**T.C.**

**SAKARYA ÜNİVERSİTESİ**

**BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ**

**BSM451 AĞ PROGRAMLAMARA**

Web Socket programlama ile sohbet uygulaması

**Proje Raporu**

**B201210590-Enes Smajli/B201210010-Yasin Gökçe**

**Hoca: Dr.Öğr.Üyesi MUSA BALTA**

**2024-2025 Güz Dönemi**

# Giriş

# Bu rapor, kullanıcıların kullanımını kolaylaştırmak için uygulamamızın kavramlarını, teknolojilerini ve diğer yönlerini açıklayacaktır. Proje başlığımız: “**Soket programlama (TCP veya UDP temelli) ile kişilerin mesajlaşmasına olanak veren güvenli uygulama**”

# Ana fikir, kullanıcı dostu olacak, soket programlama ve en son güvenlik tekniklerini kullanarak boradcast sohbetleri, gruplar ve hatta özel sohbetler için güvenli bir şekilde mümkün kılacak bir uygulama oluşturmaktır, böylece mesajlar yalnızca yetkili kişiler tarafından okunacaktır.

# Sohbet uygulamalarına baktığımızda, sadece mobil olarak başlayanlar bile, web'in hayatımızın her alanında yüksek kullanımı nedeniyle uygulamalarının web arayüzlerini geliştiriyorlar. Projemiz için bir web kullanıcı arayüzü geliştirmeye karar vermemizin nedenlerinden biri de bu.

# React.js ile olan deneyimimiz ve yüksek uyumluluğu, kolay organizasyonu ve günümüzde trend bir teknoloji olması nedeniyle ön yüzü React.js ile çalıştık. React.js ile çok iyi giden en iyi arka uç teknolojisi, burada API'lerin uç noktalarına sahip olan tüm arka uçlar için kullandığımız Node.js'dir. otherntication ve socket.io kullanarak soket programlama.

# Bu belgenin bölümleri, uygulamamızı nasıl oluşturduğumuza, geliştirdiğimize ve nasıl çalıştıracağımıza dair adım adım bir kılavuz vermek üzere tasarlanmıştır. Her kullanıcının bunu kullanarak harika zaman geçireceğini umuyoruz.

# React/Node çerçevesi oluşturma

1. İlk adım şudur:

* Uygulamamızı oluşturmak istediğimiz klasöre gidiyoruz ve terminalde açıyoruz.
* Ardından şu komutu yazıyoruz: npx create-react-app chat\_secure

Bu, klasörleri oluşturacak ve tüm node\_modüllerini alacaktır, projeye başlamadan önce anlaşılabilir ki Node.js kurulumu gereklidir.

1. İkinci adım ise:

* Oluşturma işlemi tamamlandıktan sonra, komutun oluşturduğu klasöre gidiyoruz ve terminali açıp şunu yazıyoruz: npm start

1. Daha sonra kodlama kısmına geçiyoruz.

React.js bileşen mimarisi ile çalışır, bu nedenle ön uçtaki her öğe bir bileşen olacaktır, bu yeniden kullanılabilirliği yüksek hale getirecek. Bu projenin GitHub deposunda, bu raporda da yorumlayacağımız kod bölümünü bulabileceksiniz.

Uygulamamızın bileşenleri:

* Home.js
* Signup.js
* Login.js
* Chats.js
* ChatRoom.js
* Profil.js
* Create\_groups.js

1. BACKEND

Backend modülünü nasıl organize ettiğimiz aşağıda belirtilmiştir:

* Regsiter
* Login
* Fetch\_users
* Fetch\_messages
* Fetch\_chats
* Server
* Client
* Send\_messages
* Get\_messages
* Offline\_messages

1. DATABASE

Projede veri tabanı sağlayıcısı olarak MySQL kullanamaya karar verdik . Tablolarda nelere ihtiyaç duyduğumuz ve aralarındaki ilişkileri , nasıl göründükleri aşağıda belirtilmiştir.

Tablolar:

* Users
* Profiles
* Roles
* Chats
* Chat\_memebers
* Messages
* Offline\_messages

Here we have a diagram on how tables are related to each other In the GitHub repository you can find the database queries to create a local database for yourself in case you want to test it. But for the presentation we will use our ready database.

# Kütüphaneler

There are other dependencies and some special libraries that needed commands which are explained below.

* npm install react-router-dom
* npm install express
* npm install bcrypt
* npm install cors
* npm install mysql2
* npm install jsonwebtoken
* npm install crypto
* npm install http
* npm install socket.io
* npm install dotenv

Server.js

**Server.js Modülünün Genel Amacı**

* **HTTP ve WebSocket** tabanlı bir uygulama oluşturmak.
* Bir **MySQL veritabanıyla** bağlantı kurmak ve verileri yönetmek.
* **JWT** kullanarak kullanıcı oturumlarını güvenli bir şekilde doğrulamak.
* **Şifreleme ve şifre çözme işlemleri** için bir AES algoritması kullanmak.
* CORS (Cross-Origin Resource Sharing) ve statik dosya sunumu gibi temel özelliklerle API’ye erişimi yönetmek.

1. **Gereken Modüllerin Dahil Edilmesi**

**express**: HTTP tabanlı bir web framework'ü, API'ler oluşturmak için kullanılıyor.

**bcrypt**: Parola hash'leme ve doğrulama işlemleri için.

**cors**: CORS ayarlarını yapmak, farklı kaynaklardan gelen istekleri yönetmek için.

**mysql2/promise**: MySQL veritabanı ile bağlantı kurmak ve sorgular çalıştırmak için.

**jsonwebtoken (jwt)**: Kullanıcı oturumlarının güvenli bir şekilde yönetilmesi için.

**crypto**: AES-256-CBC algoritması ile şifreleme yapmak için.

**http**: HTTP sunucusu oluşturmak için.

**socket.io**: WebSocket bağlantılarını yönetmek için.

**dotenv**: .env dosyasından çevresel değişkenleri yüklemek için.

1. **HTTP ve WebSocket Sunucusunun Kurulumu**

**HTTP sunucusu:** app (Express uygulaması) ile oluşturulan HTTP isteklerini yöneten bir sunucu oluşturuluyor.

**WebSocket sunucusu:** Socket.IO kullanılarak WebSocket bağlantıları kuruluyor.

origin: '\*' ile herhangi bir kaynaktan bağlantıya izin veriliyor (bu, geliştirme sırasında faydalı ama üretimde dikkatli kullanılmalı).

1. **MySQL Veritabanı Bağlantısı**

**mysql.createPool:** Veritabanı bağlantı havuzu oluşturuyor, bu da birden fazla eşzamanlı sorgu için daha verimli hale geliyor.

Çevresel değişkenler (process.env) kullanılarak veritabanı bilgileri dışarıdan sağlanıyor.

1. **Sabitler ve Şifreleme Anahtarı**

**JWT\_SECRET:** JSON Web Token'ları imzalamak ve doğrulamak için kullanılan gizli anahtar.

**algorithm:** AES şifreleme algoritmasının tipi (aes-256-cbc).

**key:** Şifreleme için kullanılan anahtar, çevresel değişkenden alınarak Buffer formatına dönüştürülüyor.

1. **Middleware (Ara Katmanlar)**

**CORS:** API'ye farklı kaynaklardan erişime izin veriyor**.**

**express.static('public'):** public klasöründeki dosyaları statik olarak sunuyor.

**express.json():** Gelen isteklerde JSON gövdelerini otomatik olarak ayrıştırıyor.

1. **JWT Token Doğrulama (Middleware)**

**Nasıl Çalışır?**

1. **Authorization Header'ı Kontrol Ediliyor:**
   * İstemci tarafından gönderilen JWT Authorization başlığıyla alınır.
   * Format: Bearer <JWT> şeklindedir. split(" ")[1] ile "Bearer" kısmı çıkarılır ve token alınır.
2. **Token'ın Var Olup Olmadığını Kontrol Eder:**
   * Eğer token yoksa, 401 Unauthorized döndürülür.
3. **Token Doğrulaması:**
   * jwt.verify kullanılarak token, JWT\_SECRET ile çözülür.
   * Eğer token geçersizse veya süresi dolmuşsa, 403 Forbidden hatası döner.
   * Token geçerliyse, kullanıcı bilgileri (user) çözülür ve req.user içine eklenir.

**Ne İşe Yarar?**

* Bu middleware, yalnızca geçerli token’a sahip kullanıcıların belirli rotalara erişimini sağlar.
* Örneğin, bir kullanıcı profil bilgilerini görmek veya mesajlaşmak istiyorsa, öncelikle token doğrulamasından geçer.
  1. **JWT Token Geçerliliği Kontrolü (Helper Function)**

**Nasıl Çalışır?**

* Token'ı doğrular ve sonucu true veya false olarak döndürür.
* Bu, basit bir kontrol gerektiren durumlarda kullanılabilir.

**Ne İşe Yarar?**

* Örneğin, bir kullanıcının oturumunun geçerli olup olmadığını hızlıca kontrol etmek istediğinizde bu fonksiyonu çağırabilirsiniz.

1. **AES ile Şifreleme**

**Nasıl Çalışır?**

1. **IV (Initialization Vector):**
   * Şifreleme işlemini daha güvenli hale getirmek için rastgele bir başlangıç değeri kullanılır.
   * Her şifreleme işleminde yeni bir IV oluşturulur.
2. **Şifreleme İşlemi:**
   * Girilen metin, utf8 formatından alınarak hex formatına dönüştürülür ve şifrelenir.
   * Sonuç, IV:şifrelenmiş metin formatında döndürülür.

**Ne İşe Yarar?**

* Kullanıcı bilgileri, mesajlar veya hassas veriler gibi gizli bilgileri şifrelemek için kullanılır.

**7.2. AES ile Şifre Çözme**

**Nasıl Çalışır?**

1. encryptedText iki parçaya ayrılır: IV ve şifrelenmiş metin.
2. IV, şifreleme sırasında kullanılan aynı başlangıç değeri olarak kullanılır.
3. crypto.createDecipheriv fonksiyonu ile şifrelenmiş metin çözülür.

**Ne İşe Yarar?**

* Daha önce şifrelenmiş bir metni çözerek orijinal haline geri döndürür.

1. **const onlineUsers = new Map();**



Map nesnesi oluşturarak çevrimiçi kullanıcıların takibini yapmak için bir veri yapısı oluşturuyor.

* 1. **io.use: Socket.IO Middleware ile Token**

**Doğrulama**

Bu middleware, güvenlik amacıyla soket bağlantısı sırasında istemci tarafından sağlanan bir JWT token'ı doğruluyor.

**Nasıl Çalışır:**

1. **Token Alımı:**
   * socket.handshake.query.token ile istemciden gelen token alınıyor.
2. **Doğrulama:**
   * jwt.verify fonksiyonu ile token'ın geçerli olup olmadığı kontrol ediliyor.
   * Geçerli bir token ise, kullanıcının bilgileri (username, userId) socket nesnesine ekleniyor.
3. **Hata Durumu:**
   * Eğer token geçerli değilse veya eksikse, bağlantı kesiliyor.

**Kodun Önemi:**

* Her soket bağlantısının güvenli olmasını sağlar.
* Sunucu, kimliği doğrulanmamış istemcilerden gelen bağlantıları reddeder.

1. **io.on('connection'): Soket Olayları**

Sunucunun bağlantı kurulduğunda ve diğer olaylar meydana geldiğinde nasıl davranacağını belirler.

**a. Bağlantı (connection):**

* Bir istemci bağlandığında, kullanıcının kimliği (username, socket.id) loglanır.
* Kullanıcı onlineUsers gibi bir yapıya eklenebilir (şu an kullanılmıyor ama genişletilebilir).

**b. Oda Yönetimi:**

1. **Odaya Katılma (joinRoom):**
   * Kullanıcı, belirtilen bir chatId odasına katılır.
   * socket.join(chatId) fonksiyonu odanın oluşturulmasını veya kullanıcıyı mevcut odaya eklemeyi sağlar.
2. **Odadan Ayrılma (leaveRoom):**
   * Kullanıcı, belirtilen bir chatId odasından ayrılır.
   * socket.leave(chatId) ile kullanıcı odadan çıkarılır**.**

**c. Mesaj Gönderme (chatMessage):**

* İşlev: Kullanıcı bir mesaj gönderdiğinde şu işlemleri yapar:
  1. Mesajın içeriği (content) şifrelenir.
  2. Mesaj veritabanına (messages tablosuna) kaydedilir.
  3. Mesaj, odaya (chatId) bağlı tüm istemcilere yayınlanır (io.to(chatId).emit).

**d. Bağlantının Kesilmesi (disconnect):**

* Kullanıcı bağlantıyı kestiğinde, kullanıcının bilgileri loglanır.
* İsteğe bağlı olarak kullanıcı onlineUsers gibi bir yapıdan çıkarılabilir.

**Endpointler**

Kullanıcıların sisteme kaydolmasını, oturum açmasını, sohbetlere katılmasını ve yönetimsel görevler gerçekleştirmesi işlevlerini içerir.

Mesajlaşma platformunun kullanıcı yönetimi, grup dinamikleri ve sohbet mesajlaşma özelliklerini desteklemek için kapsamlı bir yapı sunar. Kullanıcı ve admin rollerine uygun yetkilendirme ve kontrol işlemleri içerir.

**Kayıt ve Giriş**

#### **/register**

* **İşlev**: Kullanıcı kaydı oluşturur.
* Kullanıcı adı, şifre ve e-posta bilgilerini alır.
* Şifreyi hashleyerek veritabanına kaydeder.
* Kullanıcıya bir varsayılan profil oluşturur.
* Kullanıcıyı genel bir yayın sohbetine (broadcast chat) ekler.

#### **/login**

* **İşlev:** Kullanıcının giriş yapmasını sağlar.
* Kullanıcı adı ve şifre doğrulaması yapar.
* Şifre doğruysa, bir JWT (JSON Web Token) oluşturur ve kullanıcıya döner.
* Kullanıcının rol bilgilerini JWT içinde iletir.

### **Sohbet ve Mesaj Yönetimi**

#### **/admin/chats**

* **İşlev:** Adminin tüm yayın ve grup sohbetlerini (broadcast, multicast) görmesini sağlar.
* Özel (private) sohbetler listede yer almaz.

#### **/user/:userId/chats**

* **İşlev:** Belirtilen kullanıcı için katıldığı sohbetleri listeler.
* Her sohbet için:
  + Sohbet adı, türü, oluşturulma tarihi.
  + Son mesajın içeriği ve zamanı (şifre çözme işlemi yapılırsa).
* Zaman sırasına göre sıralama yapar.

#### **/chat/:chatId/messages**

* **İşlev:** Belirtilen sohbet için tüm mesajları getirir.
* Mesaj içeriği şifrelenmişse, çözülür ve kullanıcıya döner.
* Mesaj sırasını zaman sırasına göre listeler.

#### **/chat/private**

* **İşlev:** İki kullanıcı arasında özel bir sohbet (private chat) oluşturur.
* Eğer zaten bir sohbet mevcutsa, mevcut sohbet bilgilerini döner.
* Yeni sohbet oluşturulursa, her iki kullanıcıyı da bu sohbete ekler.

### 

### **Grup Sohbetleri**

#### **/search-group**

* İşlev: Grup adını alarak grubu arar.
* Eğer grup varsa, grup bilgilerini döner.
* Grup bulunamazsa, 404 hatası döner.

#### **/create-group**

* İşlev: Yeni bir grup oluşturur.
* Grup adını ve grubu oluşturan kullanıcıyı alır.
* Grup oluşturulduktan sonra bir sohbet (multicast) ve grup üyeliği bilgisi ekler.

#### **/join-group**

* İşlev: Bir kullanıcıyı mevcut bir gruba ekler.
* Eğer kullanıcı zaten grup üyesiyse, hata döner.
* Kullanıcıyı hem grup hem de sohbet üyeleri arasına ekler.

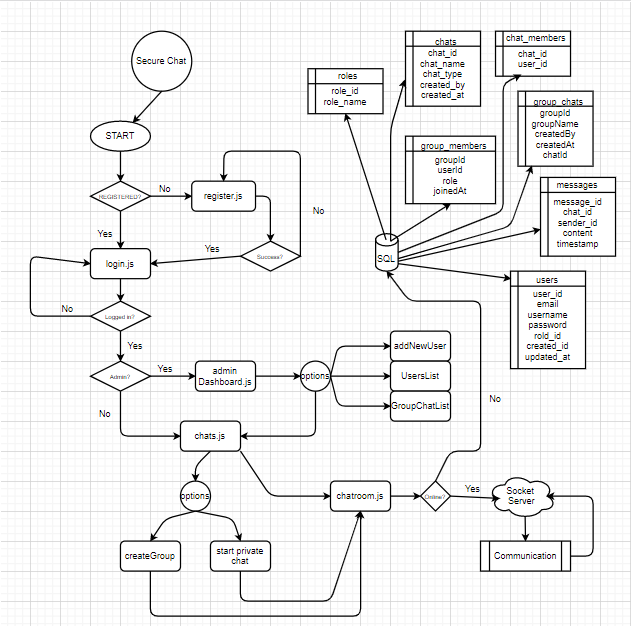
#### **/leave-group**

* İşlev: Bir kullanıcının gruptan ayrılmasını sağlar.
* Kullanıcıyı hem grup hem de sohbet üyelerinden siler.

#### **/ban-user**

* İşlev: Bir gruptaki kullanıcıyı yasaklamak için admin yetkisi gerektirir.
* Admin kontrolü yapılır, ardından belirtilen kullanıcı gruptan ve sohbet üyelerinden silinir.

**UML Diagram**

****