COMP4920 Senior Design Project II, Spring 2025
Advisor: Assoc. Prof. Dr. Ahmet Hasan Koltuksuz

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

Revision 2.0
20.04.2025

By:
Berker Vergi, 21070001202
Giray Aksakal, 21070001030

Revision History

Revision	Date	Explanation
1.0	20.01.2025	Initial high level design
2.0	20.04.2025	Added GEMBOS Software System Detailed Design

Table of Contents

Revision History 1
Table of Contents 2
1. Introduction3
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
4.0.1 Application Lifecycle Overview 21
4.0.1.1 Android Lifecycle Methods 21
4.0.1.2 iOS Lifecycle Methods 21
4.1 Main Module: SplashScreen 21
4.2 Subsystem S1: User Management 21
4.3 Subsystem S2: Messaging 21
4.4 Subsystem S3: Key and Encryption Handling 21
4.5 Subsystem S4: Notifications and Sync 21
5. Testing Design 21
References 23

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.

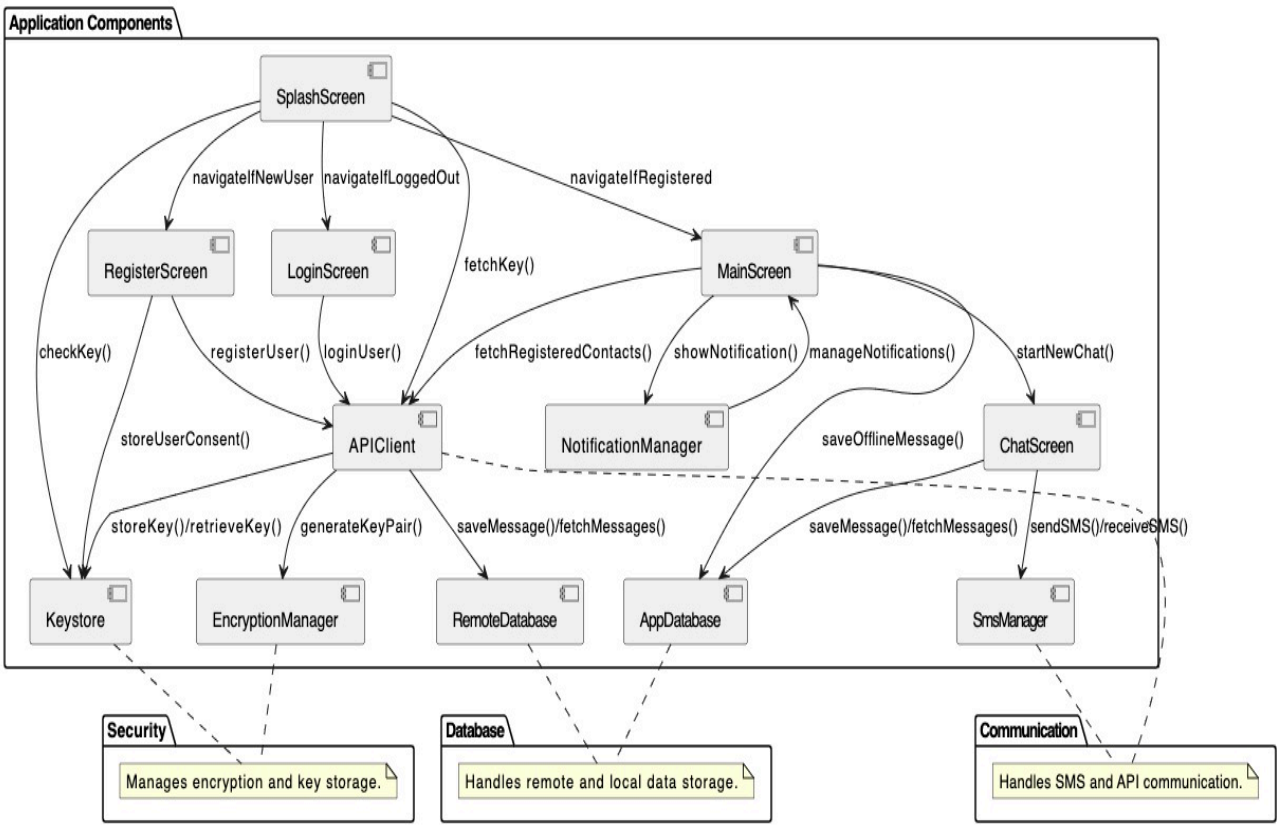


Figure 5 Component Diagram of GEMBOS Software System

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.

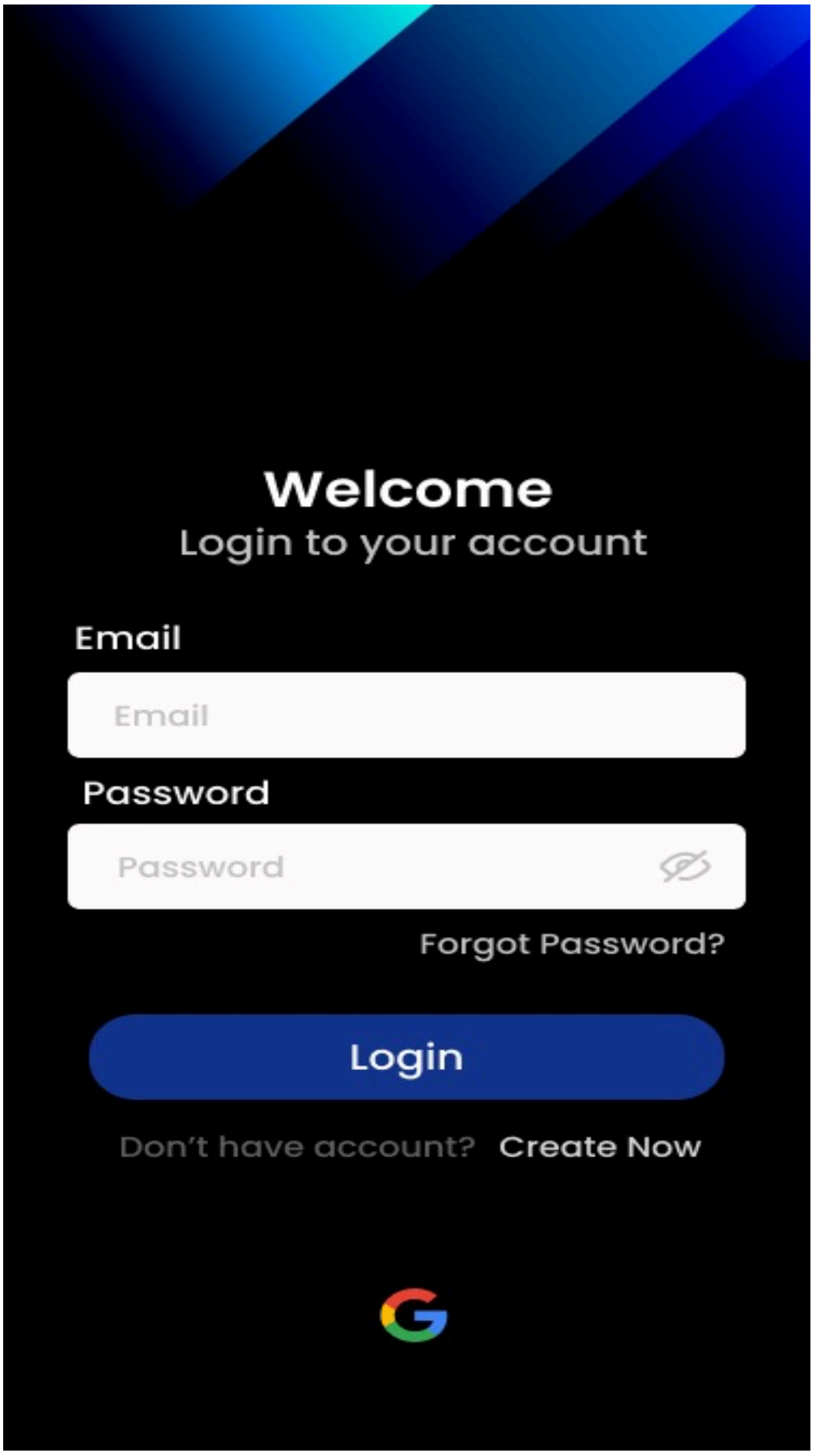


Figure 6 Login Screen

- 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.

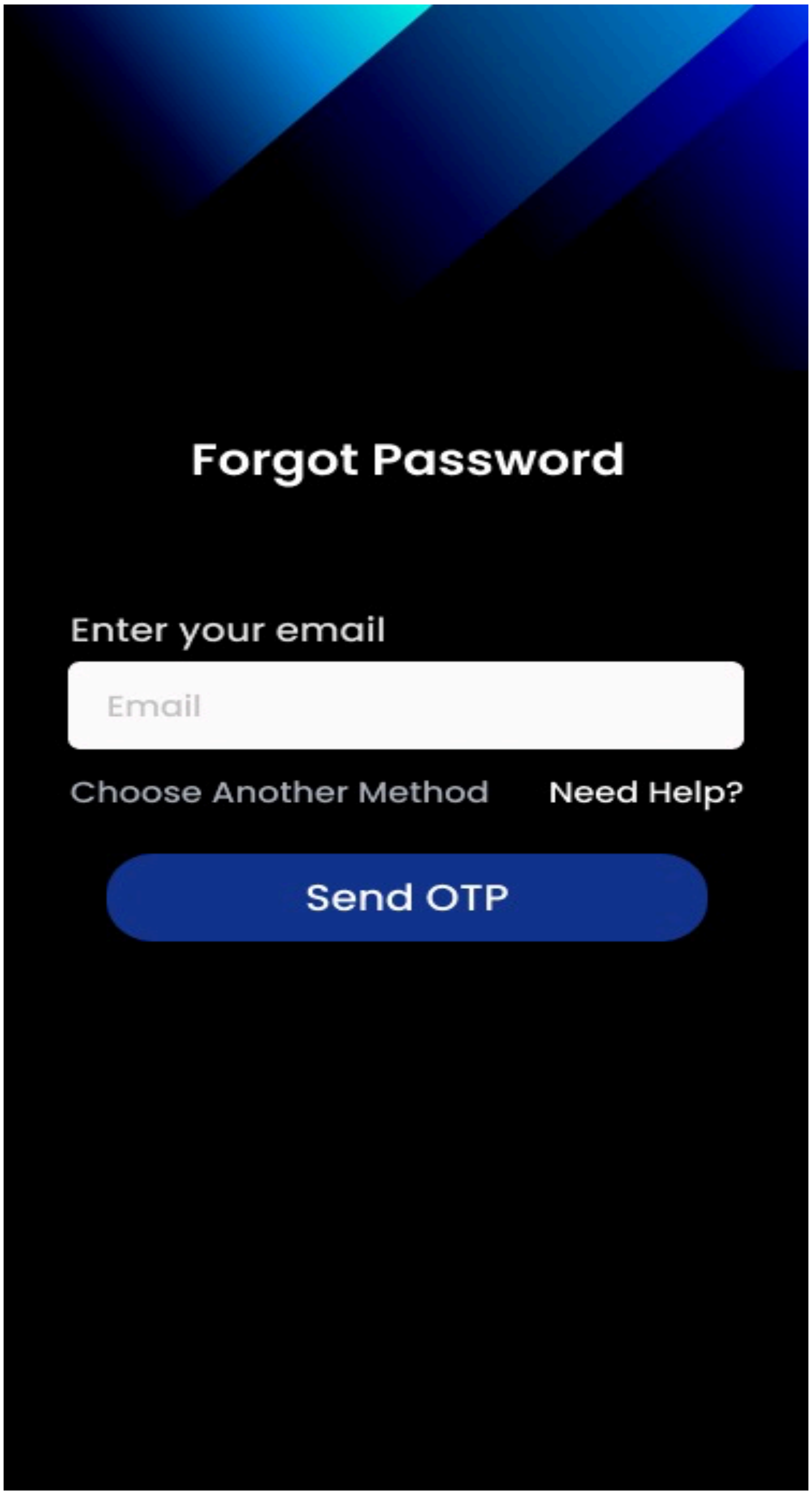


Figure 7 Forgot Password Screen

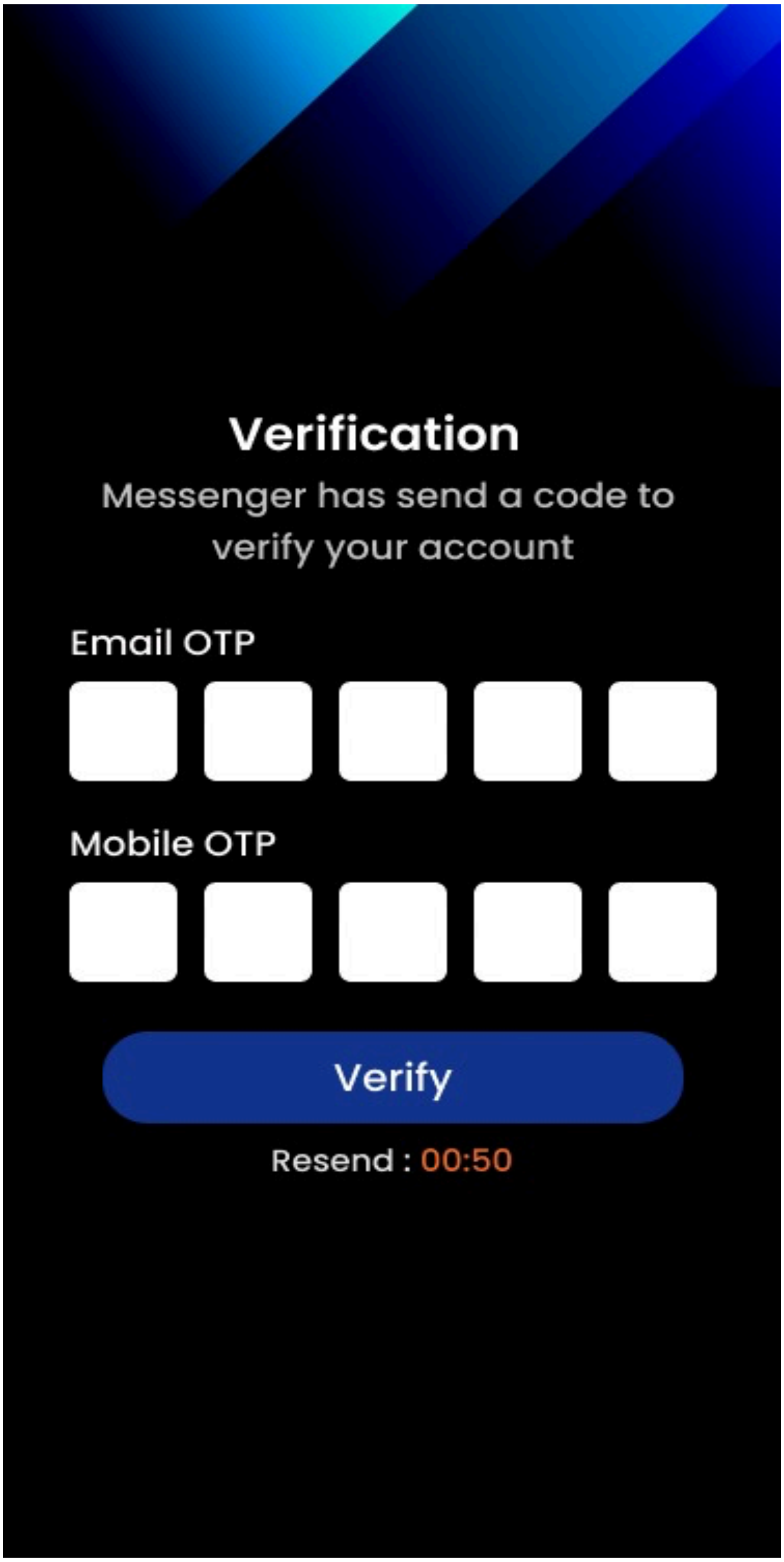
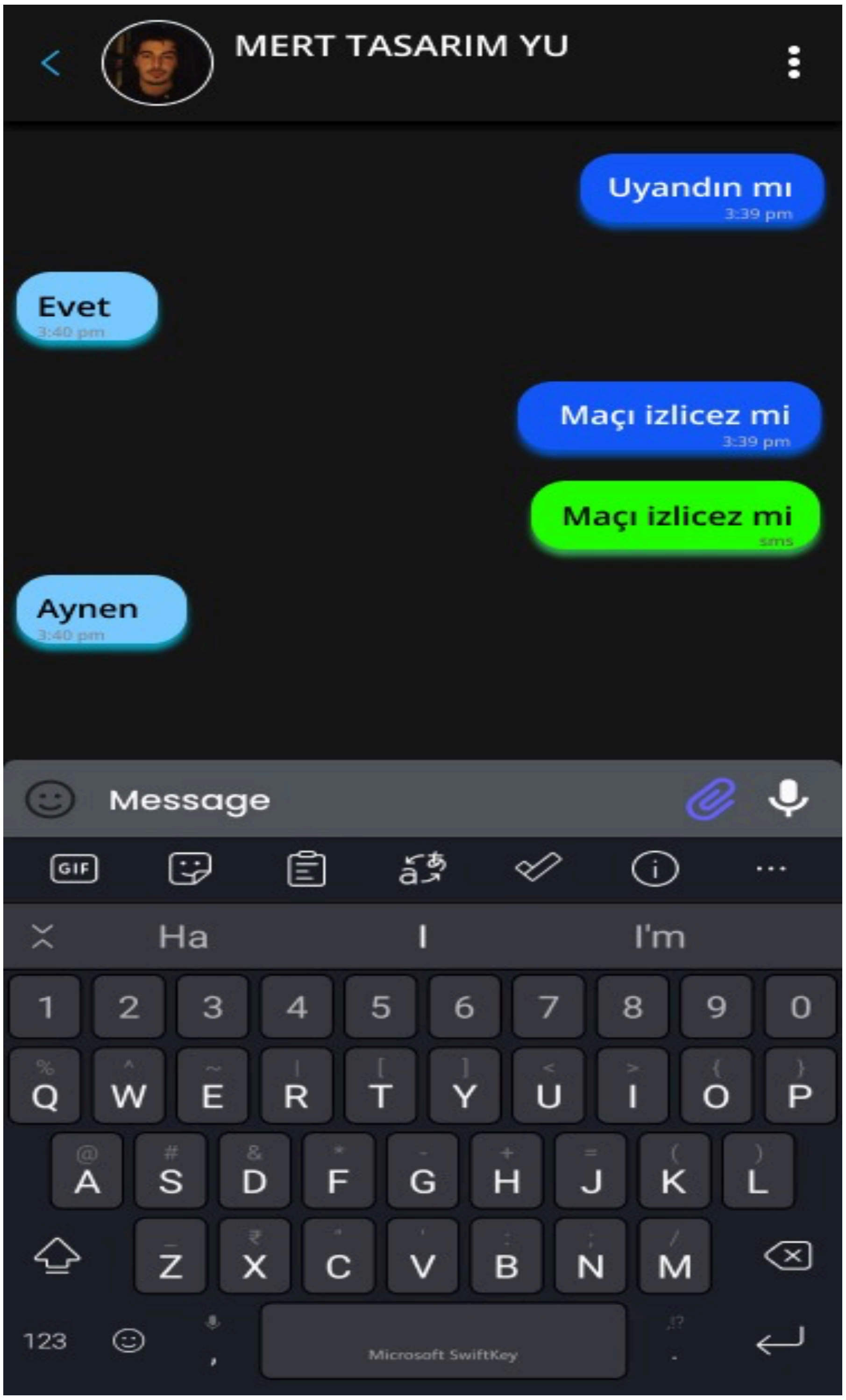


Figure 8 Verification Screen

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.


MERT TASARIM YU

Uyandın mı
3:39 pm

Evet
3:40 pm

Maçı izlicez mi
3:39 pm

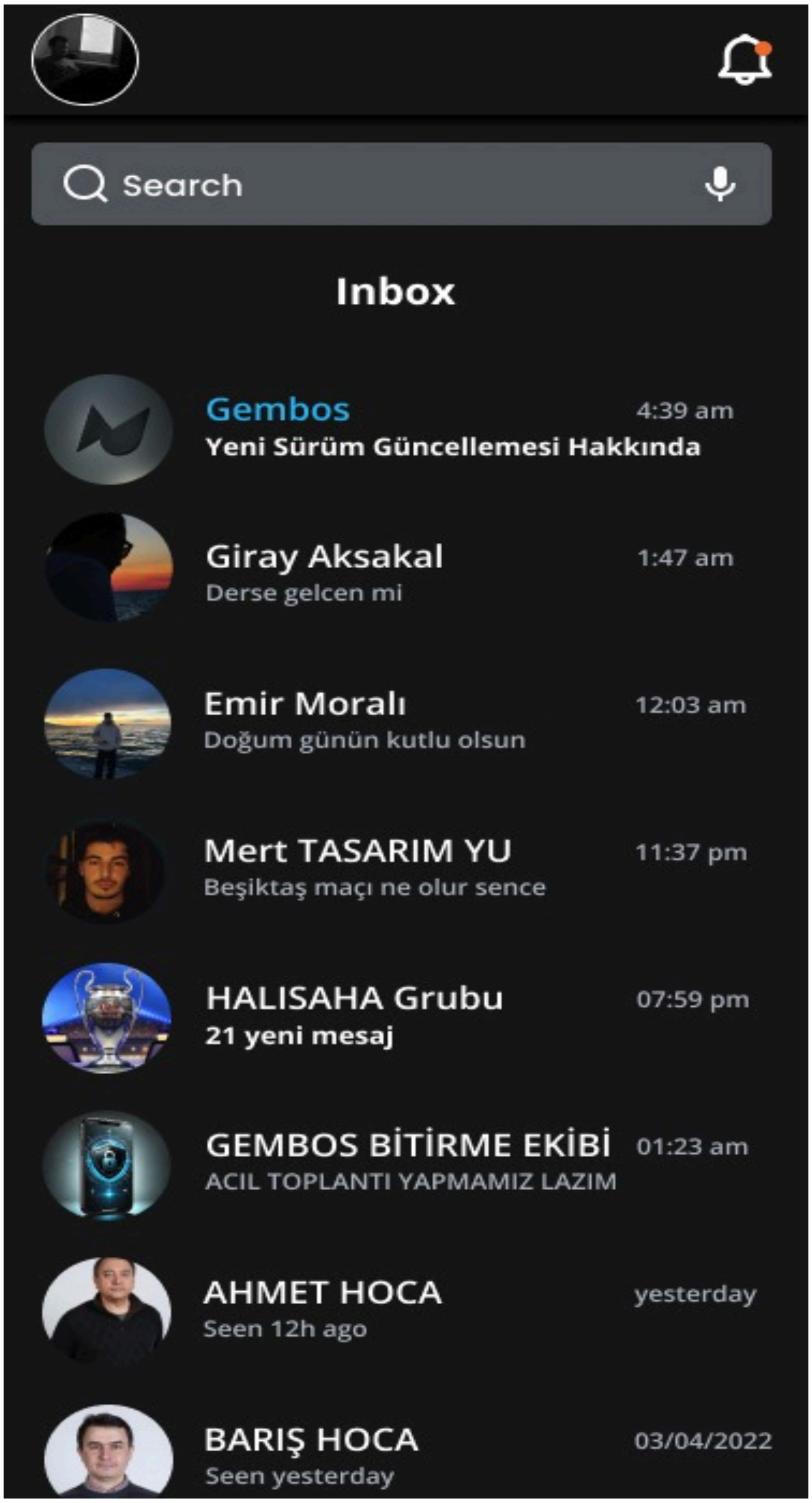
Maçı izlicez mi
sms

Aynen
3:40 pm

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.



Gembos 4:39 am
Yeni Sürüm Güncellemesi Hakkında

Giray Aksakal 1:47 am
Derse gelcen mi

Emir Moralı 12:03 am
Doğum günün kutlu olsun

Mert TASARIM YU 11:37 pm
Beşiktaş maçı ne olur sence

HALISAHA Grubu 07:59 pm
21 yeni mesaj

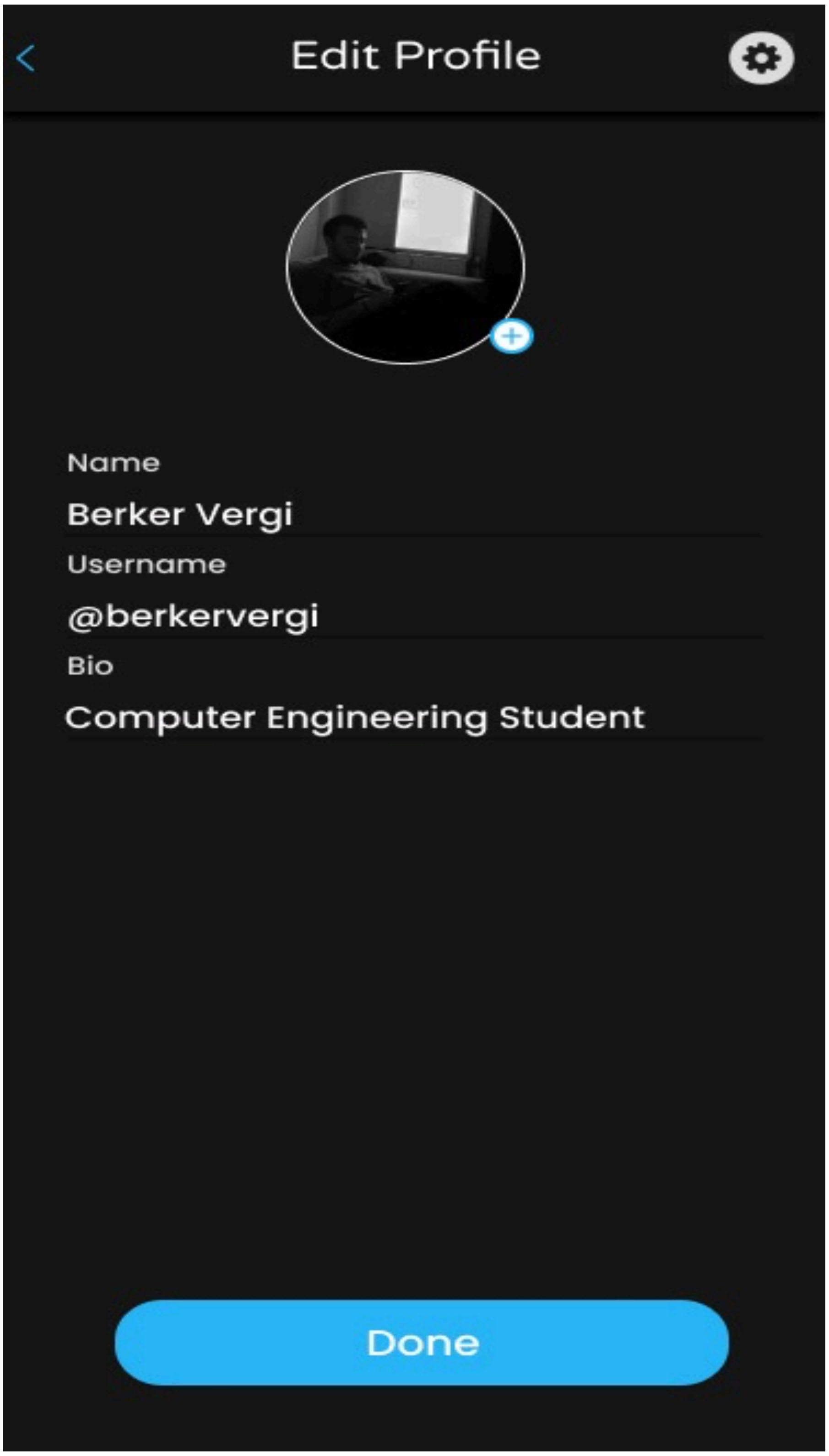
GEMBOS BİTİRME EKİBİ 01:23 am
ACIL TOPLANTI YAPMAMIZ LAZIM

AHMET HOCA yesterday
Seen 12h ago

BARIŞ HOCA 03/04/2022
Seen yesterday

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).



Name
Berker Vergi
Username
@berkervergi
Bio
Computer Engineering Student

Figure 11 Edit Profile

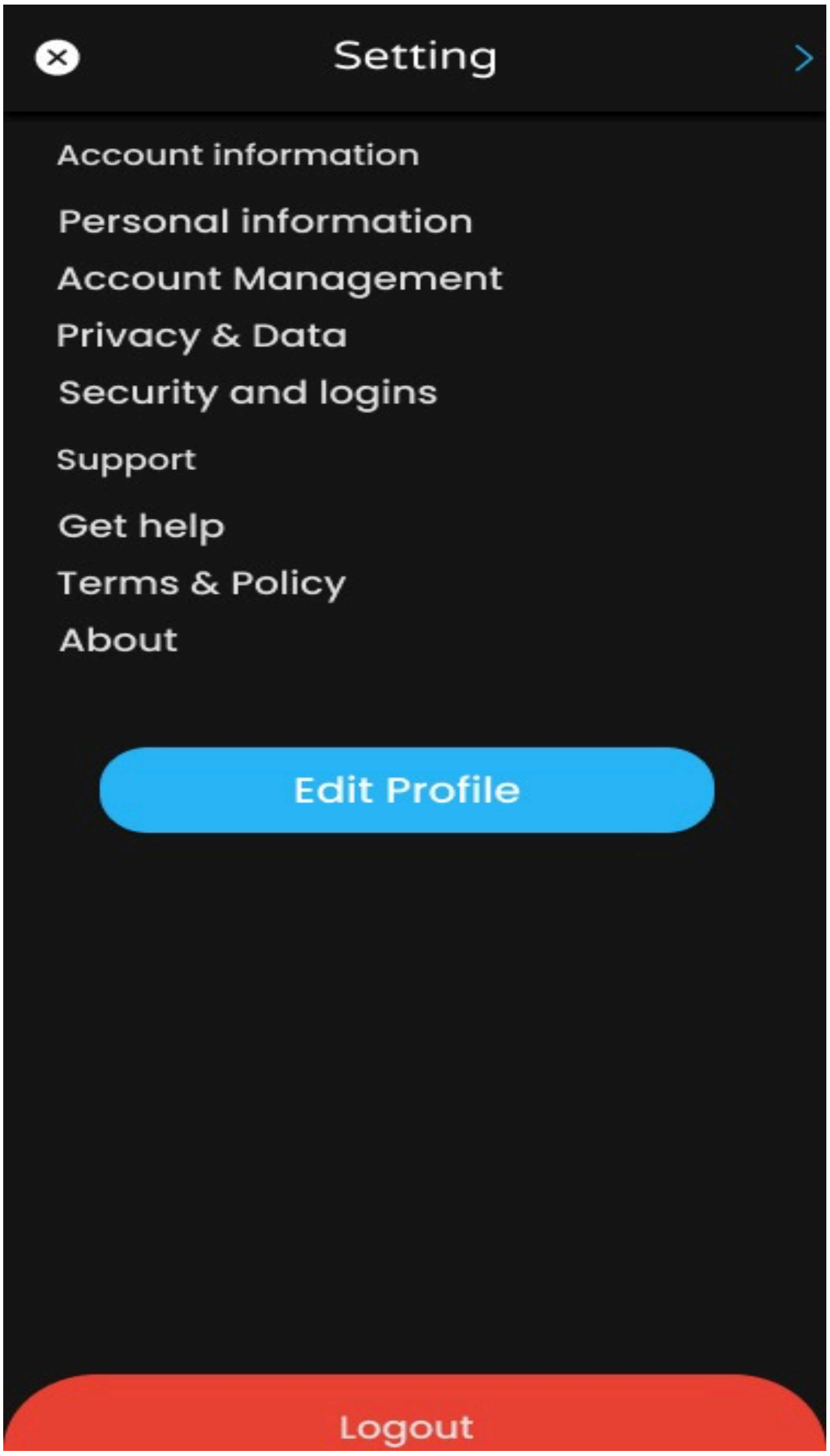


Figure 12 Settings Screen

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:

4.0.1 Application Lifecycle Overview

4.0.1.1 Android Lifecycle Methods

- **onCreate() -** Initializes API Client, Keystore, and Database.
- **onStart() -** Checks connectivity and retrieves pending updates.
- **onResume() -** Resumes sessions and updates UI.
- **onPause() -** Saves drafts and temporary data.
- **onStop() -** Releases resources.
- **onDestroy() -** Cleans up background tasks.

4.0.1.2 iOS Lifecycle Methods

- **application(_:didFinishLaunchingWithOptions:) -** Initializes core modules.
- **applicationWillEnterForeground -** Syncs and restores UI.
- **applicationDidBecomeActive -** Displays notifications, resumes secure session.
- **applicationWillResignActive -** Saves unsent data.
- **applicationDidEnterBackground -** Encrypts and syncs.
- **applicationWillTerminate -** Persists app state and clears memory.

4.1 Main Module: SplashScreen

- **fetchKey() -** Retrieves ECC key from server.
- **navigateToNextScreen() -** Determines next activity based on login status.

4.2 Subsystem S1: User Management

Modules: LoginScreen, RegisterScreen

- **registerNewUser() -** Registers users via phone/email + SSID
- **acceptKVKK() -** Logs KVKK consent
- **loginUser() -** Authenticates credentials
- **logoutUser() -** Ends session and clears token

4.3 Subsystem S2: Messaging

Modules: MainScreen, ChatScreen, MessageManager, SmsManager

- **sendMessage() -** Sends message via internet or SMS
- **receiveMessage() -** Receives incoming message
- **hashMessage() -** Hashes content for integrity
- **verifyHash() -** Validates received hash
- **sendSMS() -** Uses Android/iOS SMS APIs
- **performKeyExchangeWithDiffieHellman() -** Establishes secure channel
- **saveMessageLocally() -** Stores message if offline
- **syncOfflineMessages() -** Sends stored messages later

4.4 Subsystem S3: Key and Encryption Handling

Modules: EncryptionManager, ApplicationKeystore

- **generateEllipticCurveKeyPair () -** Creates ECC key pair
- **encryptMessageWithKey() -** Encrypts plaintext
- **decryptMessageWithKey() -** Decrypts ciphertext
- **storeKey() -** Saves key securely
- **retrieveKey() -** Loads key
- **verifyEllipticCurveKey() -** Ensures key validity

4.5 Subsystem S4: Notifications and Sync

Modules: NotificationManager

- **showNotification() -** Displays push/in-app alert
- **scheduleNotification() -** Schedules based on task or event
- **triggerSync() -** Syncs when internet connection established again

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).