ChatApp Documentation

1. Genel Bakış

Bu doküman, React/TypeScript tabanlı frontend ile Express.js + MongoDB + Socket.IO + RabbitMQ kullanan backend mimarisine sahip bir sohbet uygulamasını (ChatApp) detaylı şekilde anlatır.

2. Mimariye Genel Bakış

* **Frontend**: React, React Router, Bootstrap, Zustand (store)
* **Backend**: Node.js, Express.js, MongoDB (Mongoose), Socket.IO
* **Mesaj Kuyruğu**: RabbitMQ (amqplib)
* **Gerçek Zamanlı Haberleşme**: Socket.IO
* **Dokümantasyon**: Swagger/OpenAPI

3. Kurulum & Başlatma

* **Ortam Değişkenleri** (.env):
* NODE\_ENV=development  
  PORT=5000  
  MONGODB\_URI=mongodb://localhost:27017/chat-app  
  RABBITMQ\_URL=amqp://localhost  
  JWT\_SECRET=your-jwt-secret  
  CORS\_ORIGIN=http://localhost:3000, [http://localhost:5000](http://localhost:5000/)
* **Backend**
* cd backend  
  npm install  
  npm run consumer *# messageLogger consumer için*  
  npm start *# express + socket.io + rabbitmq*
* **Frontend**
* cd web-client  
  npm install  
  npm start *# React uygulamasını başlatır*
* **Swagger UI**
* Açık: <http://localhost:5000/api-docs>

4. API Referansı

4.1 Auth

|  |  |  |
| --- | --- | --- |
| Metot | Endpoint | Açıklama |
| POST | /api/auth/register | Yeni kullanıcı kaydı |
| POST | /api/auth/login | Giriş işlemi |
| POST | /api/auth/refresh-token | Access token yenileme |
| POST | /api/auth/verify-email | E‑posta doğrulama (placeholder) |
| POST | /api/auth/logout*auth* | Çıkış |
| GET | /api/auth/profile*auth* | Profil bilgisi |
| PUT | /api/auth/profile*auth* | Profil güncelleme |
| POST | /api/auth/change-password*auth* | Şifre değişikliği |

4.2 Chats

|  |  |  |
| --- | --- | --- |
| Metot | Endpoint | Açıklama |
| GET | /api/chats*auth* | Kullanıcının sohbetleri (pagi) |
| POST | /api/chats*auth* | Yeni sohbet oluştur |
| GET | /api/chats/:id*auth* | Belirli sohbet detayları |
| DELETE | /api/chats/:id*auth* | Sohbet sil |
| GET | /api/chats/:id/messages*auth* | Mesajları getir (pagi) |
| POST | /api/chats/:id/messages*auth* | Mesaj gönder |
| PUT | /api/chats/:id/read*auth* | Mesajları okundu olarak işaretle |
| POST | /api/chats/:id/participants*auth* | Grup sohbetine katılımcı ekle |
| DELETE | /api/chats/:id/participants*auth* | Katılımcı çıkar |

4.3 Users

|  |  |  |
| --- | --- | --- |
| Metot | Endpoint | Açıklama |
| GET | /api/users/search*auth* | Kullanıcı ara (q, pagi) |
| GET | /api/users/online*auth* | Çevrimiçi kullanıcılar |
| GET | /api/users/contacts*auth* | İletişim listesi |
| GET | /api/users/stats*auth* | Kullanıcı istatistikleri |
| GET | /api/users/:id*auth* | Kullanıcı profili |
| POST | /api/users/:id/block*auth* | Kullanıcı engelleme (placeholder) |
| POST | /api/users/:id/report*auth* | Kullanıcı şikayet (placeholder) |

\_auth\_: Bearer gerektirir

5. Gerçek Zamanlı Olaylar (Socket.IO)

|  |  |
| --- | --- |
| Event | Açıklama |
| connection | Auth token ile bağlanma |
| join\_room | Chat odasına katıl |
| leave\_room | Chat odasından çık |
| send\_message | Mesaj gönder (payload: { chatId, content, messageType, replyTo? }) |
| message\_received | Yeni mesaj alındığında yayınlanır |
| message\_status\_updated | sent, delivered, read statü güncellemeleri |
| typing | Yazıyor bilgisini yayınlar ({ chatId, isTyping }) |
| user\_online / user\_offline | Kullanıcı çevrimiçi/offline |

6. RabbitMQ Tüketici & Mesaj Kuyruğu

* **Queue**: chat\_messages
* Gönderilen mesajlar JSON formatında:
* {  
   "type": "NEW\_MESSAGE",  
   "chatId": "...",  
   "senderId": "...",  
   "senderUsername": "...",  
   "content": "...",  
   "messageType": "text",  
   "timestamp": "ISO8601",  
   "messageId": "..."  
  }
* **Consumer**: src/services/consumers/messageLogger.js
* Kuyruktan gelen tüm mesajları loglar ve ack() eder.

7. Swagger/OpenAPI Entegrasyonu

* **Yol**: /api-docs
* **Özellikler**:
* components.securitySchemes.bearerAuth ile JWT auth
* Tüm CRUD uç noktaları tanımlı
* Query parametreleri (page, limit, q) & Path parametreleri (:id) açıklamaları hazır

8. İleriki Adımlar

* **Test Kapsamı**: Jest + Supertest ile unit/integration test’ler.
* **CI/CD Pipeline**: GitHub Actions / GitLab CI.
* **Docker Compose**: Servisleri tanımlayıp tek komutla ayağa kaldırma.
* **Mikroservis Geçişi**: Auth, User, Chat, Message servislerini bağımsızlaştırma.

*Bu doküman, projenin tüm bileşenlerini tek bir kaynaktan takip etmek ve geliştirme sürecini hızlandırmak amacıyla hazırlanmıştır.*