Text based erd

1. **User**
   * **Attributes**: UserID, Username, FirstName, LastName, Email, RoleID, DepartmentID, Password, Status
   * **Relationships**:
     + One-to-Many with **Role** (RoleID)
     + Many-to-One with **Department** (DepartmentID)
     + One-to-Many with **Message** (SenderID)
     + One-to-Many with **Group** (GroupAdminID)
     + Many-to-Many with **Group** through **GroupMembers**
2. **Role**
   * **Attributes**: RoleID, RoleName
   * **Relationships**:
     + One-to-Many with **User** (RoleID)
3. **Department**
   * **Attributes**: DepartmentID, DepartmentName
   * **Relationships**:
     + One-to-Many with **User** (DepartmentID)
     + One-to-Many with **Group** (DepartmentID)
4. **Message**
   * **Attributes**: MessageID, Content, SenderID, RecipientUserID, RecipientGroupID, Timestamp, ReadStatus
   * **Relationships**:
     + Many-to-One with **User** (SenderID)
     + Many-to-One with **User** (RecipientUserID)
     + Many-to-One with **Group** (RecipientGroupID)
5. **Group**
   * **Attributes**: GroupID, GroupName, GroupAdminID, DepartmentID
   * **Relationships**:
     + Many-to-One with **User** (GroupAdminID)
     + Many-to-One with **Department** (DepartmentID)
     + Many-to-Many with **User** through **GroupMembers**
6. **GroupMembers**
   * **Attributes**: GroupID, UserID
   * **Relationships**:
     + Many-to-Many with **Group**
     + Many-to-Many with **User**
7. **Requirements Analysis**:
   * Identify all functional and non-functional requirements.
   * Understand user roles and their specific needs.
8. **Architecture Design**:
   * Decide on the architecture style (e.g., RESTful API, Microservices).
   * Plan the tech stack (e.g., backend framework, database, authentication service).
9. **Database Design**:
   * Finalize your database schema based on the ERD.
   * Plan for indexing, partitioning, and normalization.
10. **API Design**:
    * Define API endpoints for all CRUD operations.
    * Plan request and response formats (e.g., JSON).
11. **Authentication and Authorization**:
    * Implement user authentication (e.g., JWT, OAuth).
    * Define authorization rules for different user roles.
12. **User Interface Design**:
    * Plan the UI/UX for the Android and web applications.
    * Create wireframes or mockups.
13. **Testing Strategy**:
    * Define unit tests, integration tests, and end-to-end tests.
    * Plan for continuous integration/continuous deployment (CI/CD).
14. **Deployment Plan**:
    * Choose cloud providers or on-premise solutions.
    * Plan for scalability, load balancing, and backup.

# Requirements Analysis

**Functional Requirements**

**User Management**

1. **User Registration**:
   * Users can register with their username, first name, last name, email, password, role, and department.
   * Duplicate email and username are not allowed.
2. **User Login**:
   * Users can log in using their email and password.
   * Implement JWT for session management.
3. **User Profiles**:
   * Users can view and update their profile information.
   * Admins can view and manage all user profiles.

**Role Management**

1. **Role Assignment**:
   * Users can have roles such as Doctor, Nurse, Staff, and Admin.
   * Roles define access levels and permissions.
2. **Role Management**:
   * Admins can create, update, and delete roles.

**Messaging**

1. **Send Messages**:
   * Users can send messages to other users.
   * Users can send messages to groups.
2. **Receive Messages**:
   * Users can receive messages from other users.
   * Users can receive messages from groups they belong to.
3. **Group Messaging**:
   * Users can create groups with a group name, admin, and members.
   * Users can send messages to all group members.
   * Group admins can add or remove members.
4. **Message Read Status**:
   * Track whether a message has been read by the recipient.
5. **Message History**:
   * Users can view their message history with other users and groups.

**Group Management**

1. **Create Group**:
   * Users can create groups with a specified name and admin.
2. **Manage Group**:
   * Group admins can add or remove members.
   * Group admins can update group details.
3. **View Groups**:
   * Users can view the groups they belong to.
   * Admins can view all groups.

**Non-Functional Requirements**

**Performance**

1. **Scalability**:
   * The system should handle increasing numbers of users and messages.
2. **Response Time**:
   * API requests should be handled within acceptable response times (< 500ms for most operations).

**Security**

1. **Authentication**:
   * Implement secure authentication using JWT.
2. **Authorization**:
   * Ensure users can only perform actions permitted by their roles.
3. **Data Protection**:
   * Encrypt sensitive data, such as passwords.
   * Use HTTPS for secure data transmission.

**Usability**

1. **API Documentation**:
   * Provide clear API documentation for developers.
2. **User Interface**:
   * Design user-friendly interfaces for both Android and web applications.

**Reliability**

1. **Availability**:
   * Ensure high availability of the system with minimal downtime.
2. **Backup and Recovery**:
   * Implement regular data backups and recovery procedures.

**Maintainability**

1. **Code Quality**:
   * Follow best practices for code quality and documentation.
2. **Testing**:
   * Implement unit tests, integration tests, and end-to-end tests.
3. **Continuous Integration/Continuous Deployment (CI/CD)**:
   * Set up CI/CD pipelines for automated testing and deployment.

**Use Cases**

1. **User Registration and Login**
   * User registers with required details.
   * User logs in with email and password.
2. **Profile Management**
   * User updates their profile information.
   * Admin views and manages user profiles.
3. **Role Management**
   * Admin creates, updates, or deletes roles.
4. **Sending Messages**
   * User sends a message to another user or a group.
5. **Receiving Messages**
   * User receives messages from other users or groups.
6. **Group Management**
   * User creates a group.
   * Group admin adds/removes members.
   * User views the groups they belong to.

**Detailed Requirements Document**

**User Registration**

* Input: Username, FirstName, LastName, Email, Password, Role, Department
* Output: Success message or error (e.g., "Email already exists")

**User Login**

* Input: Email, Password
* Output: JWT token or error (e.g., "Invalid credentials")

**Send Message**

* Input: SenderID, RecipientID (User or Group), Content
* Output: Success message or error

**Receive Message**

* Input: UserID or GroupID
* Output: List of messages

**Create Group**

* Input: GroupName, GroupAdminID, DepartmentID
* Output: Success message or error

**Manage Group**

* Input: GroupID, UserID (for adding/removing members)
* Output: Success message or error

**Update Profile**

* Input: UserID, profile details to be updated
* Output: Success message or error

# Architecture Design

### Architecture Overview

1. **Client Applications**:
   * **Android Application**
   * **Web Application**
2. **API Gateway**:
   * Acts as a single entry point for all client requests.
   * Routes requests to the appropriate microservices.
   * Handles authentication, rate limiting, and logging.
3. **Microservices**:
   * **User Service**: Manages user registration, authentication, and profile management.
     + 1. user-service/
       2. ├── .env
       3. ├── package.json
       4. ├── server.js
       5. ├── config/
       6. │ └── db.js
       7. │ └── swagger.js
       8. ├── controllers/
       9. │ └── userController.js
       10. ├── models/
       11. │ └── User.js
       12. ├── routes/
       13. │ └── userRoutes.js
       14. ├── middlewares/
       15. │ └── authMiddleware.js
       16. └── utils/
       17. └── generateToken.js
   * **Role Service**: Manages roles and permissions.
     + 1. role-service/
       2. ├── .env
       3. ├── package.json
       4. ├── server.js
       5. ├── config/
       6. │ └── db.js
       7. │ └── swagger.js
       8. ├── controllers/
       9. │ └── roleController.js
       10. ├── models/
       11. │ └── Role.js
       12. ├── routes/
       13. │ └── roleRoutes.js
       14. └── middlewares/
       15. └── authMiddleware.js
   * **Message Service**: Handles sending, receiving, and storing messages.
     + 1. message-service/
       2. ├── .env
       3. ├── package.json
       4. ├── server.js
       5. ├── config/
       6. │ └── db.js
       7. │ └── swagger.js
       8. ├── controllers/
       9. │ └── messageController.js
       10. ├── models/
       11. │ └── Message.js
       12. ├── routes/
       13. │ └── messageRoutes.js
       14. └── middlewares/
       15. └── authMiddleware.js
   * **Group Service**: Manages group creation, membership, and group messaging.
     + 1. group-service/
       2. ├── .env
       3. ├── package.json
       4. ├── server.js
       5. ├── config/
       6. │ └── db.js
       7. │ └── swagger.js
       8. ├── controllers/
       9. │ └── groupController.js
       10. ├── models/
       11. │ └── Group.js
       12. ├── routes/
       13. │ └── groupRoutes.js
       14. └── middlewares/
       15. └── authMiddleware.js
   * **Notification Service**: Manages real-time notifications for new messages.
     + 1. notification-service/
       2. ├── .env
       3. ├── package.json
       4. ├── server.js
       5. ├── config/
       6. │ └── db.js
       7. │ └── swagger.js
       8. ├── controllers/
       9. │ └── notificationController.js
       10. ├── models/
       11. │ └── Notification.js
       12. ├── routes/
       13. │ └── notificationRoutes.js
       14. └── middlewares/
       15. └── authMiddleware.js
   * **Logging Service**: Centralized logging for all microservices.
     + 1. logging-service/
       2. ├── .env
       3. ├── package.json
       4. ├── server.js
       5. ├── config/
       6. │ └── db.js
       7. │ └── swagger.js
       8. ├── controllers/
       9. │ └── logController.js
       10. ├── models/
       11. │ └── Log.js
       12. ├── routes/
       13. │ └── logRoutes.js
       14. └── middlewares/
       15. └── authMiddleware.js
   * **Monitoring Service**: Monitors the health and performance of microservices.
     + 1. monitoring-service/
       2. ├── .env
       3. ├── package.json
       4. ├── server.js
       5. ├── config/
       6. │ └── db.js
       7. │ └── swagger.js
       8. ├── controllers/
       9. │ └── monitoringController.js
       10. ├── models/
       11. │ └── ServiceStatus.js
       12. ├── routes/
       13. │ └── monitoringRoutes.js
       14. └── middlewares/
       15. └── authMiddleware.js
4. **Database**:
   * **User Database**: Stores user-related data.
   * **Message Database**: Stores messages and related data.
   * **Group Database**: Stores group-related data.
5. **Authentication and Authorization**:
   * **Auth Service**: Handles JWT token generation and validation.
6. **Inter-service Communication**:
   * **Message Queue**: For asynchronous communication between microservices (e.g., RabbitMQ, Kafka).
7. **External Services**:
   * **Email Service**: For sending verification emails and notifications.
   * **Push Notification Service**: For sending push notifications to mobile devices.