, and ChatScreen ensure a user-friendly interface, prioritizing ease of access and smooth navigation.

### 4. **Extensibility:**

- The modular design of the system enables easy scalability. New features or enhancements can be integrated into specific classes without affecting the overall architecture.



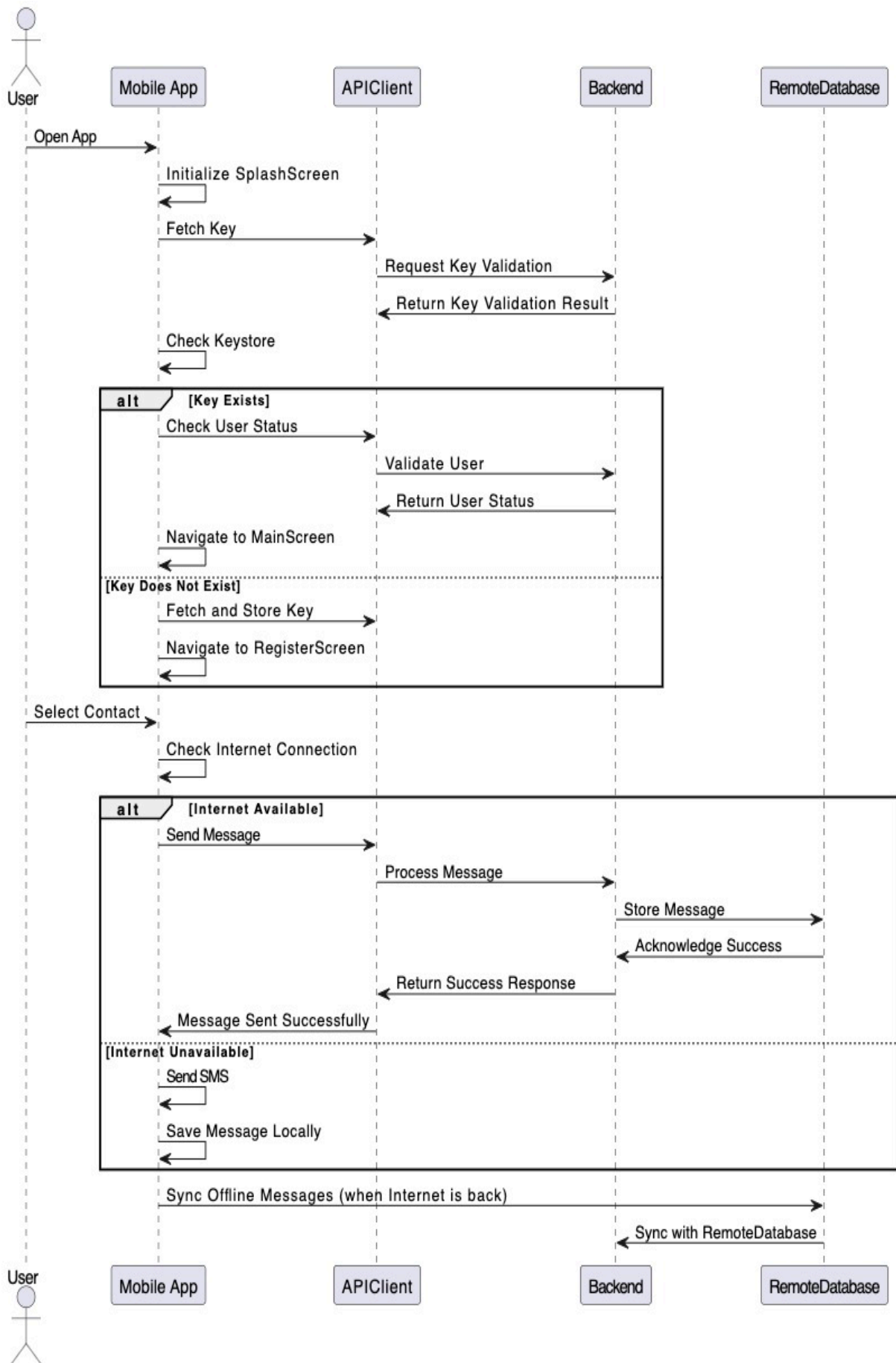


Figure 4

This sequence diagram illustrates the flow of interactions between the **User**, **Mobile App**, **APIClient**, **Backend**, and **RemoteDatabase** in the GEMBOS system. It highlights the process of application initialization, user authentication, message sending, and offline synchronization, ensuring a secure and seamless communication experience.

## Key Components and Their Roles

1. **User:**  
The primary actor initiating interactions with the system, such as opening the application, sending messages, and selecting contacts.
2. **Mobile App:**  
The client-side interface that facilitates user actions, including authentication, contact management, and message transmission. It serves as the gateway to the backend services via the **APIClient**.
3. **APIClient:**  
Acts as the intermediary between the mobile app and backend, handling API requests, key validation, and secure communication.
4. **Backend:**  
Processes user requests, validates keys and user data, and ensures secure storage and retrieval of messages.
5. **RemoteDatabase:**  
Provides centralized storage for user data and messages, enabling synchronization across devices and ensuring data consistency.

## Detailed Flow Description

### 1. Application Initialization

- The process begins with the **User** opening the application on the **Mobile App**.
- The **Mobile App** initializes the **SplashScreen**, preparing the application for operation.
- The app then fetches the encryption key from the **Keystore**:
  - **Alternative Flow: Key Validation**
    - If the key exists, the **APIClient** requests validation from the **Backend**, which verifies the key and returns the result.
    - If the key does not exist, the **Mobile App** fetches a new key from the **APIClient**, stores it securely in the keystore, and navigates the user to the **RegisterScreen** for account creation.

### 2. User Authentication

- If a valid key is present, the system checks the user's status:
  - **Registered User:** The user is directed to the **MainScreen**, where synchronized contacts are displayed.
  - **Unregistered User:** The user is navigated to the **RegisterScreen** for registration.

### 3. Message Sending

- The **User** selects a contact from the **MainScreen** and initiates the message sending process.
- The **Mobile App** checks the internet connection:
  - **Alternative Flow: Internet Available**
    - The **Mobile App** encrypts the message and sends it via the **APIClient** to the **Backend**.
    - The **Backend** processes the message, stores it in the **RemoteDatabase**, and returns a success acknowledgment.
    - The **Mobile App** notifies the user that the message was successfully sent.
  - **Alternative Flow: Internet Unavailable**
    - If no internet connection is available, the **Mobile App** sends the message via SMS and stores it locally.

- The stored message will be synchronized with the **RemoteDatabase** once the internet connection is restored.

#### 4. Offline Synchronization

- When the internet connection is re-established, the **Mobile App** automatically syncs locally stored messages with the **RemoteDatabase** via the **APIClient** and **Backend**.
- This ensures that all messages are logged in the centralized database for consistency and future access.

#### Key Features Highlighted in the Diagram

1. **Secure Key Management:**
  - The system validates and securely stores encryption keys using the **Keystore** and **APIClient**, ensuring robust data protection.
2. **Seamless User Experience:**
  - The application dynamically adapts to the user's registration status, guiding them to the appropriate screen (MainScreen or RegisterScreen).
3. **Dual Message Transmission Modes:**
  - The system supports both online (via the internet) and offline (via SMS) communication, ensuring uninterrupted messaging capabilities.
4. **Offline Support and Synchronization:**
  - Locally stored messages are automatically synchronized with the **RemoteDatabase** when the internet becomes available, maintaining data consistency.
5. **Backend Processing and Acknowledgment:**
  - The backend ensures messages are securely processed, stored, and acknowledged, providing feedback to the user.

#### Summary of Technical Features:

1. **Secure Registration and Login:** The system implements robust security measures for user authentication, combining OTP (One-Time Password) verification and password-based access control. This dual-layered approach ensures that only verified users can access the application, thereby safeguarding user identity.
2. **Contact Integration:** The application facilitates seamless synchronization of user contacts from their device. These contacts are securely stored within the system, allowing users to easily select recipients for communication while maintaining data integrity and privacy.
3. **End-to-End Encryption:** Secure communication is achieved through the use of advanced encryption techniques. The system employs the Diffie-Hellman algorithm for secure key exchange and the Elliptic Curve algorithm for message encryption and decryption, ensuring that all communications remain confidential and tamper-proof.
4. **Cross-Platform Functionality:** GEMBOS offers a consistent and cohesive user experience across multiple platforms, including mobile and web interfaces. This ensures that users can seamlessly switch between devices without compromising functionality or security.
5. **Offline Message Handling:** The system supports the storage of messages locally when internet connectivity is unavailable. These messages are automatically synchronized with the central database once connectivity is restored, ensuring uninterrupted communication and preventing data loss.

These functionalities collectively ensure that GEMBOS provides a secure, efficient, and user-centric communication platform, meeting the diverse needs of its users.

### 3. Non-Functional Requirements

#### 3.1. Development Environment

##### Frontend - MOBILE

The GEMBOS application utilizes modern technologies for mobile frontend development to ensure seamless user interaction and a robust interface:

- **Languages:** Java, Swift.  
These languages are used to develop Android and iOS applications, providing native support and optimal performance on their respective platforms.
- **Frameworks/Libraries:**
  - **Android SDK:** Enables efficient development of Android applications with rich features and system-level integration.
  - **SwiftUI:** Simplifies UI design for iOS apps with declarative syntax and seamless integration into the Apple ecosystem.
  - **Android Keystore:** Manages cryptographic keys in a secure environment for Android devices, ensuring data protection.
  - **iOS Keychain:** Provides a secure mechanism for storing sensitive data like encryption keys on Apple devices.

##### Backend

The backend infrastructure is designed to handle secure communication and robust data processing:

- **Languages:** Java.  
Chosen for its reliability, scalability, and compatibility with enterprise-grade backend development.
- **Frameworks:**
  - **Spring Boot:** Provides a lightweight and modular architecture, enabling rapid development of RESTful APIs and handling complex backend operations.

##### Databases

**SQL:** MySQL is the primary database management system used to handle structured data, ensuring reliability, scalability, and efficiency for GEMBOS's backend operations.

##### Database Management Strategy:

- Relational database management to handle user profiles, contact information, and message histories.
- Periodic backup strategies, including daily incremental backups and weekly full backups, to ensure data recovery in case of failures.

##### DevOps and Hosting

To streamline development and ensure reliable hosting, the following tools and platforms are employed:

- **GitHub:** Used for version control, collaboration, and code repository management.
- **AWS (Amazon Web Services):** Provides scalable and reliable cloud hosting for backend services, ensuring high availability and performance.

#### 3.2. Security

The GEMBOS system places a strong emphasis on security to safeguard user communication and data:

- **SSL (Secure Sockets Layer):** Encrypts data during transmission to protect against interception.
- **Password Hashing:** All user passwords are hashed using secure algorithms (e.g., SHA-256 or bcrypt) to prevent unauthorized access.
- **API Key Management:** API keys are securely stored in environment variables, reducing the risk of exposure.
- **Encryption Algorithms:**
  - **Diffie-Hellman Algorithm:** Ensures secure key exchange between users for encrypted communication.
  - **Elliptic Curve Cryptography (ECC):** Provides strong encryption for securing sensitive data, ensuring confidentiality, integrity, and efficiency with smaller key sizes compared to traditional methods.

- **Handshake Protocol:** Establishes a secure connection between users before initiating any messaging, ensuring mutual authentication.

### 3.3. Scalability

The system is designed with scalability to handle increasing user demands and data growth:

- **Scalable Database:** MySQL's robust architecture supports horizontal scaling, ensuring it can manage growing user data, including message logs and contact lists.
- **API Architecture:** RESTful APIs are designed to efficiently handle increased traffic and concurrent requests, maintaining high performance.

### 3.4. Testing

The GEMBOS system employs rigorous testing methodologies to ensure the reliability and robustness of its features:

- **Unit Testing:** Verifies individual components such as message encryption, user authentication, and contact synchronization to ensure proper functionality.
- **Integration Testing:** Ensures seamless interaction between various system components, such as backend APIs, database operations, and frontend interfaces.
- **User Acceptance Testing (UAT):** Involves end-users in testing to validate that the application meets their expectations and requirements.

### 3.5. Data Privacy

The system prioritizes the privacy of user data, ensuring compliance with industry standards and regulations:

- **Data Encryption:** All user data, including messages and contact information, is encrypted at rest and during transmission using Elliptic Curve encryption.
- **Secure Storage:** Sensitive information, such as user credentials and encryption keys, is securely stored in the Android Keystore and iOS Keychain.
- **Privacy Policies:** The system ensures compliance with data protection regulations, such as GDPR (General Data Protection Regulation), to safeguard user rights.

## 4. References

1. General Mobile. (2024, November 8). Meeting with representatives from General Mobile regarding potential improvements on our project.
2. Repel Cyber Security. (2024, November 8). Meeting with representatives from Repel Cyber Security regarding potential improvements and future collaboration opportunities on our project.
3. Diffie, W., & Hellman, M. (1976). **New Directions in Cryptography**. IEEE Transactions on Information Theory.
4. Miller, V. S. (1986). **Use of Elliptic Curves in Cryptography**. Advances in Cryptology – CRYPTO '85 Proceedings, 417, 417–426. DOI: 10.1007/3-540-39799-X\_31

## **APPENDIX B: DESIGN SPECIFICATION DOCUMENT**

**COMP4910 Senior Design Project 1, Fall 2024**  
**Advisor: Assoc. Prof. Dr. Ahmet Hasan Koltuksuz**



**GEMBOS:Global Encrypted Mobile-Based  
Obscured SMS Application  
High Level Design  
Design Specifications Document**

**Revision 1.0**  
**20.01.2025**

**By:**  
**Berker Vergi, 21070001202**  
**Giray Aksakal, 21070001030**  
**Mert Kahraman, 20070006020**  
**Emir Morali, 21070006048**

**Revision History**

Revision	Date	Explanation
1.0	20.01.2025	Initial high level design



## Table of Contents

Revision History	1
Table of Contents	2
1. Introduction	3
2. GEMBOS System Design	4
3. GEMBOS Software Subsystem Design	6
3.1. GEMBOS Software System Architecture	6
3.2. GEMBOS Software System Structure	9
3.3. GEMBOS Software System Environment	19
4. GEMBOS Software System Detailed Design:	21
5. Testing Design	21
References	22

## 1. Introduction

### Purpose:

The purpose of this project is to design and implement the GEMBOS application as a secure, distributed messaging system, building upon the requirements outlined in the GEMBOS Requirements Specification Document (RSD). GEMBOS is designed to address the challenges of secure communication, offline support, and real-time synchronization in a distributed environment. This Detailed Software Design (DSD) document provides a comprehensive overview of the architectural, structural, and implementation details of the GEMBOS application, leveraging **Elliptic Curve Cryptography (ECC)** for encryption, distributed databases for scalability, and a modular design for flexibility and extensibility.

### Main Functions of the GEMBOS System:

#### 1. User Authentication and Account Management:

- Secure login, registration, and logout functionalities.
- Phone number verification through two-factor authentication (2FA) using OTPs.
- Password management, including reset functionality with secure SMS-based verification.

#### 2. Messaging System:

- Support for secure, end-to-end encrypted messages using Elliptic Curve Cryptography (ECC).
- Offline message storage and synchronization upon reconnection.
- Dual transmission modes (internet-based and SMS-based) for reliable communication.

#### 3. Contact Integration:

- Synchronization of phone contacts with the GEMBOS application.
- Display of synchronized contacts with the ability to initiate secure chats.

#### 4. Distributed Data Storage:

- Centralized RemoteDatabase for real-time message storage.
- Local AppDatabase for offline storage and synchronization.
- Scalable architecture to manage distributed data flow.

#### 5. Notifications:

- In-app and system notifications for new messages, updates, and system events.
- Management of queued notifications for seamless user interaction.

#### 6. Security Features:

- Implementation of handshake protocols for secure key exchanges.
- Encryption and decryption of messages using ECC for enhanced data security.
- Secure storage of encryption keys in platform-specific keystores (Android Keystore and iOS Keychain).

## Design Basis and Methodology:

The design of GEMBOS is derived from the GEMBOS Requirements Specification Document (RSD), Version 1.5, dated January 2025. The design process adheres to established organizational standards and software development best practices to ensure a secure and efficient system. This DSD document extends the RSD by providing detailed architectural decisions, component breakdowns, and technical specifications.

The GEMBOS design methodology emphasizes compliance with ISO/IEC 27001 standards for information security management. These standards ensure that the system is designed to meet security, scalability, and usability requirements while maintaining a high level of performance and reliability. Unified Modeling Language (UML) is used extensively throughout this document to represent the system's architecture, interactions, and class structures.

By employing UML and adhering to international standards, the GEMBOS design ensures:

1. **Security:** Robust encryption mechanisms, secure key management, and protection against unauthorized access.
2. **Scalability:** Distributed system architecture that supports growth in user base and data volume.
3. **Usability:** A user-centric design approach for intuitive navigation and seamless user experience.
4. **Performance:** Efficient message processing and real-time synchronization across devices.

This DSD document serves as a comprehensive guide for the development and implementation of the GEMBOS system, ensuring that it aligns with the requirements and goals specified in the RSD while adhering to industry best practices.

## 2. GEMBOS System Design

The GEMBOS system design focuses on the software subsystem, as no specialized hardware components are required for the development and deployment of this project. This section provides an in-depth overview of the system, detailing its components and their interactions. The design adopts a distributed architecture to ensure scalability, maintainability, and extensibility, aligning with the secure and user-friendly principles of the GEMBOS application.

The system is visualized using a UML Component Diagram, to represent the key components and their relationships. The major components include **User Interface Modules**, **Communication Modules**, **Security Modules**, and **Data Management Systems**. These components collaboratively enable secure and reliable messaging within a distributed architecture, ensuring seamless user experience and data integrity.

### Key Components and Their Roles

1. **User Interface Modules**
  - **SplashScreen:** The entry point for the application, responsible for initializing the application environment and directing the user to the appropriate screen (registration or main menu) based on their status.
  - **LoginScreen and RegisterScreen:** Handle user authentication and account creation. These screens interface with the backend through APIClient to validate credentials and register new users securely.
  - **MainScreen:** Displays synchronized contacts, notifications, and the primary communication interface for users.
2. **Communication Modules**
  - **ChatScreen:** Manages user-to-user communication, including real-time message exchange and offline message handling.
  - **SmsManager:** Handles SMS-based communication when internet connectivity is unavailable, ensuring uninterrupted messaging capabilities.

3. **Security Modules**
  - **EncryptionManager:** Implements Elliptic Curve Cryptography (ECC) for encrypting and decrypting messages, ensuring secure end-to-end communication.
  - **Keystore:** Manages encryption keys securely, enabling safe storage and retrieval of keys for message processing.
  - **APIClient:** Acts as the intermediary between the mobile application and the backend, ensuring secure communication via APIs and handshake protocols.
4. **Data Management Systems**
  - **RemoteDatabase:** Serves as the centralized storage for user data and messages, enabling synchronization across devices and maintaining data integrity.
  - **AppDatabase:** Provides local storage for offline functionality, temporarily saving messages and syncing them with the RemoteDatabase when connectivity is restored.
5. **Notification Modules**
  - **NotificationManager:** Manages system notifications, alerting users of new messages, updates, or system events.

## Component Interactions

The GEMBOS system is structured around well-defined interactions between components to ensure a smooth and secure user experience:

- **User Authentication:**  
The SplashScreen interacts with APIClient to validate encryption keys and user credentials, directing users to the appropriate interface (MainScreen or RegisterScreen) based on their authentication status.
- **Message Exchange:**  
Messages are encrypted by EncryptionManager using keys stored in the Keystore before being sent through APIClient to the RemoteDatabase. The ChatScreen displays messages and saves unsent messages locally via AppDatabase during offline periods.
- **Offline Synchronization:**  
When the internet is restored, AppDatabase interacts with RemoteDatabase through APIClient to synchronize locally stored messages, ensuring consistency across devices.
- **Security Handshake:**  
A handshake protocol, facilitated by APIClient and EncryptionManager, ensures that secure keys are exchanged and validated before any data transmission.
- **Notification Management:**  
Notifications are processed by NotificationManager and displayed on the MainScreen for immediate user attention, enhancing real-time communication.

## Design Principles

1. **Scalability:**  
The distributed architecture ensures the system can handle increasing user data and concurrent messaging operations without performance degradation.
2. **Security:**  
The implementation of ECC, secure key storage, and handshake protocols guarantees end-to-end encryption and robust data protection.
3. **Extensibility:**  
The modular design allows for the seamless integration of new features, such as advanced analytics or additional communication channels, without affecting the existing architecture.
4. **User-Centric Approach:**  
The interface design prioritizes ease of use, ensuring that users can navigate the system intuitively while maintaining secure communication.

This system design ensures that GEMBOS meets its functional and non-functional requirements while providing a secure, efficient, and user-friendly messaging platform. The accompanying component diagram visualizes the relationships between these modules, showcasing the architectural flow of the system.

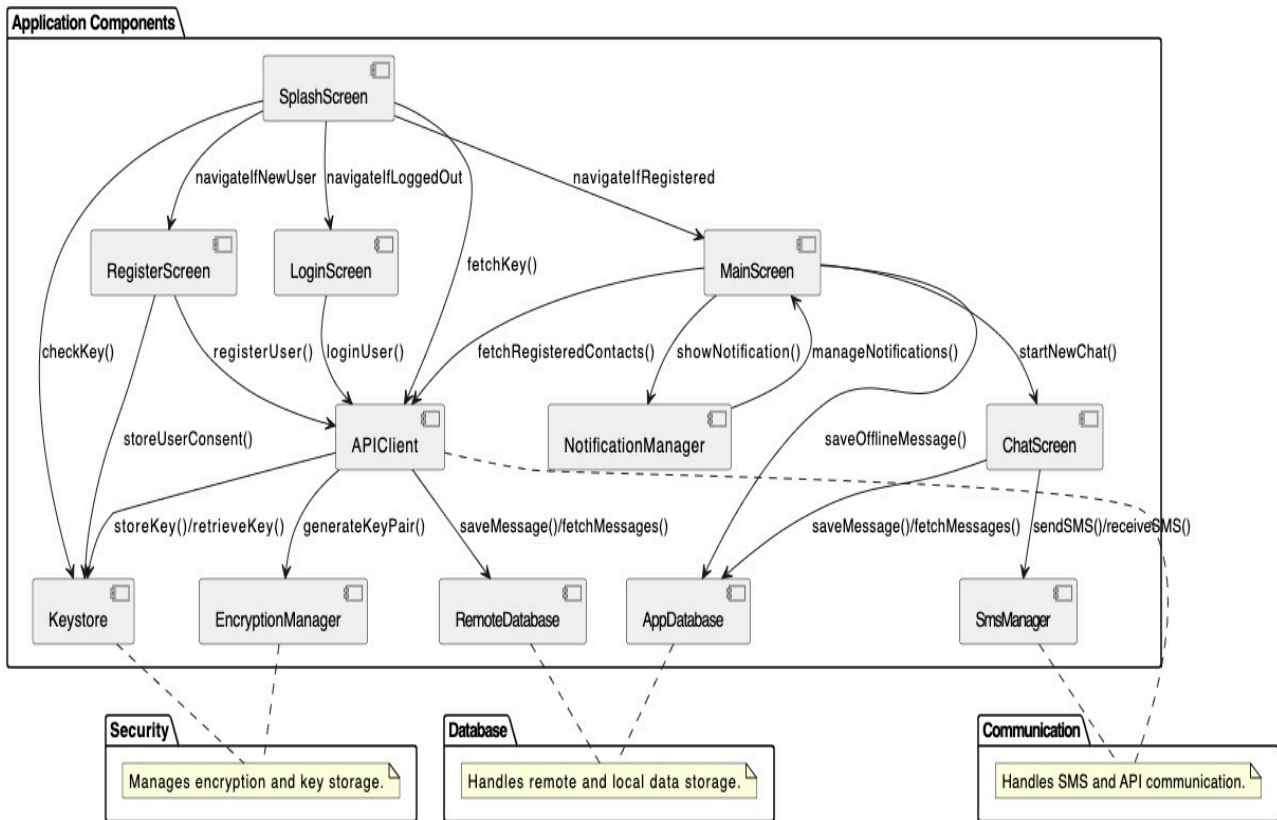


Figure 5 Component Diagram of GEMBOS Software System

### 3. GEMBOS Software Subsystem Design

As the GEMBOS project is a software-only system, this section focuses exclusively on the design of its software subsystem, encompassing the entire architecture of the system. Below is a detailed explanation of the architectural style chosen for the GEMBOS system, along with a justification for the associated design decisions.

#### 3.1. GEMBOS Software System Architecture

The GEMBOS system employs a **Distributed Layered Architecture**, which is well-suited for secure and scalable communication systems. The architecture is organized into three primary layers, promoting modularity, separation of concerns, and adaptability to future requirements.

##### Architecture Layers

##### 1. Presentation Layer

- **Role:** Handles user interactions and serves as the gateway for accessing the system's features.
- **Technology:**
  - Developed using **Java** (Android) and **Swift** (iOS).
  - Android SDK and Swift UI frameworks for building dynamic, platform-native user interfaces.
- **Features:**
  - Manages user interactions, such as login, registration, message sending, and offline message viewing.

- Displays data retrieved from the backend using RESTful APIs.
- Provides a responsive and user-friendly interface for seamless communication.

## 2. Application Layer

- **Role:** Acts as the intermediary between the presentation and data layers, processing business logic and managing secure communication.
- **Technology:**
  - Implemented in **Java** using the **Spring Boot** framework.
  - Integrates secure communication protocols, such as the Elliptic Curve Cryptography (ECC) algorithm and handshake protocols.
- **Features:**
  - Processes user inputs and applies security protocols (e.g., encryption and key validation).
  - Manages message handling, including encryption, offline storage, and synchronization.
  - Ensures secure communication between the frontend and distributed data layers.

## 3. Data Layer

- **Role:** Responsible for persistent data storage, retrieval, and synchronization.
- **Technology:**
  - **MySQL** for relational database management.
  - Distributed database architecture to support scalability and fault tolerance.
- **Features:**
  - Stores user credentials, message history, and encryption keys securely.
  - Synchronizes offline data with the central RemoteDatabase when connectivity is restored.
  - Ensures data integrity and consistency through robust indexing, constraints, and backup strategies.

## Justification for Distributed Layered Architecture

### 1. Scalability:

- The layered structure enables each layer to scale independently, allowing the system to handle increasing user traffic and data volumes efficiently.

### 2. Security:

- The application layer enforces secure communication using ECC, while the data layer ensures encryption and controlled access, minimizing security risks.

### 3. **Maintainability:**

- Clear separation of concerns between layers facilitates easy debugging, updates, and integration of new features.

### 4. **Adaptability:**

- The architecture accommodates future enhancements, such as advanced analytics, new communication channels, or additional security protocols.

## **Design Decisions and Justification**

The design of the GEMBOS system is driven by the requirements for scalability, security, and user-centered functionality. The following decisions were made to align with these goals:

### 1. **Layered Architecture**

- **Reason:** Promotes modularity and clear separation of responsibilities, allowing independent development and testing of each layer.
- **Benefits:** Simplifies debugging and maintenance while facilitating the addition of new features.

### 2. **Integration of Distributed Systems**

- **Reason:** A distributed database architecture was chosen to ensure fault tolerance and support for multiple devices.
- **Benefits:** Enables high availability and consistency of data across users and devices.

### 3. **End-to-End Encryption with ECC**

- **Reason:** ECC provides robust encryption with smaller key sizes, reducing computational overhead while ensuring data security.
- **Benefits:** Strengthens privacy and protects against unauthorized access during message transmission.

### 4. **Scalable Database Design**

- **Reason:** MySQL was selected for its efficiency in handling relational data and support for distributed architectures.
- **Benefits:** Facilitates efficient data retrieval and ensures the system can accommodate growth in user numbers and message volumes.

### 5. **Frontend Technology (Java and Swift)**

- **Reason:** Java and Swift provide platform-native support for Android and iOS, ensuring optimal performance and user experience.
- **Benefits:** Delivers a responsive and dynamic interface tailored to the needs of mobile users.

### 6. **Offline Support and Synchronization**

- **Reason:** Local AppDatabase ensures uninterrupted functionality, storing messages locally when the user is offline.
- **Benefits:** Enhances user experience by synchronizing messages with the RemoteDatabase once connectivity is restored.

This comprehensive design ensures that GEMBOS meets its functional and non-functional requirements while providing a robust, secure, and user-friendly messaging platform. The following sections elaborate on the specific components and their interactions, supported by detailed UML diagrams to illustrate the system architecture.

### 3.2. GEMBOS Software System Structure

The GEMBOS application is structured into interconnected components, ensuring modularity, scalability, and a secure communication platform. The system architecture is divided into distinct layers: frontend, backend, and database. Each layer is carefully designed to perform its specific responsibilities efficiently. This section provides an in-depth explanation of the system's structure, accompanied by the UI design illustrations that reflect the user interaction workflows.

#### Key Components and Their Roles:

##### Frontend Package

**Responsibilities:** The frontend serves as the primary point of interaction between the user and the system. It is responsible for handling input and output operations and ensuring an intuitive and responsive user interface.

##### Features:

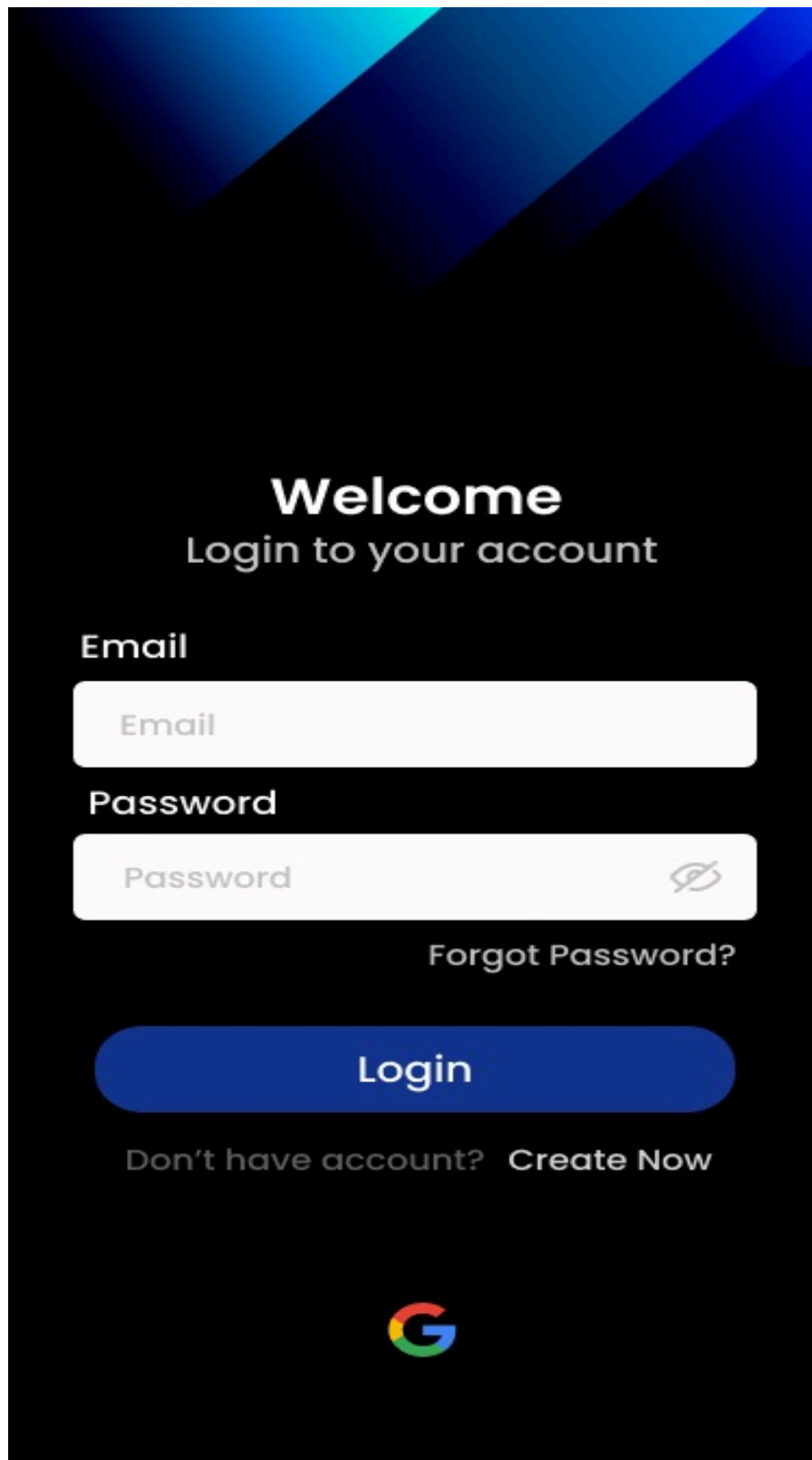
- User authentication workflows (registration, login, OTP verification).
- Interactive chat interfaces for sending encrypted messages or fallback SMS.
- Real-time updates for profile management and settings.
- Visual indicators for encryption and message delivery modes.

#### Sub-components and Examples:

##### 1. Welcome and Login Screens:

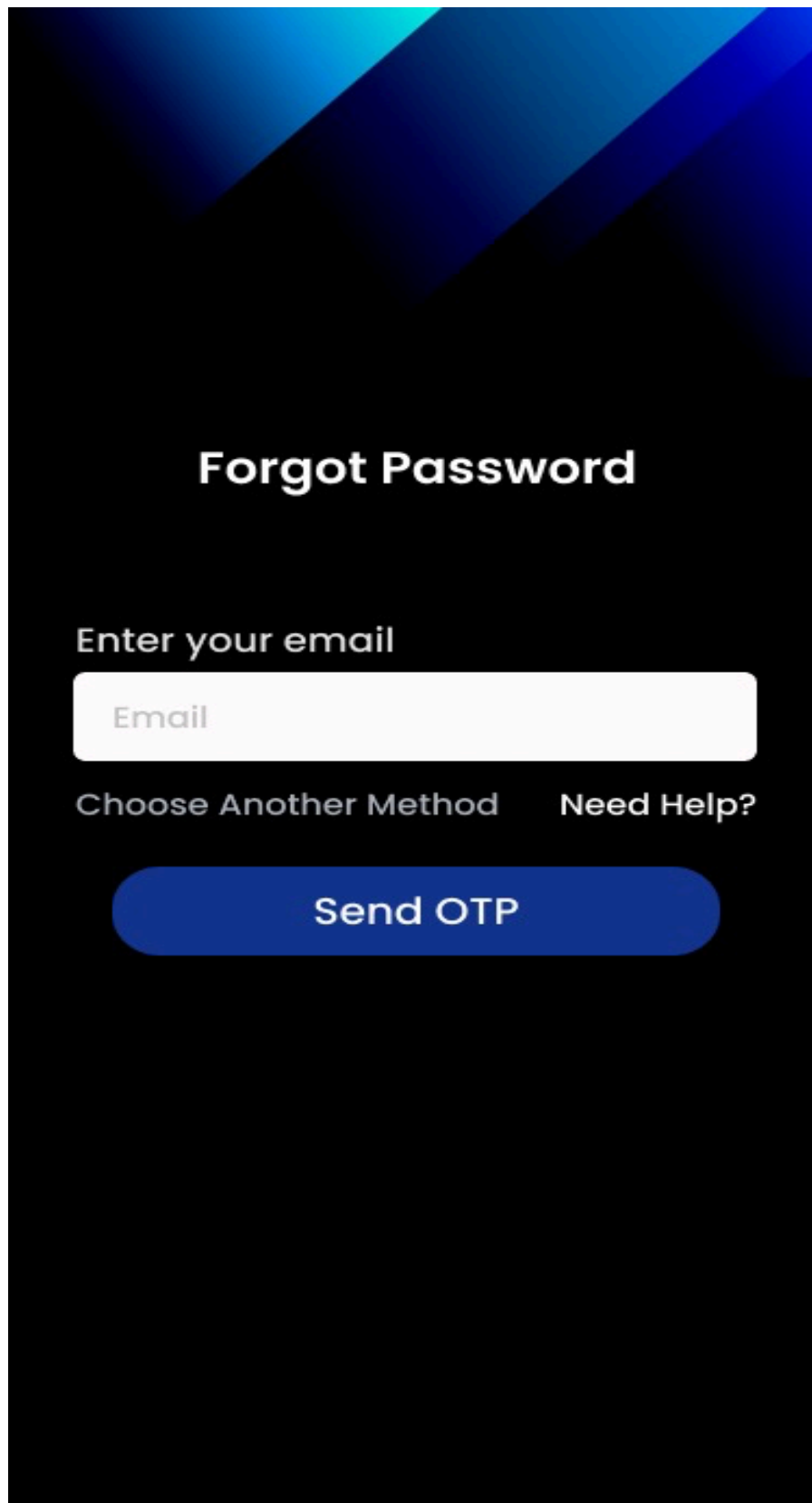
- The "Welcome" screen provides fields for entering login credentials (email and password), with an option for password recovery or account creation. This is shown in Figure 6.





*Figure 6 Login Screen*

- The "Forgot Password" and "Verification" screens guide the user through OTP verification steps for account recovery (Figures 7 and 8). These ensure secure password resets.



*Figure 7 Forgot Password Screen*

**Verification**

Messenger has send a code to  
verify your account

Email OTP

Mobile OTP

**Verify**

Resend : 00:50

*Figure 8 Verification Screen*

## **2. Main Messaging Interface:**

- The primary messaging screen (Figure 9) displays conversations with distinct formatting for encrypted and SMS-based messages. For example, messages sent over SMS are labeled with "SMS" in green, as seen in design.
- This interface also includes real-time updates for contact status and allows users to send attachments securely.



Figure 9 Messaging Interface

### 3. Inbox View:

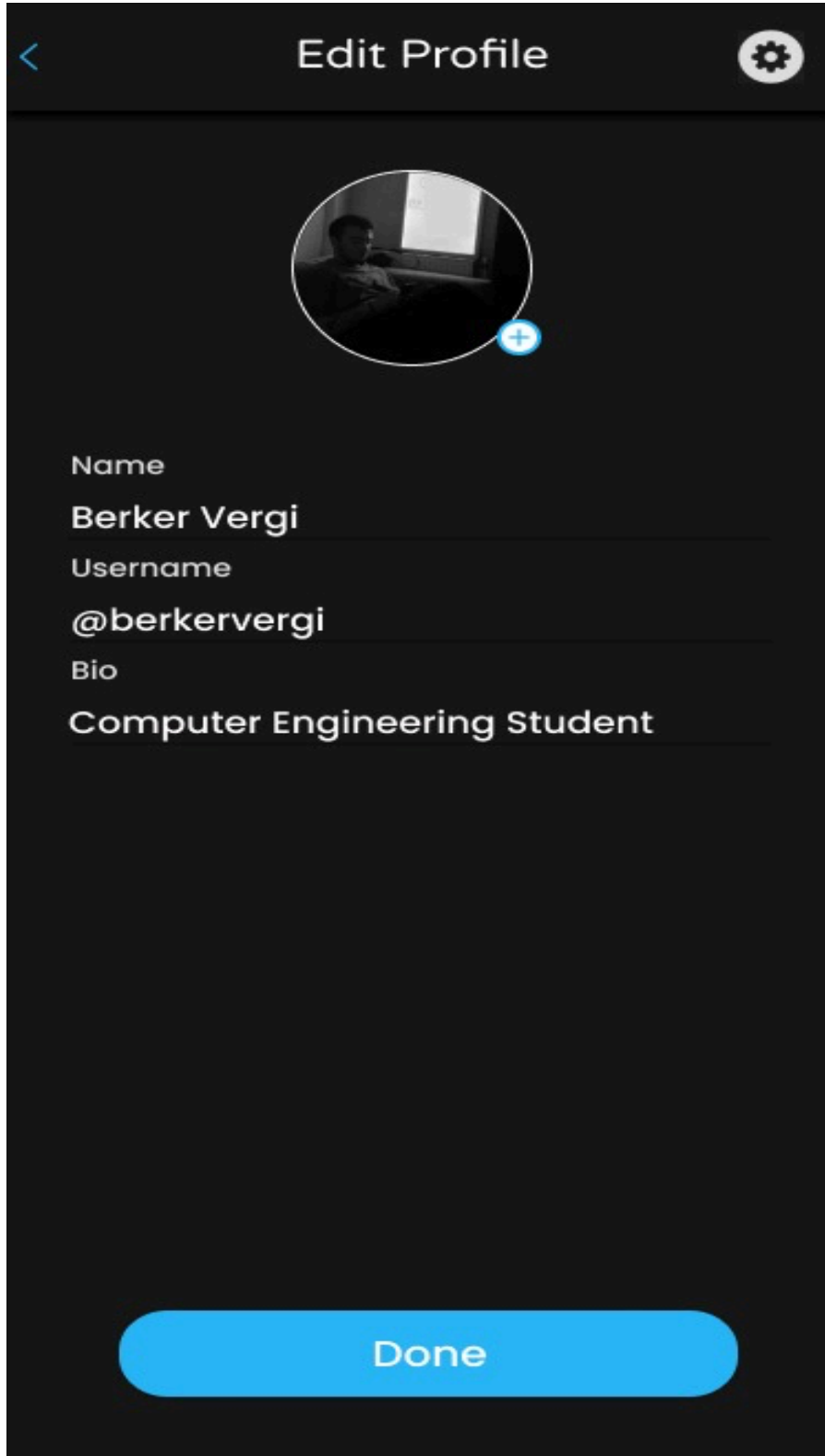
- The inbox view (Figure 10) organizes all active conversations in a compact list, with indicators for unread messages and timestamps for activity.




Figure 10 Inbox Screen

#### 4. Profile and Settings Management:

- The "Edit Profile" screen allows users to update personal details like username, bio, and profile picture (Figure 11).
- The "Settings" screen offers account management options, privacy settings, and logout functionality (Figure 12).



**Edit Profile**



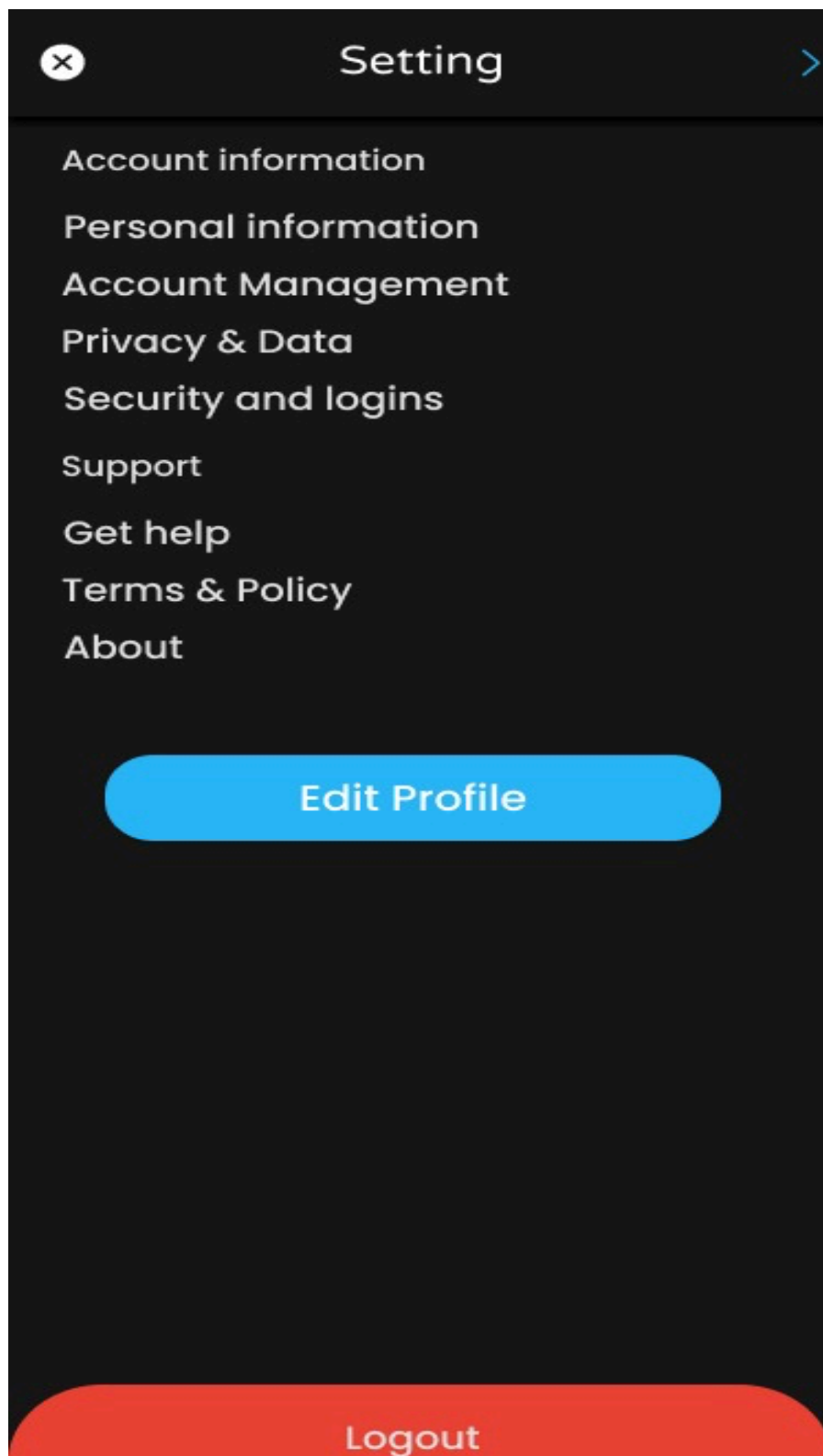
Name  
**Berker Vergi**

Username  
**@berkervergi**

Bio  
**Computer Engineering Student**

**Done**

*Figure 11 Edit Profile*



*Figure 12 Settings Screen*

### **Backend Package**

#### **Responsibilities:**

The backend is the core of the system, managing all business logic and interactions between the



frontend and database layers. It ensures secure encryption for messages and facilitates communication with external APIs for key validation and message synchronization.

**Features:**

- Implements Elliptic Curve Cryptography (ECC) for secure message encryption.
- Processes user authentication and session management.
- Handles the transmission and synchronization of messages, even in offline scenarios.

**Sub-components:**

1. **Message Service:**

- Encrypts outgoing messages and decrypts incoming ones.
- Fallback mechanism to send messages via SMS when the internet is unavailable.

2. **User Service:**

- Handles user authentication, including login, logout, and registration workflows.
- Manages user profile updates and settings modifications.

3. **Notification Service:**

- Sends real-time notifications to the frontend for events such as new messages or updates.

**Illustrative UI Integration:**

- The backend operations directly support UI elements such as the real-time notifications shown in **Figure 9 (Messaging Interface)**.

**Database Package**

**Responsibilities:**

The database layer is responsible for secure and persistent data storage. It manages all structured data, including user profiles, messages, and system logs.

**Features:**

- Ensures data security through encrypted storage of sensitive information like passwords and messages.
- Supports offline access with local caching of messages.

**Sub-components:**

1. **Remote Database:**

- Centralized storage for all user and message data, ensuring consistency across devices.

2. **App Database:**

- Temporarily stores offline messages for synchronization when the internet is restored.

**Illustrative UI Integration:**

- Offline message storage is reflected in **Figure 9 (Messaging Interface)**, where messages sent offline are labeled and synced later.

### 3.3. GEMBOS Software System Environment

The GEMBOS software subsystem is engineered to function in a robust and distributed environment, leveraging modern hardware, advanced system software, and middleware technologies to ensure optimal scalability, reliability, and security. This section provides a comprehensive overview of the target software environment, development tools, and supporting middleware.

#### 3.3.1 System Software Environment

The software environment of the GEMBOS system comprises the backend, frontend, database, and middleware technologies that collectively support its distributed architecture and secure messaging features.

##### 1. Backend

- **Programming Language:**
  - Java (primary language for Android-compatible backend functionality).
  - Swift (primary language for iOS-compatible backend functionality).
- **Frameworks:**
  - **Spring Boot:** A lightweight, scalable, and secure framework for RESTful API development.
- **APIs:**
  - **Elliptic Curve Cryptography API:** Ensures secure key exchange for encrypted communications.
  - **AES Encryption Algorithms:** Used for encrypting and decrypting sensitive data.
- **Security Layers:**
  - Key management using Android Keystore and iOS Keychain.
  - Secure Transmission via SSL/TLS for all API communications.
- **Message Processing:**
  - Utilizes distributed microservices for message storage, key validation, and data encryption.

##### 2. Frontend

- **Programming Languages:**
  - JavaScript/TypeScript for frontend scripting.
  - Java for Android app development.
  - Swift for iOS app development.
- **Frameworks:**
  - **Android and iOS SDK'S:** For creating native mobile apps user interfaces with real-time responsiveness.
  - **SpringBoot:** For server-side rendering and optimizing performance.
- **Styling:**
  - **Tailwind CSS:** For modern, responsive, and dynamic UI design.
  - **Bootstrap 5:** To ensure consistent design across platforms.
- **UI Features:**
  - Includes dynamic components for login, messaging, contact sync, and profile editing (refer to **Figures 6–12** for detailed UI representations).
  - Integrated with the backend via secure API calls to fetch data and submit user inputs.

##### 3. Database

- **Database Management System:**
  - MySQL: Selected for its performance, scalability, and support for complex queries.
- **Key Features:**
  - Support for relational and non-relational data to handle structured and unstructured data (e.g., user profiles, messages, encryption keys).
  - Advanced indexing for rapid data retrieval.
- **Replication:**
  - Uses master-slave replication to ensure high availability in a distributed environment.
- **Security Features:**
  - Data encryption at rest using Elliptic Curve.
  - Regular backups and recovery processes for data integrity.

### 3.3.2 Development Tools

The GEMBOS development process employs a range of modern tools and platforms to facilitate efficient development, debugging, testing, and deployment.

1. **Integrated Development Environment (IDE):**
  - **IntelliJ IDEA:** For Java-based backend development.
  - **Xcode:** For Swift development in iOS.
  - **Android Studio:** For Android app development.
  - **Visual Studio Code:** For frontend scripting and API integrations.
2. **Version Control:**
  - **Git:** For source code management.
  - **GitHub:** Central repository for collaborative development and issue tracking.
3. **Testing Tools:**
  - **Postman:** For testing RESTful API endpoints.
  - **Selenium:** For automating user interface tests.
4. **CI/CD Pipeline:**
  - **GitHub Actions:** For continuous integration and automated testing before deployment.
  - **Docker:** Containerization for deployment in distributed environments.

### 3.3.3 Middleware and Supporting Software

To ensure smooth interaction between the frontend, backend, and database, the GEMBOS system employs advanced middleware for secure, efficient, and scalable communication.

1. **Authentication:**
  - **JSON Web Tokens (JWT):** For user authentication and session management.
2. **Data Validation:**
  - **express-validator:** Ensures that all user inputs (e.g., messages, profile updates) meet specified validation rules.
3. **Data Parsing:**
  - **express.json():** For parsing incoming JSON requests.
  - **express.urlencoded():** For parsing form submissions.
4. **Error Handling:**
  - Custom middleware in Spring Boot to handle errors and provide user-friendly feedback.
5. **Caching:**
  - **Redis:** For storing frequently accessed data to reduce database load and improve response time.
6. **Message Queue:**
  - **RabbitMQ:** For asynchronous communication between distributed components, ensuring reliable message delivery.
7. **Distributed File Storage:**
  - **Amazon S3:** For storing and retrieving media files (e.g., profile pictures).

### 3.3.4 Justification for Environment Choices

The selected tools and frameworks for GEMBOS are based on their compatibility with distributed systems, performance, and scalability requirements:

1. **Modern Frameworks:**
  - Android, iOS and Spring Boot were chosen for their ability to handle application development and scalable backend operations.
2. **Secure Middleware:**
  - Tools like JWT and Redis enhance system security and performance by ensuring secure authentication and efficient caching.
3. **Database Performance:**
  - MySQL advanced indexing and master-slave replication ensure fast, reliable, and scalable data storage.

## 4. GEMBOS Software System Detailed Design:

Detailed design will be carried out in the context of the COMP 4920 course next semester.

## 5. Testing Design

The testing design for the **GEMBOS system** is structured to ensure the application is reliable, secure, and user-friendly. A combination of **unit testing, integration testing, and user acceptance testing (UAT)** will be utilized to validate the system's functionality, performance, and usability.

### 5.1 Unit Testing

Unit testing will be conducted to validate the individual components of the system in isolation. Each class, method, and function will be tested to ensure they behave as expected under various scenarios.

#### Scope:

- Validation of encryption and decryption methods in the **EncryptionManager**.
- Testing secure key storage and retrieval mechanisms in the **Keystore**.
- Ensuring proper functioning of the **APIClient** methods, such as `fetchKey()` and `sendMessage()`.
- Verifying the logic in **SmsManager** for sending and receiving SMS.
- Testing backend services, including the user registration and login workflows.

#### Tools:

- **JUnit** for backend components.
- **Karma** for frontend unit testing.

### 5.2 Integration Testing

Integration testing will verify seamless interactions between different modules, ensuring that the system works cohesively.

#### Scope:

- Validation of interactions between the **MainScreen**, **ChatScreen**, and **APIClient**.
- Testing database operations, including storing and retrieving messages from **RemoteDatabase**.
- Ensuring correct API calls from the frontend to the backend for user authentication and message synchronization.
- Checking the communication flow between the **NotificationManager** and the frontend UI to display real-time alerts.

#### Tools:

- **Postman** for API endpoint testing.
- **Spring Test** for integration testing of backend services.

### 5.3 User Acceptance Testing (UAT)

UAT will be conducted to confirm the system meets the expectations and requirements of end-users. Test scenarios will simulate real-world use cases to ensure functionality and usability.

#### Scope:

- Verifying the usability of the **Inbox**, **Chat**, and **Profile Management** features (Figures 2, 5, and 6).
- Testing the registration and login process (Figures 9 and 10) for a smooth user experience.
- Ensuring the encryption and messaging workflows are seamless and secure.
- Evaluating the responsiveness of the UI across various devices.

#### Methods:

- Engaging beta testers to provide feedback on the application's functionality.
- Simulating network disruptions to validate offline messaging and synchronization workflows.
- Testing accessibility features such as color contrast and font sizes for readability.

### 5.4 Justification for Testing Approach

The combination of **unit testing, integration testing, and UAT** ensures a comprehensive validation of the GEMBOS system:

- **Unit testing** ensures that individual components are robust and error-free.
- **Integration testing** guarantees that modules interact seamlessly, supporting the distributed system architecture.
- **User acceptance testing** validates that the application meets user needs and provides an intuitive, secure experience.

By employing these methods, the GEMBOS system is ensured to meet its goals of **reliability, security, and user satisfaction**.

## References

1. GEMBOS Requirements Specification Document (RSD).

## **APPENDIX C: PROJECT MANAGEMENT DOCUMENTS**

### **APPENDIX C1: PROJECT PLAN**

Tablo 1

COMP 4910 GEMBOS: GEMBOS, Project Plan, 22.01.2025, v1.0																	
Task No	Task Name	Weeks															Any Additional Notes
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	Project Planning & Requirements Gathering	X		X						X							
2	Literature Review					X		X							X		
3	General Mobile and Repel Company Meeting						X	X									
4	Search Key Exchange Algorithms	X							X					X			
5	Search Key Generate Algorithms		X					X		X		X			X		
6	UI/UX Design			X							X			X			
7	Database Structure & System Design Planning				X	X				X				X			
8	Programing SMS Protocols for Application		X					X		X	X				X		
9	Presentatio n Preparation		X														
Form: GEMBOS-Project-Plan-2025-01-22-v1.0.xlsx																	

## **APPENDIX C2: PROJECT EFFORT LOG- CONSOLIDATED**



## COMP 4910 Project Effort Log, Project Code: GEMBOS, 19.01.2025, v1.0

Week	Dates	Giray Aksakal	
		Work Done	Total Hours Spent
Week 1	29.09.2024 - 05.10.2024	Defining project, objectives and team forming.	3
Week 2	06.10.2024 - 12.10.2024	Search for project advisor.	0
Week 3	13.10.2024 - 20.10.2024	Search for project advisor.	0
Week 4	21.10.2024 - 27.10.2024	Prepare PAF and research for building Android Common Kernel	8
Week 5	28.10.2024 - 03.11.2024	Research for building Generic Kernel Image (GKI).	9
Week 6	04.11.2024 - 10.11.2024	Prepared for presentation and meeting at General Mobile.	10
Week 7	11.11.2024 - 17.11.2024	Technical meetings conducted with GM and Repel.	36
Week 8	18.11.2024 - 24.11.2024	Design and develop API controllers.	6
Week 9	25.11.2024 - 01.12.2024	Research for new cryptograph algorithms as advised.	20
Week 10	02.12.2024 - 08.12.2024	Designed Login and Register screens' UI design.	6
Week 11	09.12.2024 - 15.12.2024	Develop new login and register screens.	6
Week 12	16.12.2024 - 22.12.2024	Integrated API and improved UI design.	8
Week 13	23.12.2024 - 29.12.2024	Implemented encryption functionalities for mobile app.	10
Week 14	30.12.2024 - 05.01.2025	Defined testing designs and class structures.	6
Week 15	06.12.2025 - 12.01.2025	Improvements to DSD and RSD files have done.	9
Week 17	13.01.2025 - 19.01.2025	Encryption tests have been conducted.	8
Week 18	20.01.2025 - 22.01.2025	Improvements to DSD and RSD files have done.	8
Total Effort in Man-Hours			150,00
Total Effort in Man-Days			18,75
Notes:			
1. This table shows the consolidated project effort, based on project effort tables prepared by each team member			
2. Replace Team Member i with team member name and lastname, then fill out one column for each team member			

Form: GEMBOS-Project Effort Log-2025-01-19-v1.0.xlsx

Berker Vergi		Mert Kahraman	
Work Done	Total Hours Spent	Work Done	Total Hours Spent
Defining project, objectives and team forming.	3	Defining project, objectives and team forming.	3
Search for project advisor.	0	Search for project advisor.	5
Search for project advisor.	0	Search for project advisor.	5
Search for "Android Kernel Overview".	8	Project Proposal: Introduction and Aim & Objectives parts	7
Search for "Android Security Implementation".	8	Search for "Building custom AOSP versions"	7
Prepared for presentation and meeting at General Mobile.	10	Prepared for presentation and meeting at General Mobile.	10
Technical meetings conducted with GM and Repel.	36	Technical meetings conducted with GM and Repel.	36
Search for SMS protocol, OTP messages and Android Key Store	10	Revised project proposal regarding the mentor feedback	6
Implement texting functionality with SMS protocol to mobile ap	20	Complete Introduction and Purpose of the Project headlines of SOW file	8
Designed Dashboard and Texting Screens' UI.	8	Complete Use-case Point Complexity, Applicable Standards parts of SOW fi	8
Develop designed screens for mobile application.	9	Organized GitHub repositories and workflow on Jira	6
Design encryption functions for mobile app.	12	API endpoints and database logging	8
Implement encryption functions to mobile application.	10	Planning and redesigning API structure	10
Define software system designs for DSD file.	10	Defined Use-Case and UML diagrams for RSD file	5
Improvements done to RSD and DSD files.	9	Continued working on Use-Case and UML diagrams for RSD file	5
Application and SMS tests are done.	6	Conducted API tests and resolved unexpected behaviors.	5
Improvements to DSD and RSD files have done.	8	Improvements to DSD and RSD files have done.	1
	164,00		122,00
	20,50		15,25

Emir Morah		Total Weekly Effort in Man-Hours
Work Done	Total Hours Spent	
Defining project, objectives and team forming.	3	12,00
Search for project advisor.	5	10,00
Search for project advisor.	5	10,00
Project Proposal: User Personas , Expected Outcomes and Happy Paths	8	31,00
Search for "Cross device SDK" of Android OSP	7	31,00
Prepared for presentation and meeting at General Mobile.	10	40,00
Technical meetings conducted with GM and Repel.	36	144,00
Revised project proposal regarding the mentor feedback	6	28,00
Complete User Personas, Happy Path and Scope of Work headlines of SOW	8	56,00
Completed rest of SOW file: Low Level Functionalities, Special Requirements	8	30,00
Added message controlling endpoints to API.	4	25,00
Designed models and database for API	7	35,00
Added authentication endpoints to API	8	38,00
Started use case and UML diagrams for RSD file	5	26,00
Completed use case and UML diagrams for RSD file	5	28,00
Conducted API tests and resolved unexpected behaviors.	4	23,00
Improvements to DSD and RSD files have done.	1	18,00
	117,00	553,00
	14,63	69,13

## **APPENDIX C3: PROJECT EFFORT LOGS FOR EACH TEAM MEMBER-**

**COMP 4910 Project Effort Log, Project Code: GEMBOS, 19.01.2025, v1.0****Team Member: Giray Aksakal**

Week	Dates	Work Done in Some Detail	Total Hours Spent
Week 1	29.09.2024 - 05.10.2024	Defining project, objectives and team forming.	3
Week 2	06.10.2024 - 12.10.2024	Search for project advisor.	0
Week 3	13.10.2024 - 20.10.2024	Search for project advisor.	0
Week 4	21.10.2024 - 27.10.2024	Prepare PAF and research for building Android Common Kernel.	8
Week 5	28.10.2024 - 03.11.2024	Research for building Generic Kernel Image (GKI).	9
Week 6	04.11.2024 - 10.11.2024	Prepared for presentation and meeting at General Mobile.	10
Week 7	11.11.2024 - 17.11.2024	Technical meetings conducted with GM and Repel.	36
Week 8	18.11.2024 - 24.11.2024	Design and develop API controllers.	6
Week 9	25.11.2024 - 01.12.2024	Research for new cryptograph algorithms as advised.	20
Week 10	02.12.2024 - 08.12.2024	Designed Login and Register screens' UI design.	6
Week 11	09.12.2024 - 15.12.2024	Develop new login and register screens.	6
Week 12	16.12.2024 - 22.12.2024	Integrated API and improved UI design.	8
Week 13	23.12.2024 - 29.12.2024	Implemented encryption functionalities for mobile app.	10
Week 14	30.12.2024 - 05.01.2025	Defined testing designs and class structures.	6
Week 15	06.12.2025 - 12.01.2025	Improvements to DSD and RSD files have done.	9
Week 17	13.01.2025 - 19.01.2025	Encryption tests have been conducted.	8
Week 18	20.01.2025 - 22.01.2025	Improvements to DSD and RSD files have done.	8
Total Effort in Man-Hours			153,00
Total Effort in Man-Days			19,13

**Notes:**

1. This table shows the team member project effort. One Man-Day is Eight Man-Hours.
2. Each team member must fill out the form periodically (preferably at the end of the week of any work done).
3. Each filled-out table must be emailed at the end of each month to a selected Project Member (cc to Project Advisor), who will produce a consolidated table.

Form: GEMBOS-Project Effort Log-2025-01-19-v1.0-Aksakal.xlsx

**COMP 4910 Project Effort Log, Project Code: GEMBOS, 19.01.2025, v1.0****Team Member: Berker Vergi**

Week	Dates	Work Done in Some Detail	Total Hours Spent
Week 1	29.09.2024 - 05.10.2024	Defining project, objectives and team forming.	3
Week 2	06.10.2024 - 12.10.2024	Search for project advisor.	0
Week 3	13.10.2024 - 20.10.2024	Search for project advisor.	0
Week 4	21.10.2024 - 27.10.2024	Search for "Android Kernel Overview".	8
Week 5	28.10.2024 - 03.11.2024	Search for "Android Security Implementation".	8
Week 6	04.11.2024 - 10.11.2024	Prepared for presentation and meeting at General Mobile.	10
Week 7	11.11.2024 - 17.11.2024	Technical meetings conducted with GM and Repel.	36
Week 8	18.11.2024 - 24.11.2024	Search for SMS protocol, OTP messages and Android Key Store.	10
Week 9	25.11.2024 - 01.12.2024	Implement texting functionality with SMS protocol to mobile app.	20
Week 10	02.12.2024 - 08.12.2024	Designed Dashboard and Texting Screens' UI.	8
Week 11	09.12.2024 - 15.12.2024	Develop designed screens for mobile application.	9
Week 12	16.12.2024 - 22.12.2024	Design encryption functions for mobile app.	12
Week 13	23.12.2024 - 29.12.2024	Implement encryption functions to mobile application.	10
Week 14	30.12.2024 - 05.01.2025	Define software system designs for DSD file.	10
Week 15	06.12.2025 - 12.01.2025	Improvements done to RSD and DSD files.	9
Week 17	13.01.2025 - 19.01.2025	Application and SMS tests are done.	6
Week 18	20.01.2025 - 22.01.2025	Improvements to DSD and RSD files have done.	8
Total Effort in Man-Hours			167,00
Total Effort in Man-Days			20,88

**Notes:**

1. This table shows the team member project effort. One Man-Day is Eight Man-Hours.
2. Each team member must fill out the form periodically (preferably at the end of the week of any work done).
3. Each filled-out table must be emailed at the end of each month to a selected Project Member (cc to Project Advisor), who will produce a consolidated table.

**COMP 4910 Project Effort Log, Project Code: GEMBOS, 19.01.2025, v1.0****Team Member: Mert Kahraman**

Week	Dates	Work Done in Some Detail	Total Hours Spent
Week 1	29.09.2024 - 05.10.2024	Defining project, objectives and team forming.	3
Week 2	06.10.2024 - 12.10.2024	Search for project advisor.	5
Week 3	13.10.2024 - 20.10.2024	Search for project advisor.	5
Week 4	21.10.2024 - 27.10.2024	Project Proposal: Introduction and Aim & Objectives parts	7
Week 5	28.10.2024 - 03.11.2024	Search for "Building custom AOSP versions"	7
Week 6	04.11.2024 - 10.11.2024	Prepared for presentation and meeting at General Mobile.	10
Week 7	11.11.2024 - 17.11.2024	Technical meetings conducted with GM and Repel.	36
Week 8	18.11.2024 - 24.11.2024	Revised project proposal regarding the mentor feedback	6
Week 9	25.11.2024 - 01.12.2024	Complete Introduction and Purpose of the Project headlines of SOW file	8
Week 10	02.12.2024 - 08.12.2024	Complete Use-case Point Complexity, Applicable Standards parts of SOW file	8
Week 11	09.12.2024 - 15.12.2024	Organized GitHub repositories and workflow on Jira	6
Week 12	16.12.2024 - 22.12.2024	API endpoints and database logging	8
Week 13	23.12.2024 - 29.12.2024	Planning and redesigning API structure	10
Week 14	30.12.2024 - 05.01.2025	Defined Use-Case and UML diagrams for RSD file	5
Week 15	06.12.2025 - 12.01.2025	Continued working on Use-Case and UML diagrams for RSD file	5
Week 17	13.01.2025 - 19.01.2025	Conducted API tests and resolved unexpected behaviors.	5
Week 18	20.01.2025 - 22.01.2025	Improvements to DSD and RSD files have done.	1
Total Effort in Man-Hours			135,00
Total Effort in Man-Days			16,88

**Notes:**

1. This table shows the team member project effort. One Man-Day is Eight Man-Hours.
2. Each team member must fill out the form periodically (preferably at the end of the week of any work done).
3. Each filled-out table must be emailed at the end of each month to a selected Project Member (cc to Project Advisor), who will produce a consolidated table.

Form: GEMBOS-Project Effort Log-2025-01-19-v1.0-Kahraman.xlsx

**COMP 4910 Project Effort Log, Project Code: GEMBOS, 19.01.2025, v1.0****Team Member: Emir Morali**

Week	Dates	Work Done in Some Detail	Total Hours Spent
Week 1	29.09.2024 - 05.10.2024	Defining project, objectives and team forming.	3
Week 2	06.10.2024 - 12.10.2024	Search for project advisor.	5
Week 3	13.10.2024 - 20.10.2024	Search for project advisor.	5
Week 4	21.10.2024 - 27.10.2024	Project Proposal: User Personas , Expected Outcomes and Happy Paths	8
Week 5	28.10.2024 - 03.11.2024	Search for "Cross device SDK" of Android OSP	7
Week 6	04.11.2024 - 10.11.2024	Prepared for presentation and meeting at General Mobile.	10
Week 7	11.11.2024 - 17.11.2024	Technical meetings conducted with GM and Repel.	36
Week 8	18.11.2024 - 24.11.2024	Revised project proposal regarding the mentor feedback	6
Week 9	25.11.2024 - 01.12.2024	Complete User Personas, Happy Path and Scope of Work headlines of SOW file	8
Week 10	02.12.2024 - 08.12.2024	Completed rest of SOW file: Low Level Functionalities, Special Requirements	8
Week 11	09.12.2024 - 15.12.2024	Added message controlling endpoints to API.	4
Week 12	16.12.2024 - 22.12.2024	Designed models and database for API	7
Week 13	23.12.2024 - 29.12.2024	Added authentication endpoints to API	8
Week 14	30.12.2024 - 05.01.2025	Started use case and UML diagrams for RSD file	5
Week 15	06.12.2025 - 12.01.2025	Completed use case and UML diagrams for RSD file	5
Week 17	13.01.2025 - 19.01.2025	Conducted API tests and resolved unexpected behaviors.	4
Week 18	20.01.2025 - 22.01.2025	Improvements to DSD and RSD files have done.	1
Total Effort in Man-Hours			130,00
Total Effort in Man-Days			16,25

**Notes:**

1. This table shows the team member project effort. One Man-Day is Eight Man-Hours.
2. Each team member must fill out the form periodically (preferably at the end of the week of any work done).
3. Each filled-out table must be emailed at the end of each month to a selected Project Member (cc to Project Advisor), who will produce a consolidated table.