Messaging App: Full System Documentation

Table of Contents

Project Goals

Objectives

The goal is to develop a secure, scalable, and feature-rich messaging platform. Key objectives include:

- Real-time messaging with end-to-end encryption.
- Multimedia sharing and voice/video calls.
- A notification system with read/delivery receipts.

Core Features

- **User Authentication**: Email-based registration and login.
- **Messaging**: Real-time one-on-one and group chats, with read receipts and message statuses.
- **Media Sharing**: Support for image, video, and document sharing.
- **Voice and Video Calls**: Secure, end-to-end encrypted calling.
- **Admin Dashboard**: Monitoring user metrics and app management.

Technology Stack

Client Side

- **Framework**: Flutter for cross-platform mobile app development.
- **Real-time Communication**: WebSockets for instant message delivery.
- **Encryption**: RSA/AES encryption to secure data during messaging.

Backend Server

- **Framework**: NestJS or Django for handling business logic and API endpoints.
- **Database**: PostgreSQL for structured data storage.
- **Cache**: Redis for session management and quick access to frequently requested data.
- **Media Storage**: Firebase Storage or Amazon S3 for storing images, videos, and other media files.

Additional Tools

- **Push Notifications**: Firebase Cloud Messaging (FCM) for real-time notifications.
- **Authentication**: Firebase Auth or custom token-based login for email-based authentication.
- **Cloud Infrastructure**: Google Cloud Platform (GCP) or AWS for hosting and scalability.

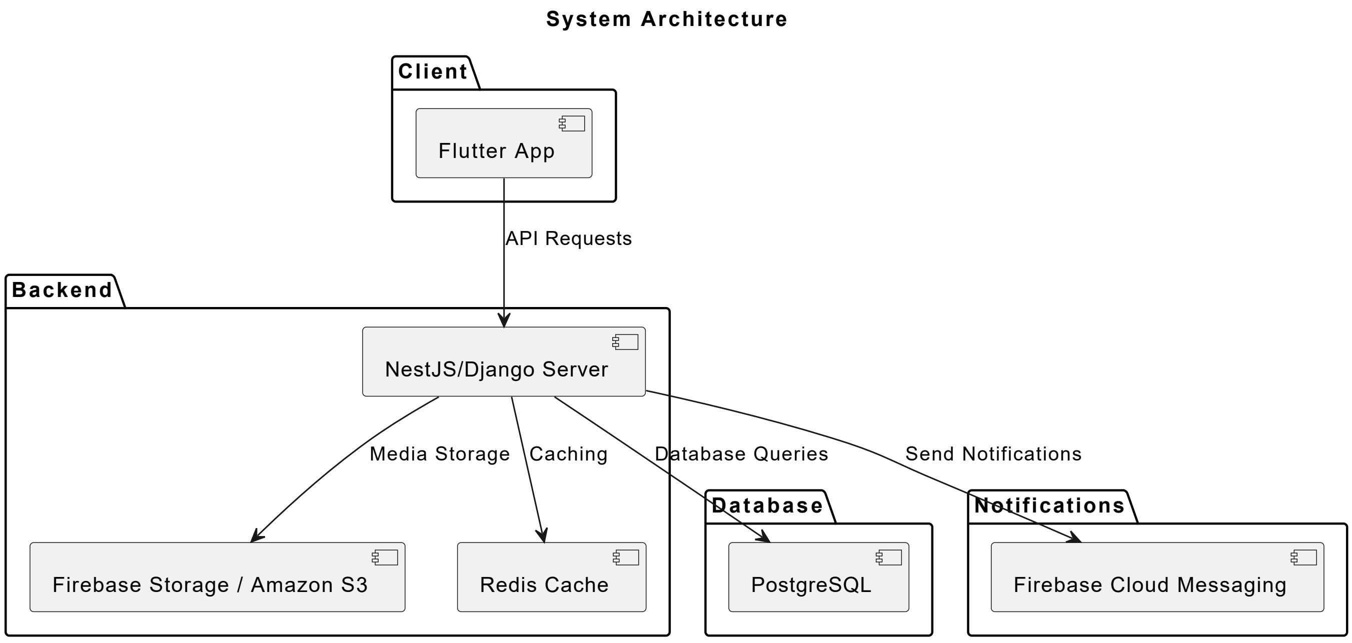
System Architecture

Overview

The system architecture consists of:

1. **Client Application**: Flutter-based app handling user interactions and real-time messaging through WebSocket connections.
2. **Backend API**: Built with NestJS or Django, this server manages user data, message routing, and API requests.
3. **Database Layer**: PostgreSQL to store structured data, with optimized indexing for search efficiency.
4. **Cache Layer**: Redis, for session management and caching frequent data, like user online status.
5. **Media Storage**: Firebase Storage or Amazon S3 for handling media uploads and secure access.
6. **Push Notification System**: Firebase Cloud Messaging to manage message notifications, calls, and other alerts.

Architecture Diagram



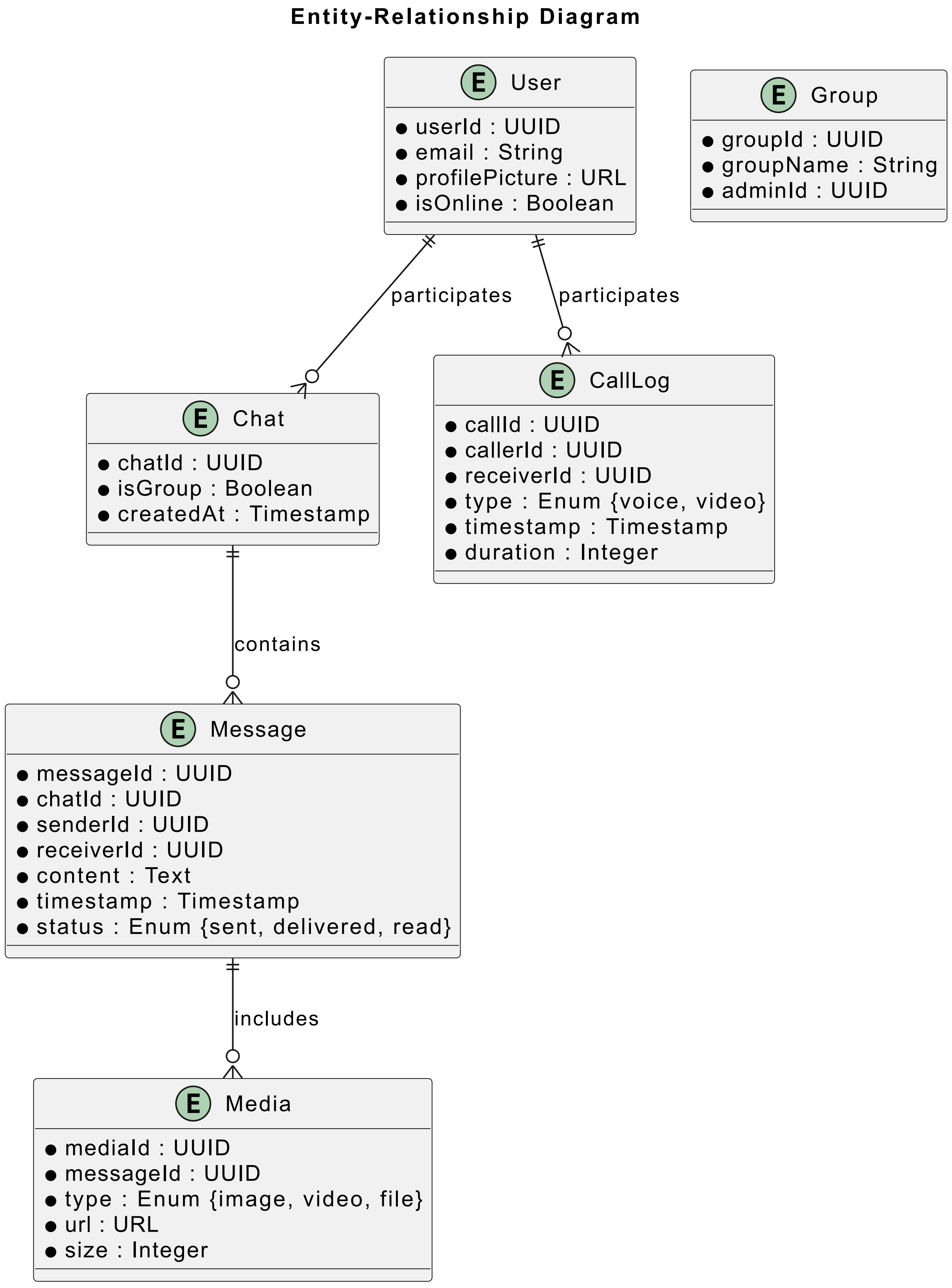
Database Schema

Key Entities and Relationships

1. **User**
   - Stores user information, including email, profile picture, and online status.
   - Columns: `userId`, `email`, `profilePicture`, `isOnline`.

2. **Chat**
   - Represents a conversation between users, supporting both one-on-one and group chats.
   - Columns: `chatId`, `isGroup`, `createdAt`.
3. **Message**
   - Stores individual messages, with fields for sender, recipient, content, and status (sent, delivered, read).
   - Columns: `messageId`, `chatId`, `senderId`, `receiverId`, `content`, `timestamp`, `status`.
4. **Group**
   - For group chats, with details about group members and admin.
   - Columns: `groupId`, `groupName`, `adminId`, `memberIds`.
5. **Media**
   - Stores metadata for media shared within chats.
   - Columns: `mediaId`, `messageId`, `type`, `url`, `size`.
6. **Call Log**
   - Tracks call details between users, including type and duration.
   - Columns: `callId`, `callerId`, `receiverId`, `type` (voice/video), `timestamp`, `duration`.

Entity Relationship Diagram (ERD)



API Documentation

Authentication

Register

- **Endpoint**: `POST /api/auth/register`

- **Description**: Registers a new user using email verification.
- **Request Body**:

```
{
  "email": "user@example.com",
  "password": "securepassword"
}
```

- **Response**:

```
{
  "userId": "abc123",
  "token": "jwt_token"
}
```

Login

- **Endpoint**: `POST /api/auth/login`
- **Description**: Logs in a user and issues a JWT token.
- **Request Body**:

```
{
  "email": "user@example.com",
  "password": "securepassword"
}
```

Messaging

Send Message

- **Endpoint**: `POST /api/messages/send`
- **Description**: Sends a message from one user to another or a group.
- **Request Body**:

```
{
  "senderId": "abc123",
  "receiverId": "xyz456",
  "content": "Hello!"
}
```

- **Response**:

```
{
  "messageId": "msg789",
  "status": "sent"
}
```

Get Messages

- **Endpoint**: `GET /api/messages/{chatId}`
- **Description**: Retrieves messages for a specific chat.
- **Response**:

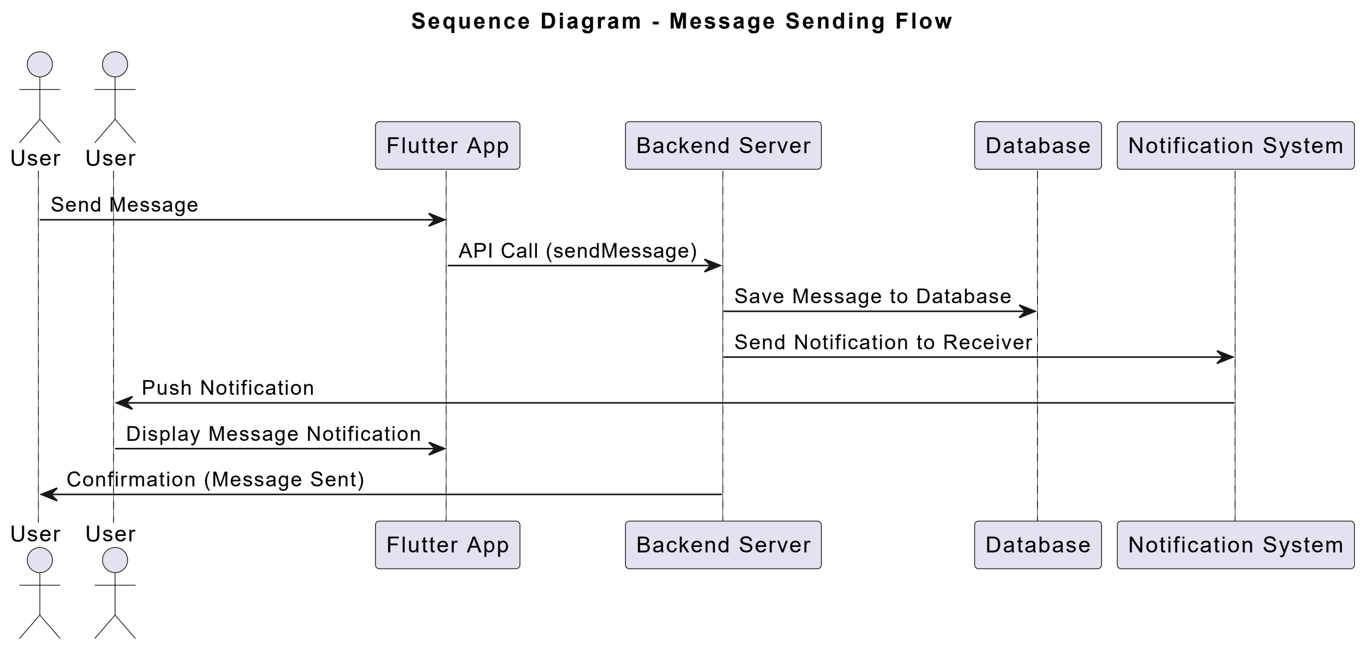
```
[
  {
    "messageId": "msg789",
    "senderId": "abc123",
    "content": "Hello!",
    "timestamp": "2024-01-01T10:00:00Z",
    "status": "read"
  }
]
```

Media Upload

- **Endpoint**: `POST /api/media/upload`
- **Description**: Uploads media files to Firebase Storage or Amazon S3.
- **Request Body**: File in multipart/form-data.
- **Response**:

```
{
  "mediaId": "media456",
  "url": "https://storage-service.com/media456.jpg"
}
```

Sequence Digram (Message Sending Flow)



Security Measures

Encryption

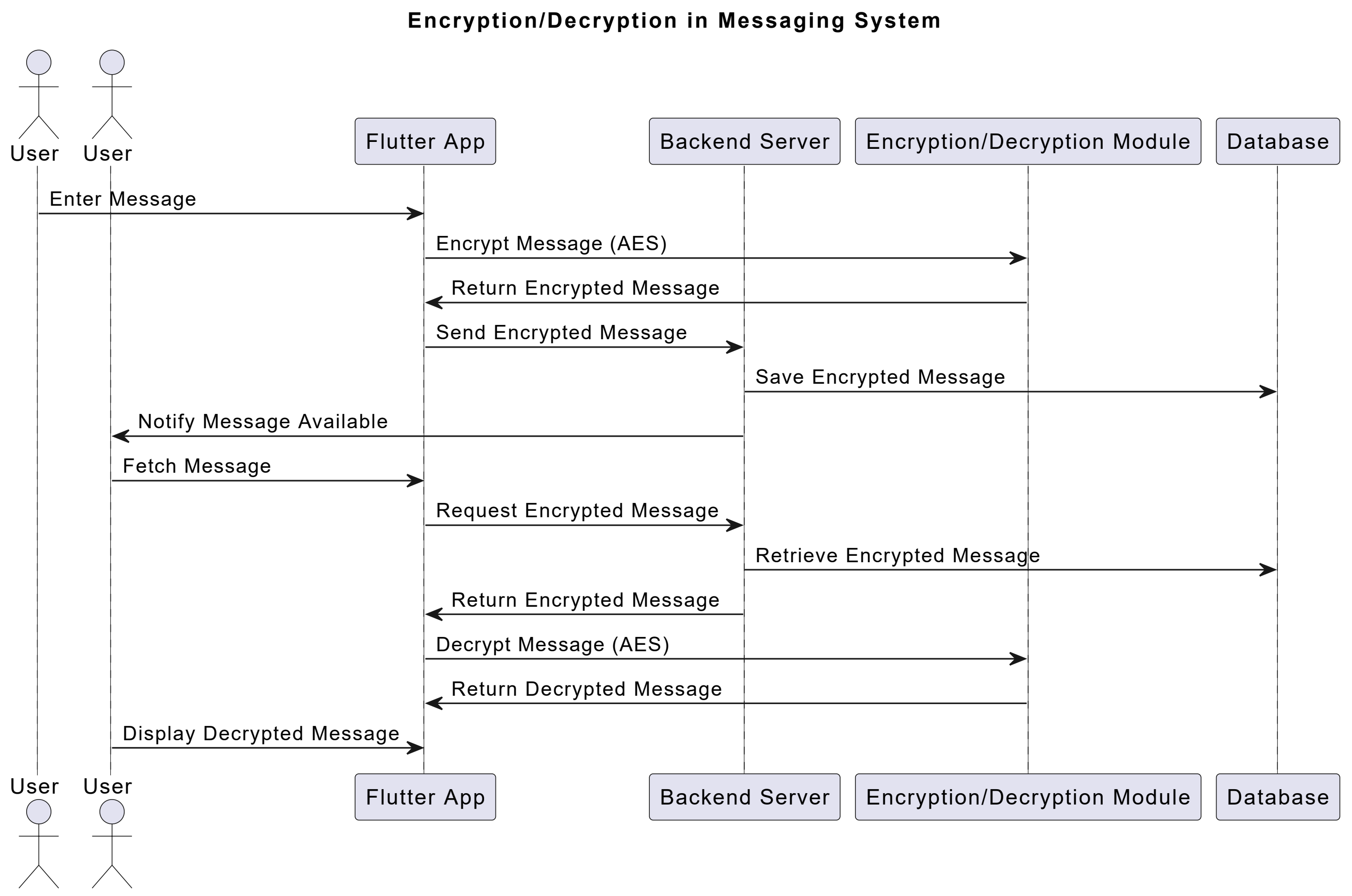
- **End-to-End Encryption**: Messages and calls are encrypted using RSA/AES for privacy.
- **Data in Transit**: Encrypted with HTTPS to protect data integrity.

Authentication

- **JWT Token**: Used for secure sessions.
- **Email Verification**: Users register using email and password, with optional email verification for added security.

Privacy and Compliance

- **GDPR Compliance**: Data retention policies and user data management meet GDPR standards.



Scaling Plan

Database Sharding

As user and message data grows, we'll implement sharding on the message and user tables to distribute load effectively.

Load Balancing

Use load balancing across multiple servers to ensure high availability and reduce latency.

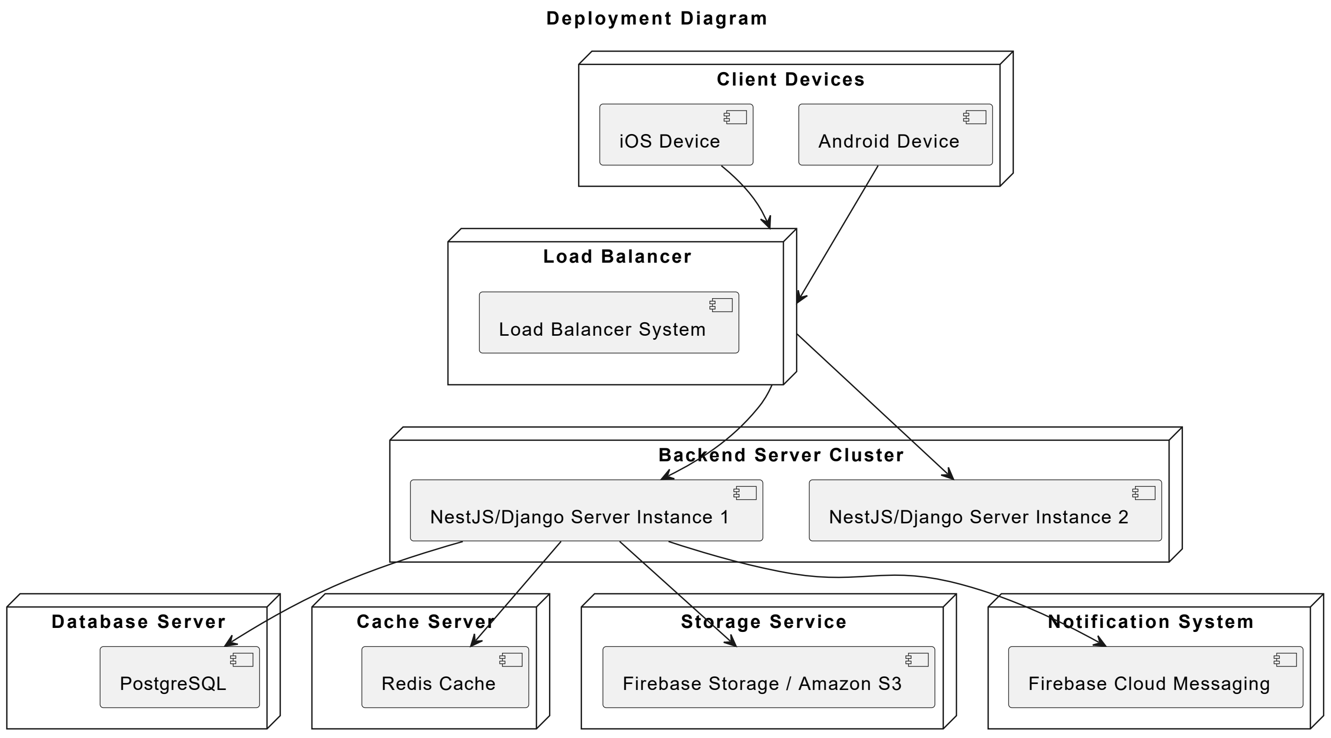
Caching

Redis caching for frequently accessed data (e.g., session management).

Backup Strategy

Automated daily backups of PostgreSQL databases and Firebase Storage to prevent data loss.

Deployment Diagram

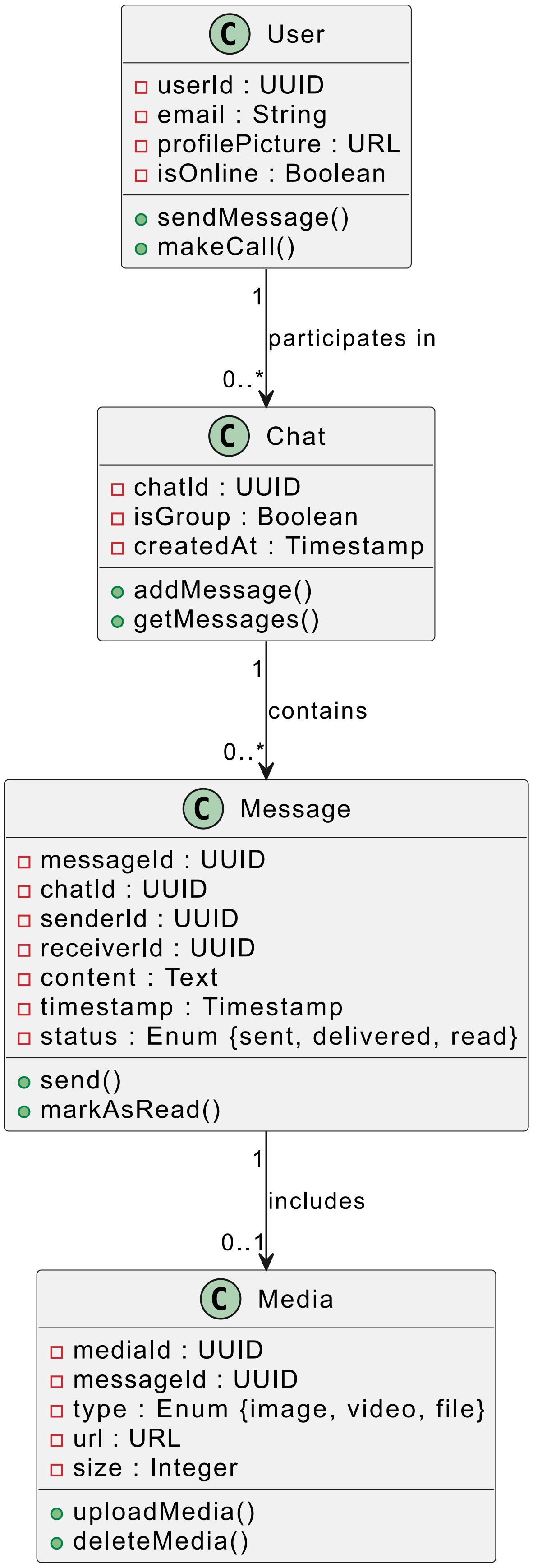


UML Class Diagram

Key Classes

1. **User**
    - **Attributes**: `userId`, `email`, `profilePicture`, `isOnline`.
    - **Methods**: `sendMessage()`, `makeCall()`.
2. **Chat**
    - **Attributes**: `chatId`, `isGroup`, `createdAt`.
    - **Methods**: `addMessage()`, `getMessages()`.
3. **Message**
    - **Attributes**: `messageId`, `chatId`, `senderId`, `receiverId`, `content`, `timestamp`, `status`.
    - **Methods**: `send()`, `markAsRead()`.
4. **Media**
    - **Attributes**: `mediaId`, `messageId`, `type`, `url`, `size`.
    - **Methods**: `uploadMedia()`, `deleteMedia()`.

UML Class Diagram


User
userId : UUID
email : String
profilePicture : URL
isOnline : Boolean
sendMessage()
makeCall()
1
participates in
0..*
Chat
chatId : UUID
isGroup : Boolean
createdAt : Timestamp
addMessage()
getMessages()
1
contains
0..*
Message
messageId : UUID
chatId : UUID
senderId : UUID
receiverId : UUID
content : Text
timestamp : Timestamp
status : Enum {sent, delivered, read}
send()
markAsRead()
1
includes
0..1
Media
mediaId : UUID
messageId : UUID
type : Enum {image, video, file}
url : URL
size : Integer
uploadMedia()
deleteMedia()
