System Design Documentation

Overview

This document provides a detailed system design for the messaging application, including the architecture, components, workflows, and interactions.

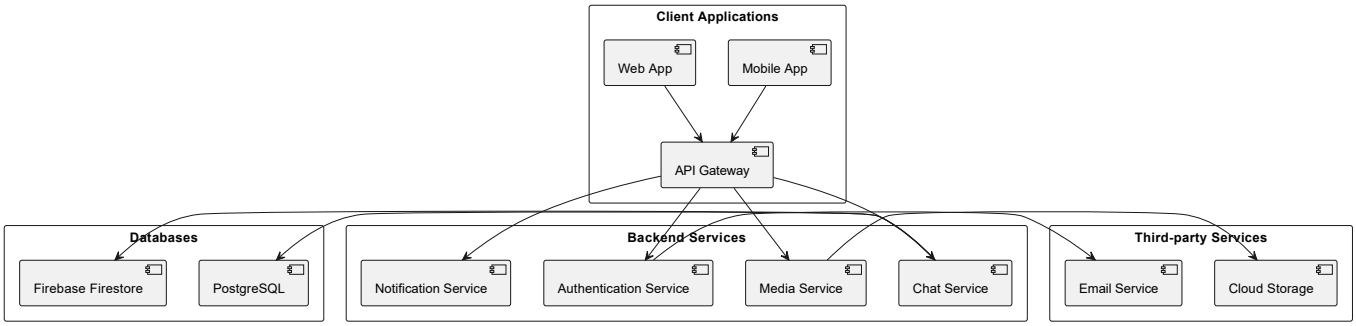
System Architecture

Description

The system architecture follows a **microservices-based architecture** to ensure scalability, maintainability, and fault tolerance. Key components include:

1. **Client Applications**:
    - Mobile App (Flutter for Android/iOS)
    - Web App (React/Next.js)
2. **Backend Services**:
    - API Gateway (NestJS)
    - Authentication Service
    - Chat Service
    - Notification Service
    - Media Service
3. **Databases**:
    - PostgreSQL for relational data
    - Firebase Firestore for real-time chat synchronization
4. **Third-party Services**:
    - Cloud Storage for media (e.g., Google Cloud Storage)
    - Email Service (e.g., SendGrid for account verification)

Architecture Diagram

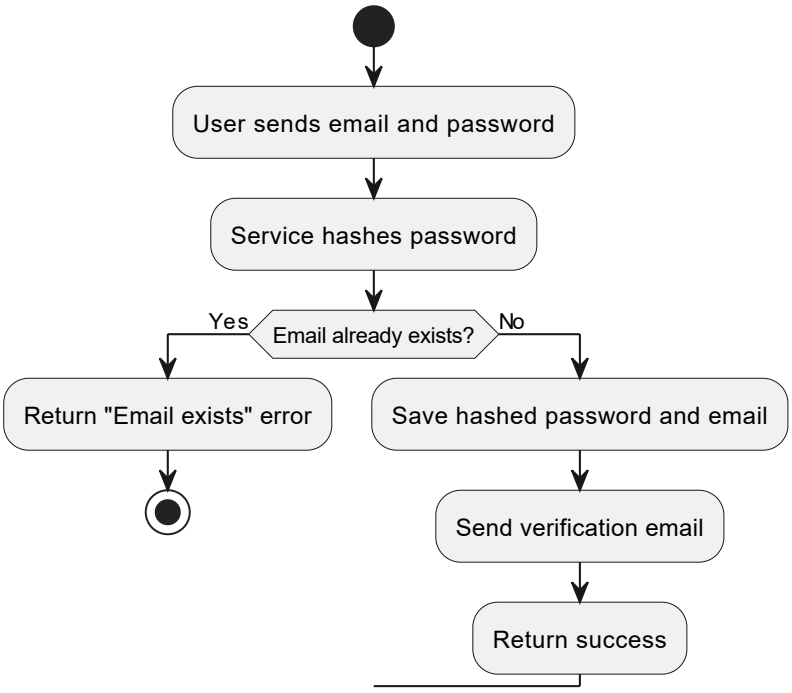


Component Design

1. Authentication Service

- **Responsibilities**:
  - Manage user registration and login.
  - Handle password hashing and token generation.
  - Provide APIs for authentication-related actions.
- **Interactions**:
  - Uses PostgreSQL to store user credentials.
  - Integrates with the email service for verification emails.

Authentication Flow

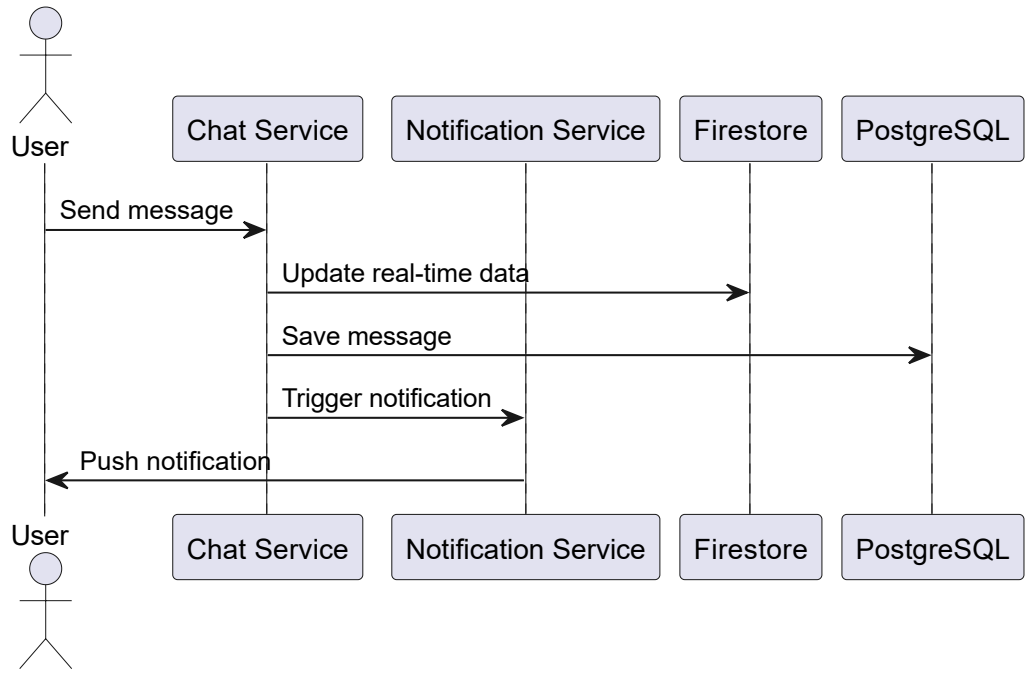


---

2. Chat Service

- **Responsibilities**:
  - Manage real-time chat functionalities.
  - Store messages in PostgreSQL and sync real-time data using Firestore.
  - Provide APIs for sending and retrieving messages.
- **Interactions**:
  - Communicates with the notification service to push new message alerts.
  - Accesses Cloud Storage for media attachments.

Message Flow



---

3. Notification Service

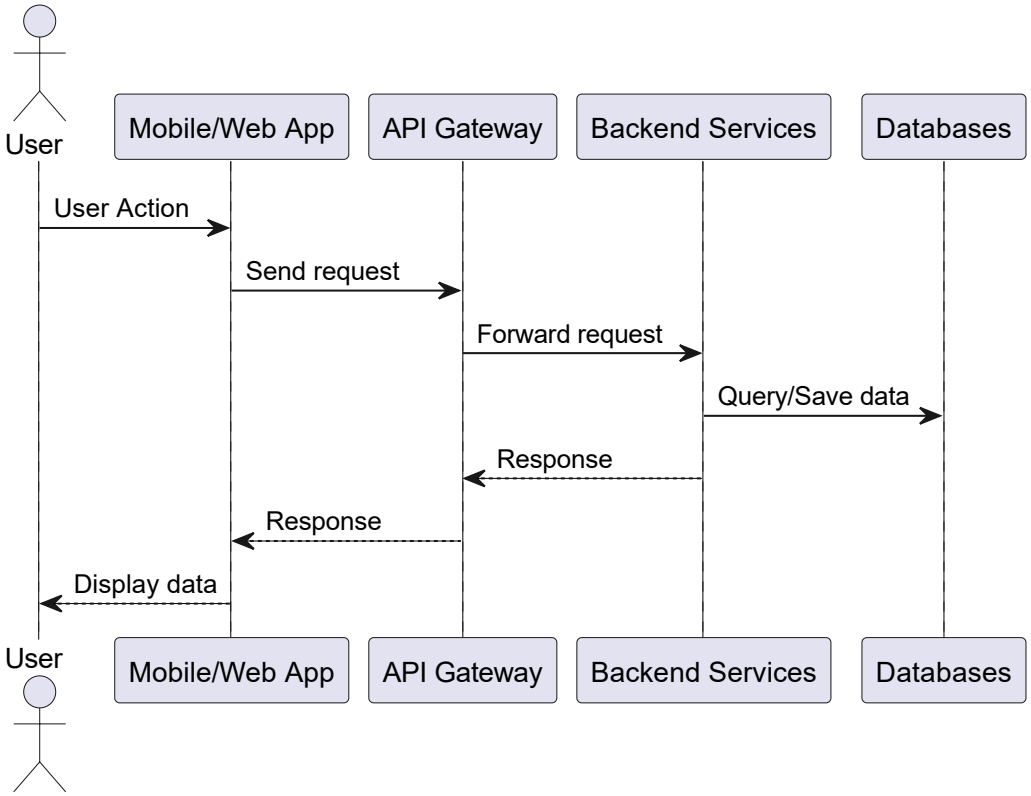
- **Responsibilities**:
  - Manage push notifications for new messages, friend requests, etc.
  - Provide APIs to retrieve notification history.
- **Interactions**:
  - Uses Firestore for real-time updates.
  - Sends push notifications to devices using Firebase Cloud Messaging (FCM).

Data Flow

Description

Data flows between the client, backend, and database using RESTful APIs and real-time synchronization mechanisms.

Data Flow

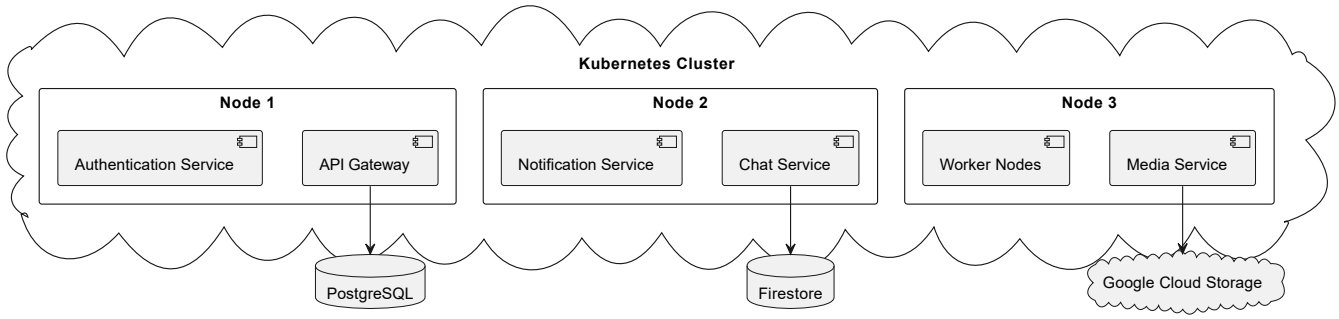


Deployment Design

Description

The deployment utilizes containerized services for flexibility and scalability. Kubernetes orchestrates the deployment across multiple nodes.

Deployment Architecture



Scaling Strategy

1. Use **horizontal scaling** for backend services to handle increased traffic.
2. Employ **read replicas** for PostgreSQL to distribute read-heavy operations.
3. Cache frequently accessed data using **Redis** to reduce database load.
4. Use **load balancers** to evenly distribute traffic across instances.

Security Measures

1. Use **OAuth 2.0** for secure authentication.
2. Encrypt data in transit using **HTTPS** and at rest using PostgreSQL's encryption.
3. Sanitize user inputs to prevent SQL injection and XSS attacks.
4. Implement **rate limiting** to mitigate DDoS attacks.
