System Architect Documentation

Overview

This document provides a comprehensive overview of the architecture for our email-based messaging app. It focuses on high-level system design, technology stack, deployment strategies, scalability, and security measures.

High-Level Architecture

The messaging app is designed as a distributed system using microservices. It includes the following key components:

1. **Frontend** (Flutter Web and Mobile Apps)
2. **Backend** (Node.js with NestJS Framework)
3. **Database** (PostgreSQL for relational data and Firebase Firestore for real-time messaging)
4. **Authentication** (Email-based login with JWT for session management)
5. **Cloud Hosting** (Google Cloud Platform)
6. **Encryption** (End-to-end encryption for messages)

Technology Stack

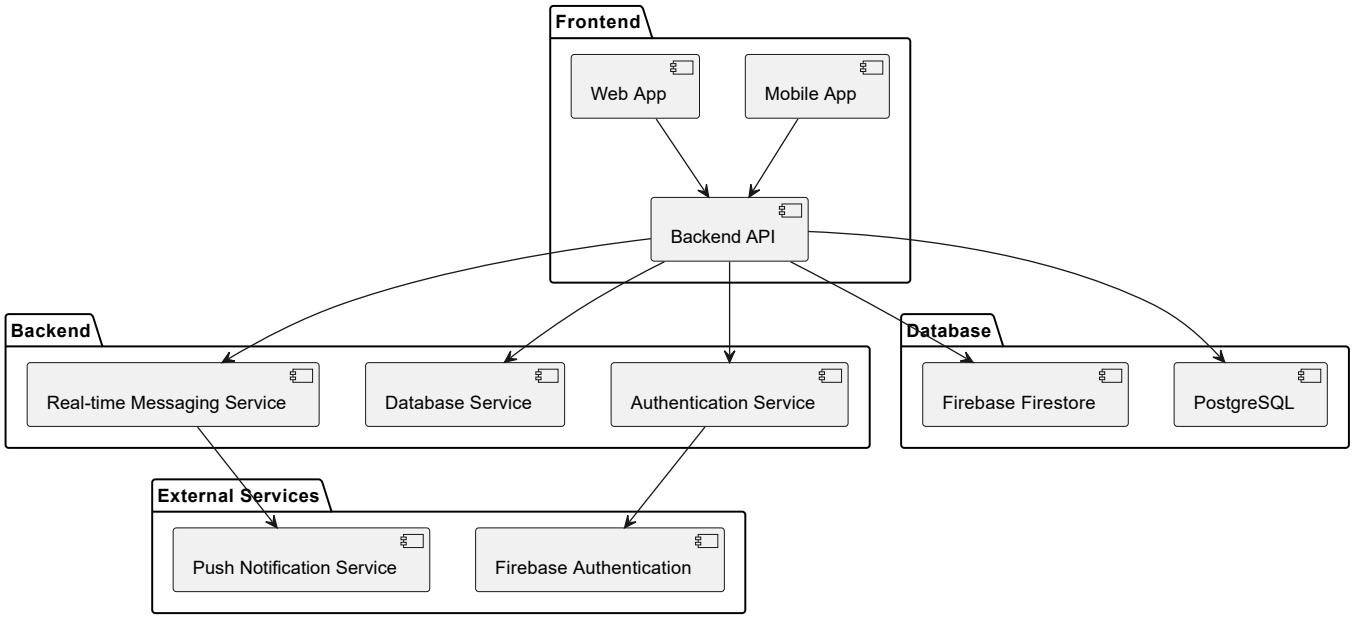
Component	Technology
Frontend	Flutter
Backend	Node.js (NestJS)
Database	PostgreSQL, Firebase Firestore
Authentication	Firebase Authentication
Cloud Hosting	GCP (Google Kubernetes Engine, Cloud Functions)
Encryption	RSA and AES algorithms

Deployment and Scalability

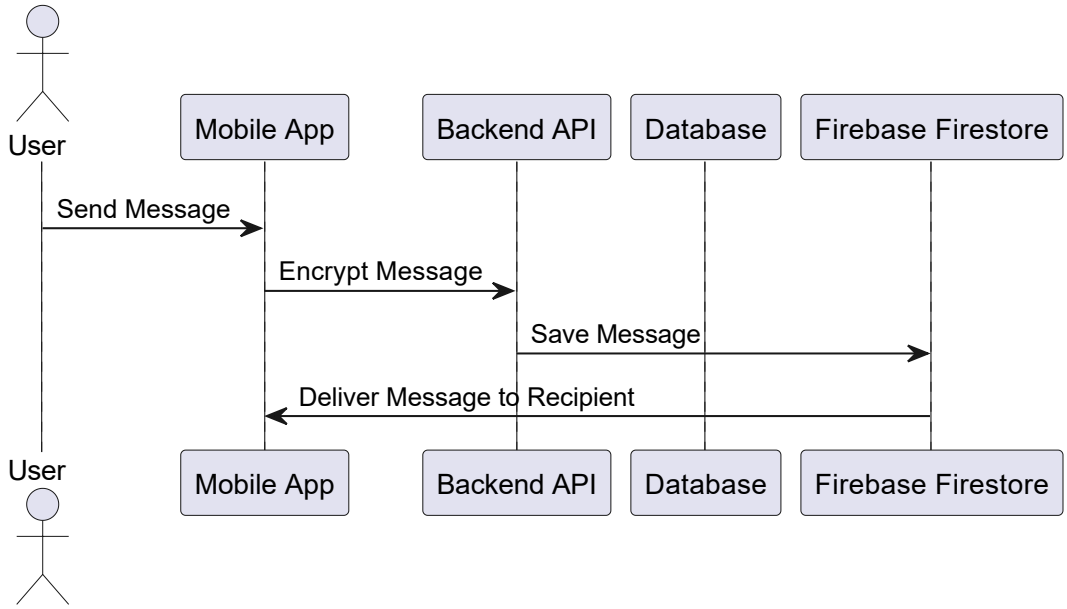
1. **Containers**: Docker is used to containerize the backend services.
2. **Orchestration**: Kubernetes ensures scalability and high availability.
3. **CI/CD Pipeline**: Automated deployment pipeline using GitHub Actions.
4. **Monitoring**: Prometheus and Grafana for system performance monitoring.

Diagrams

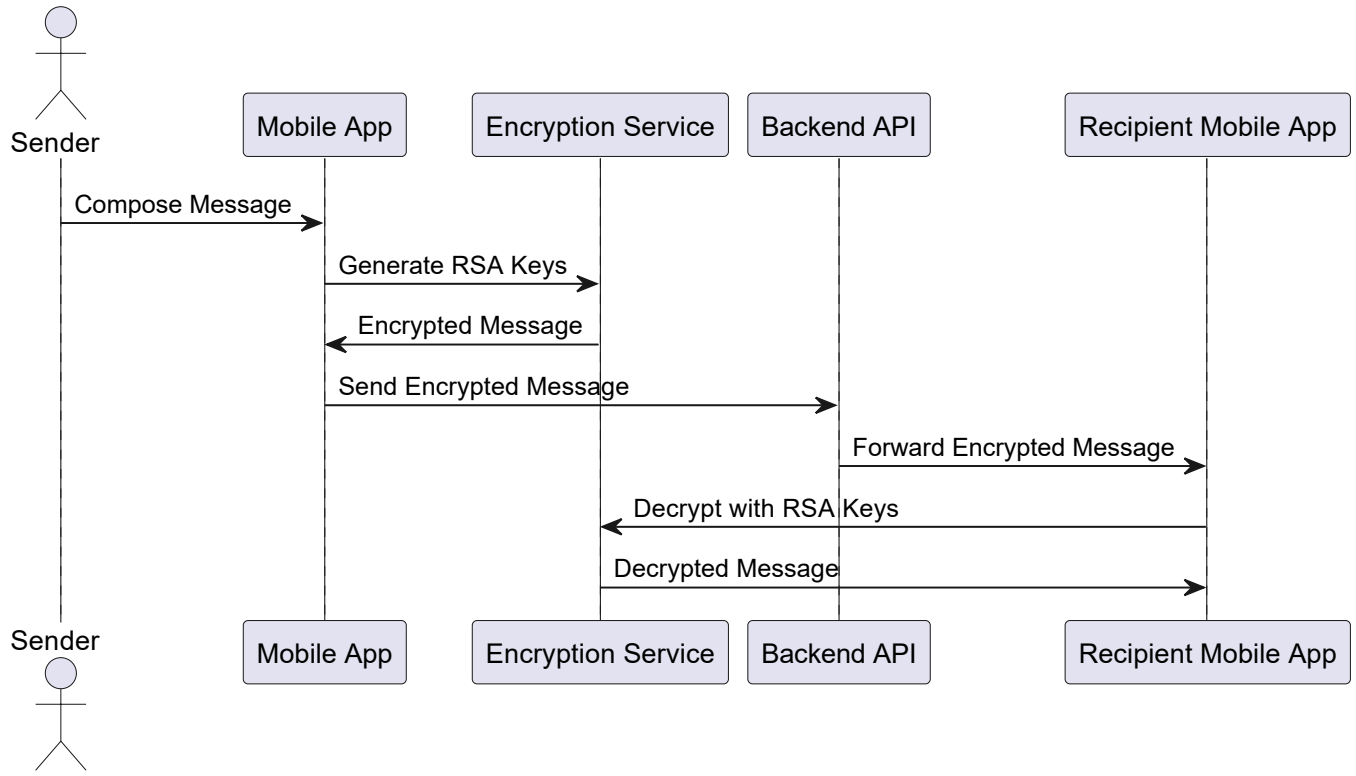
High-Level System Architecture



Message Flow Diagram



Security and Encryption Workflow



Security Measures

1. **End-to-End Encryption (E2EE)**:
   - Messages are encrypted with AES before being sent.
   - RSA is used to exchange keys securely.
2. **Authentication**:
   - Firebase Authentication for email-based login.
   - JWT tokens for secure session management.
3. **Data Security**:
   - Messages are stored encrypted in Firebase Firestore.
   - User credentials are stored in Firebase Authentication.

Scalability and Failover

- **Auto-Scaling**: Kubernetes automatically scales services based on traffic.
- **Load Balancing**: GCP's Load Balancer ensures equal traffic distribution.
- **Backup**: Daily database backups using GCP Cloud Functions.
- **Failover Mechanism**: Replicated database setup for failover handling.
