



# YMH345

# Bilgisayar Ağları Dönem Projesi

## ChatWork

## Proje Raporu

Barış Güngör - 22290692
Umut Çağatay Tapur - 22290636
Hüseyin Sarıbuğa - 22290202
Elif Nur Tekay - 23291762

# İçindekiler

# 1. Proje Tanıtımı

Bu projede, kullanıcılar arasında kısa mesajlaşma ve sesli aramaları destekleyen gerçek zamanlı bir iletişim platformu geliştirdik. Platformumuz, kullanıcı kaydı, kısa mesajlaşma (hem bireysel hem de grup) ve iki katılımcı arasında sesli aramalar gibi işlevler sunmaktadır. Çalışmalarımız sırasında, modern ağ uygulamalarının temel bileşenleri olan ağ protokolleri, gerçek zamanlı iletişim, oturum yönetimi, güvenlik ve performans optimizasyonu gibi konuları başarıyla ele aldık ve uyguladık.

# 2. Amaç ve Hedefler

## 2.1. Amaç

Bu projenin temel amacı, kullanıcıların gerçek zamanlı olarak metin mesajları göndermesini, birebir sesli aramalar yapmasını, arkadaş ekleme ve engelleme gibi sosyal etkileşim özelliklerinden faydalanmasını sağlayan güvenli ve yüksek performanslı bir iletişim platformu geliştirmektir. Modern ağ uygulamalarının gerektirdiği protokoller, oturum yönetimi ve performans optimizasyonları gibi teknik bileşenleri anlamayı ve uygulamayı hedefleyen bir altyapı oluşturulması amaçlanmaktadır.

## 2.2. Hedefler

### 2.2.1. Güvenli Kullanıcı Kimlik Doğrulama:

- Her kullanıcının benzersiz bir kimlik ile platformda tanımlanmasını sağlamak.
- Token tabanlı veya oturum tabanlı kimlik doğrulama yöntemlerini kullanarak, kullanıcı hesaplarının güvenliğini artırmak.

### 2.2.2. Gerçek Zamanlı Mesajlaşma Sistemi:

- Düşük gecikme ile bireysel ve grup mesajlaşmalarını destekleyen bir altyapı oluşturmak.
- Mesajların iletim durumlarını (gönderildi, teslim edildi, okundu) kullanıcıya bildirerek kullanıcı deneyimini geliştirmek.

### 2.2.3. Sesli Arama Sistemi:

- WebRTC gibi teknolojilerle birebir sesli iletişim imkanı sağlayan bir sistem geliştirmek.

### 2.2.4. Arkadaş Ekleme ve Kişi Engelleme:

- Kullanıcıların platform üzerinden diğer kullanıcıları arkadaş olarak eklemesini ve bu kişilerle iletişim kurmasını sağlamak.
- İstenmeyen kullanıcıları engelleyerek, daha güvenli ve kişiselleştirilmiş bir iletişim ortamı oluşturmak.

- Engelleme ve arkadaş listesi yönetimi için kullanıcı dostu bir arayüz geliştirmek.

#### 2.2.5. Mesaj ve Arama Kayıtlarının Yönetimi:

- Kullanıcıların mesaj geçmişlerini ve arama kayıtlarını güvenli bir şekilde saklamak ve gerektiğinde bu verilere erişim sağlamak.
- Veri saklama ve geri çağırma süreçlerini optimize etmek.

#### 2.2.6. Performans ve Ölçeklenebilirlik:

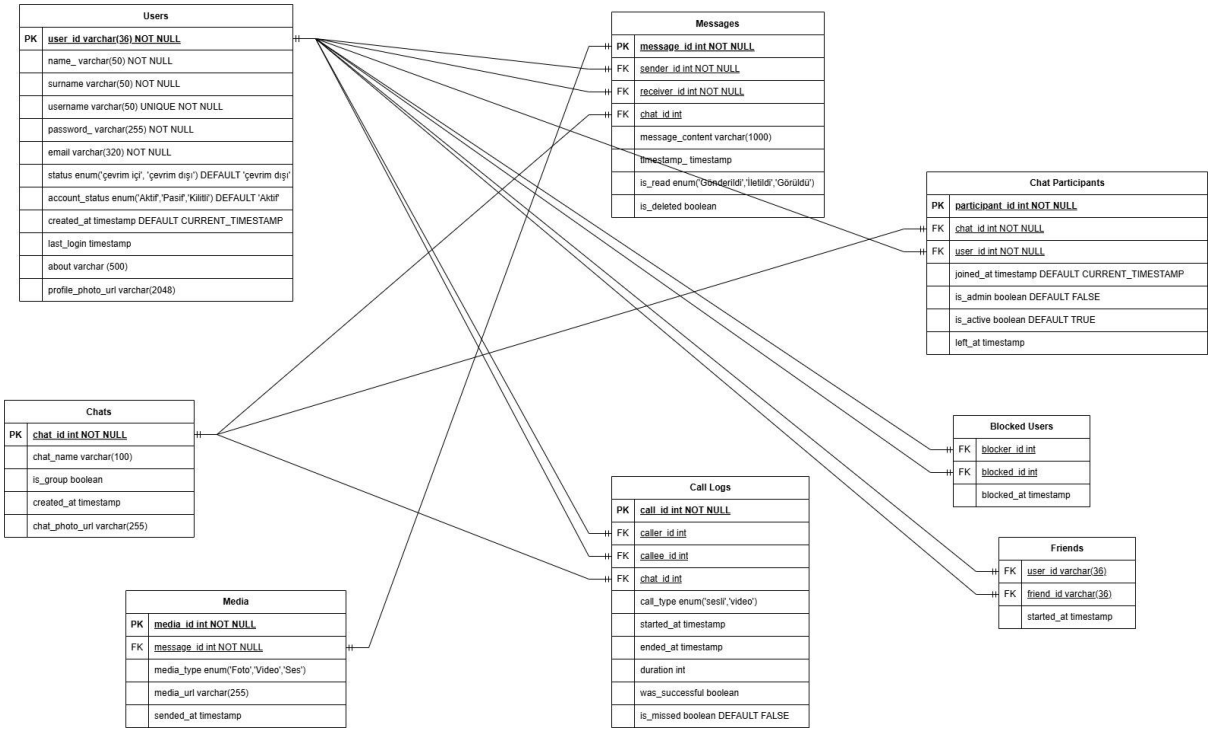
- Çoklu kullanıcı desteği ile yüksek performanslı bir sistem tasarlamak ve ölçeklenebilir bir mimari geliştirmek.
- Gerçek zamanlı iletişimde minimum gecikme süresi sağlayarak kullanıcı memnuniyetini artırmak.

## 3. Kullanılan Teknolojiler

Proje geliştirme sürecinde kullandığımız teknolojiler aşağıda bulunmaktadır:

- **Node.js**: Sunucu tarafında kullanılan JavaScript tabanlı bir çalışma zamanı ortamı.
- **Express.js**: Hafif ve esnek bir web uygulama framework'ü. API geliştirme ve yönlendirme işlemlerinde kullanıldı.
- **bcrypt**: Kullanıcı şifrelerini güvenli bir şekilde hashlemek için kullanıldı.
- **jsonwebtoken (JWT)**: Kullanıcı kimlik doğrulama ve yetkilendirme işlemleri için kullanıldı.
- **mysql2**: Veritabanı bağlantısı için modern ve yüksek performanslı bir MySQL kütüphanesi.
- **Socket.IO**: Gerçek zamanlı mesajlaşma ve çift yönlü iletişim sağlamak için kullanılan kütüphane. WebSocket ve diğer teknolojileri kullanarak tarayıcılar ve sunucular arasında iki yönlü iletişim sağlar.
- **uuid**: Kullanıcılar için benzersiz kimlikler oluşturmak amacıyla kullanılan kütüphane.
- **dotenv**: Projenin yapılandırma bilgilerini `.env` dosyasından yüklemek için kullanıldı.
- **cookie-parser**: HTTP isteklerindeki çerezleri ayrıştırmak için kullanıldı.
- **cors**: Cross-Origin Resource Sharing (CORS) politikalarını yönetmek için kullanıldı.
- **http-errors**: HTTP hata durumlarının oluşturulmasını kolaylaştırmak için kullanıldı.
- **morgan**: HTTP isteklerini kaydetmek için kullanılan bir logger.
- **WebRTC**: İki kullanıcı arasında P2P (peer-to-peer) sesli arama işlemlerini gerçekleştirmek için kullanıldı.
- **HTTPS (SSL/TLS)**: Kullanıcı oturum açma bilgileri ve diğer hassas verilerin güvenli iletimi sağlandı.

# 4. Veri Tabanı Şeması



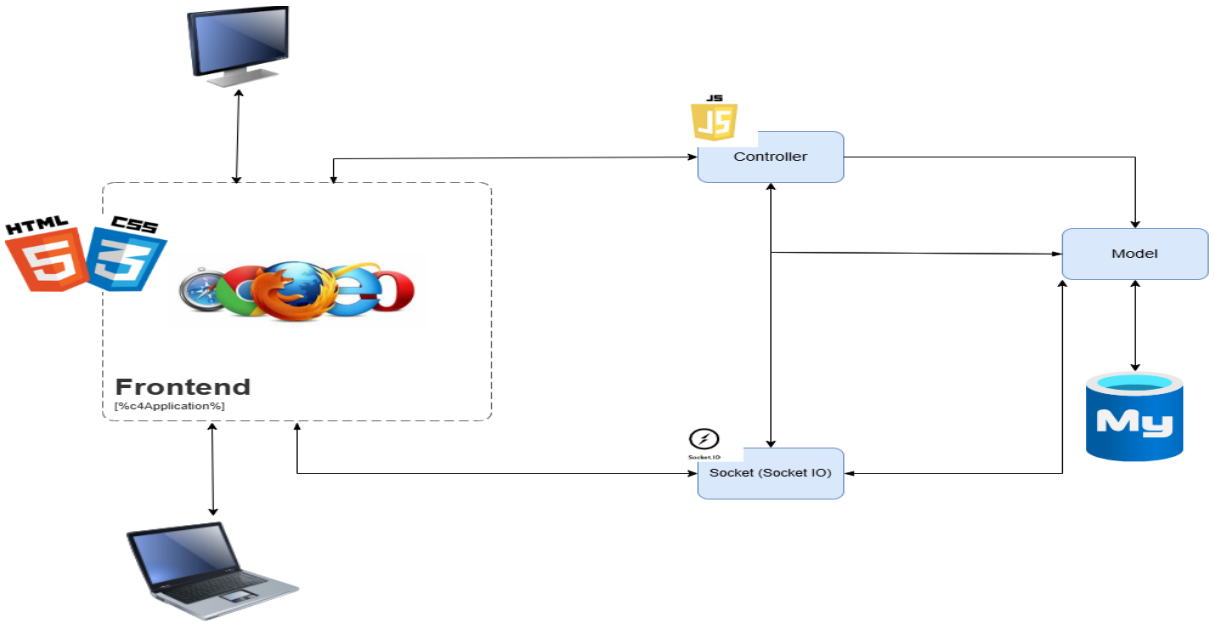

# 5. Veri Tabanı Kodları

```
CREATE TABLE users(
    user_id VARCHAR(36) PRIMARY KEY,
    name_ VARCHAR(50) NOT NULL,
    surname VARCHAR(50) NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_ VARCHAR(255) NOT NULL,
    email VARCHAR(320) NOT NULL,
    phone_number VARCHAR(15),
    status_ ENUM('çevrim içi','çevrim dışı') DEFAULT 'çevrim dışı',
    account_status ENUM('Aktif','Pasif','Kilitli') DEFAULT 'Aktif',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP,
    about VARCHAR(500),
    profile_photo_url VARCHAR(512)
);

CREATE TABLE chats(
    chat_id VARCHAR(36) PRIMARY KEY,
    chat_name VARCHAR(100),
    is_group BOOLEAN,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    chat_photo_url VARCHAR(512)
);
```

```
CREATE TABLE messages(
   message_id VARCHAR(36) PRIMARY KEY,
   sender_id VARCHAR(36) NOT NULL,
   receiver_id VARCHAR(36), -- INDIVIDUAL CHATS
   chat_id VARCHAR(36) NOT NULL, -- GROUP CHATS
   message_content TEXT NOT NULL,
   timestamp_ TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   is_read ENUM("Gönderildi","İletildi","Görüldü") DEFAULT "Gönderildi",
   status_ ENUM("Orijinal","Düzenlendi", "Silindi") DEFAULT "Orijinal",
   updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
   FOREIGN KEY (sender_id) REFERENCES users(user_id),
   FOREIGN KEY (receiver_id) REFERENCES users(user_id),
   FOREIGN KEY (chat_id) REFERENCES chats(chat_id)
);

CREATE TABLE chat_members(
   member_id VARCHAR(36) PRIMARY KEY,
   chat_id VARCHAR(36) NOT NULL,
   user_id VARCHAR(36) NOT NULL,
   joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   is_admin BOOLEAN DEFAULT FALSE,
   left_at TIMESTAMP,
   is_active BOOLEAN DEFAULT TRUE,
   UNIQUE (chat_id, user_id),
   FOREIGN KEY (chat_id) REFERENCES chats(chat_id),
   FOREIGN KEY (user_id) REFERENCES users(user_id)
);

CREATE TABLE medias(
   media_id VARCHAR(36) PRIMARY KEY,
   message_id VARCHAR(36) NOT NULL,
   media_type ENUM("Fotoğraf","Video","Ses","Bağlantı", "Kişi", "Konum","Belge",
"Anket"),
   media_url VARCHAR(512) NOT NULL,
   sended_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (message_id) REFERENCES messages(message_id)
);

CREATE TABLE blocked_users(
   blocker_id VARCHAR(36),
   blocked_id VARCHAR(36),
   blocked_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   PRIMARY KEY (blocker_id, blocked_id),
   FOREIGN KEY (blocker_id) REFERENCES users(user_id),
   FOREIGN KEY (blocked_id) REFERENCES users(user_id)
);
```

```
CREATE TABLE friends (
    user1_id VARCHAR(36),
    user2_id VARCHAR(36),
    started_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status_ ENUM('Beklemede', 'Kabul Edildi', 'Reddedildi') DEFAULT 'Beklemede',
    PRIMARY KEY (user1_id, user2_id),
    FOREIGN KEY (user1_id) REFERENCES users(user_id) ON DELETE
CASCADE,
    FOREIGN KEY (user2_id) REFERENCES users(user_id) ON DELETE CASCADE
);

CREATE TABLE call_logs (
    call_id VARCHAR(36) PRIMARY KEY,
    caller_id VARCHAR(36) NOT NULL,
    callee_id VARCHAR(36),
    chat_id VARCHAR(36) NOT NULL,
    call_type ENUM("Sesli","Görüntülü") NOT NULL,
    started_at TIMESTAMP NOT NULL,
    ended_at TIMESTAMP NOT NULL,
    duration TIME,
    was_successful BOOLEAN DEFAULT TRUE,
    is_missed BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (caller_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (callee_id) REFERENCES users(user_id) ON DELETE
CASCADE,
    FOREIGN KEY (chat_id) REFERENCES chats(chat_id) ON DELETE CASCADE
);
```

## 6. Sistem Mimarisi



**Client (Frontend)**: Mesajlaşma veya diğer olayları tetikler.
**App.js**: Socket.IO bağlaması ve temel kurulum burada yapılır.
**Routes**: HTTP API istekleri için yönlendirme sağlar.
**Controllers**:

- HTTP isteklerini işler.

**Sockets**:

- Socket olaylarını dinler ve kontrol eder.
- Doğrudan Models katmanıyla iletişim kurar.

**Models**:

- Veritabanıyla doğrudan etkileşim sağlar.
- Socket olaylarının veri tabanına işlenmesini sağlar.

**Database**: MySQL, mesajların veya kullanıcı durumlarının saklandığı yerdir.

Oluşturduğumuz yapı olay tabanlı mimariye uygundur.

# 7. Backend Fonksiyonları

## 7.1. Models Klasörü

### 7.1.1. Users Dosyası

**findByUsername(username)**: Veri tabanında verilen `username` değerine sahip kullanıcıyı arar ve o kullanıcının bilgilerini döner.

**getUserProfileDetails(userId)**: Belirtilen `userId` değerine sahip kullanıcının profil bilgilerini (ad-soyad, hakkında, e-posta) getirir.

**getBlockedUsers(userId)**: Belirtilen `userId` tarafından engellenen kullanıcıların ad-soyad, kullanıcı adı ve e-posta bilgilerini getirir.

**findMatchedUsers(searchValue, userId)**: Belirtilen `userId` dışında kalan kullanıcılar arasında bir arama yapar. `searchValue`'ya göre ad, soyad veya kullanıcı adı eşleşmelerine bakar ve eşleşen kullanıcıları döner.

**getUsernameById(userId)**: Verilen `userId` değerine karşılık gelen kullanıcının kullanıcı adını (`username`) getirir.

**getReceiverProfileDetail(username)**: Verilen `username`'e sahip kullanıcının ad, soyad, kullanıcı adı, e-posta ve hakkında bilgilerini getirir.

**getChatMembers(chatId)**: Belirtilen `chatId`'ye ait sohbetin üyelerini getirir. Üyelerin tam adlarını (`full_name`) ve sohbet içindeki admin olup olmadıklarını döner.

**addLastLogin(userId)**: Belirtilen `userId`'li kullanıcının son giriş tarihini günceller (`last_login`). Kullanıcının durumunu "çevrim içi" olarak değiştirir.

**setOnlineStatus(userId)**: Belirtilen `userId`'li kullanıcının son giriş tarihini günceller (`last_login`). Kullanıcının durumunu "çevrim dışı" olarak değiştirir.

### 7.1.2. Message Dosyası

**loadMessages(senderId, receiverUsername)**: Belirtilen `senderId` ve `receiverUsername` ile ilişkili mesajları yükler. Kullanıcıların bire bir sohbet mesajlarını getirir.

**getLastLogin(receiverUsername)**: Belirtilen kullanıcı adının (`receiverUsername`) son giriş tarihini döner.

**getStatus(receiverUsername)**: Belirtilen kullanıcı adının (`receiverUsername`) çevrim içi veya çevrim dışı gibi durum bilgisini döner.

**loadGroupMessages(chatName, createdAt)**: Verilen grup adı (`chatName`) ve oluşturulma tarihine (`createdAt`) göre grup sohbet mesajlarını getirir.

**getChatId(senderId, receiverUsername)**: Belirtilen gönderici (`senderId`) ve alıcı kullanıcı adı (`receiverUsername`) arasındaki bire bir sohbetin `chat_id` değerini döner.

**getChatIdForGroup(groupName, createdAt)**: Verilen grup adı (`groupName`) ve oluşturulma tarihine (`createdAt`) göre ilgili grup sohbetinin `chat_id` değerini döner.

**getReceiverId(chat_id, sender_id)**: Belirtilen sohbet kimliğine (`chat_id`) ve gönderen kullanıcı kimliğine (`sender_id`) göre alıcının kullanıcı kimliğini döner.

**saveMessagesToDB(messageData)**: Mesaj bilgilerini (`messageData`) veritabanına kaydeder. Mesajla ilgili `message_id`, gönderici ve alıcı kimlikleri, içerik ve diğer metadata bilgiler bu işlemde kullanılır.

**setMessageStatus(messageId, is_read)**: Belirtilen mesajın durumunu günceller. `is_read` değeri mesajın okunup okunmadığını belirtir.

### 7.1.3. Friend Dosyası

loadFriend(userId): Verilen kullanıcı ID'sine göre arkadaş listesini yükler. Kullanıcının arkadaşlarının ID'sini, kullanıcı adlarını, isimlerini, soyisimlerini, hakkında bilgilerini, arkadaşlık başlangıç tarihini ve durumunu döner.

getFriendIdFromUsername(username): Verilen kullanıcı adından ilgili kullanıcının ID'sini döner.

insertFriendRequest(currentUserId, otherUserId): İki kullanıcı arasında bir arkadaşlık isteği oluşturur. Mevcut kullanıcının kullanıcı adını döner.

loadFriendRequests(userId): Kullanıcının aldığı bekleyen arkadaşlık isteklerini yükler. İstek gönderen kullanıcıların adlarını ve isteğin gönderilme tarihini döner.

checkFriendRequest(currentUserId, otherUserId): İki kullanıcı arasında bir arkadaşlık isteği olup olmadığını kontrol eder. Var olan arkadaşlık isteğini döner.

updateFriendRequest(currentUserId, otherUserId): İki kullanıcı arasındaki arkadaşlık isteğini "Beklemede" durumuna günceller.

getCurrentUsername(userId): Kullanıcı ID'sine göre mevcut kullanıcının kullanıcı adını döner.

updateFriendRequestRejected(currentUserId, otherUserId): İki kullanıcı arasındaki arkadaşlık isteğinin durumunu "Reddedildi" olarak günceller.

updateFriendRequestAccepted(currentUserId, otherUserId): İki kullanıcı arasındaki arkadaşlık isteğinin durumunu "Kabul Edildi" olarak günceller.

createIndividualChat(chatId): Verilen chat ID'siyle bireysel bir sohbet oluşturur.

insertChatMembers(memberId, chatId, userId): Verilen kullanıcı ID'sini, üye ID'si ve sohbet ID'siyle bir sohbete ekler.

### 7.1.4. Chat Dosyası

loadChat(userId): Kullanıcının aktif sohbetlerini (grup veya birebir) yükler, sohbet adı, son mesaj, mesaj zamanı, okunma durumu gibi bilgileri getirir.

**fetchGroupNames(userId)**: Kullanıcının üyesi olduğu grup sohbetlerinin isimlerini döndürür.

**createChat(chatId, groupName)**: Yeni bir grup sohbeti oluşturur ve grubu veritabanına kaydeder.

#### 7.1.5. Call Dosyası

**fetchCall(userId)**:

- Verilen kullanıcı kimliği (userId) için arama geçmişini getirir.
- Gelen/giden çağrıların detaylarını döndürür:
    - Çağrı ID'si, arayan/arayan kullanıcıların kimlikleri.
    - Diğer kullanıcının adı ve soyadı.
    - Çağrının türü, başlangıç ve bitiş zamanı, süresi.
    - Başarılı olup olmadığı ve cevapsız olup olmadığı gibi bilgileri içerir.

### 7.2. Controllers Klasörü

#### 7.2.1. AuthControllers Dosyası

**login(req, res)**: Kullanıcı adı ve şifre kontrolü yapar, doğruysa JWT token oluşturur ve cookie olarak döner.

**protectedRoute(req, res)**: Gelen token'ı doğrular, geçerli ise erişim izni verir.

**getUserId(req, res)**: Cookie'deki token'ı kontrol eder, geçerliyse kullanıcı ID'sini döner.

#### 7.2.2. UserControllers Dosyası

**registerUser(name, surname, username, password, email, about)**: Kullanıcıyı kaydeder. Şifreyi hash'leyerek benzersiz bir user_id ile birlikte veritabanına kaydeder.

**loadProfileDetails(req, res)**: Kullanıcının profil bilgilerini cookie'deki token'dan alınan user_id ile yükler ve döner.
**getBlockedUsers(req, res)**: Kullanıcının engellediği kullanıcıların listesini yükler ve döner.

**findMatchedUsers(req, res)**: Kullanıcı adı araması yapar ve arama kriterlerine uyan kullanıcıları döner. Arama sonuçları varsa `exists: true` döner.

**getReceiverProfile(req, res)**: Gönderilen kullanıcı adına (`username`) göre alıcının profil bilgilerini döner.

**getGroupMembers(req, res)**: Grup sohbetindeki üyeleri, tam adları (`fullName`) ve yönetici olup olmadıklarına (`isAdmin`) dair bilgileriyle birlikte döner.

### 7.2.3. MessageControllers Dosyası

**sendMessage(req, res)**:

- Kullanıcıdan gelen verileri alır ve cookie'deki token'dan gönderici ID'sini belirler.
- Mesajı oluşturur ve grup ya da bireysel sohbete uygun olarak veritabanına kaydeder.
- Mesaj detaylarını başarılı şekilde kaydedilirse döner, hata durumunda uygun bir hata mesajı verir.

**getMessagesFromChatData(req, res)**:

- Kullanıcının mesaj geçmişini bir grup sohbetinden veya bireysel bir sohbetten yükler.
- Grup sohbeti için mesajları, bireysel sohbet için mesajları ve alıcının durum bilgilerini (son giriş ve çevrim içi durumu) döner.
- Hata durumunda veri tabanı hatasını bildirir.

### 7.2.4. FriendControllers Dosyası

**loadFriendRequests(req, res)**: Cookie'deki token'dan kullanıcı ID'sini alır ve kullanıcının arkadaşlık isteklerini veri tabanından yükler. Başarılıysa istekleri döner, hata olursa uygun hata mesajı verir.

### 7.2.5. ChatControllers Dosyası

**getUserChats(req, res)**: Cookie'den auth_token alır ve token'dan kullanıcı user_id'sini çözer. Kullanıcının Chat modeline ait sohbetlerini veritabanından yükleyip, JSON formatında döner.

**getUserFriends(req, res)**: Cookie'den auth_token alır ve token'dan kullanıcı user_id'yi çözer. Kullanıcının Friend modeline ait arkadaşlarını veritabanından yükleyip, JSON formatında döner.

### 7.2.6. Call Dosyası

**getBlockedUsers(req, res)**: Kullanıcının engellediği kullanıcıların listesini getirir ve JSON formatında döner.

**findMatchedUsers(req, res)**: Kullanıcı adıyla arama yapar ve sonuçları kriterlere uygun şekilde döner; sonuç varsa exists: true.

**getReceiverProfile(req, res)**: Belirtilen kullanıcı adına göre alıcının profil bilgilerini döner.

**getGroupMembers(req, res)**: Grup sohbetindeki üyelerin adlarını ve yönetici bilgilerini döner.

## 7.3. Sockets Klasörü

### 7.3.1. SocketHandler Dosyası

**initializeSocket(io)**: WebSocket sunucusunu başlatır, yeni bağlantıları ve olayları (message, friend, chat, WebRTC gibi) yönetir.

**onConnection(socket):** Yeni bir bağlantıda token doğrular, kullanıcıyı online listeye ekler, gruplara ve odalara bağlar. Doğrulama başarısızsa bağlantıyı keser.

**onDisconnect(socket)**: Kullanıcı bağlantısı koparsa, durumu çevrim dışı yapar ve online listeden çıkarır.

**onLogout(socket)**: Kullanıcı çıkış yaparsa JWT doğrular, sunucuya çıkış isteği gönderir ve istemciye bilgi verir.

**createRoomSlug(groupName)**: Grup adını temizleyip küçük harflerle ve özel karakterlerden arındırılmış bir oda adı oluşturur.

**webRTCSignalHandlers (socket)**: Gerçek zamanlı sesli arama için sinyalleşme işlemlerini çağıran kod parçasıdır.

### 7.3.2. MessageSocket Dosyası

**onMessageReceived(socket)**: Alınan mesajların durumunu (Görüldü veya İletildi) günceller ve gönderen kullanıcıya bildirir.

**onSendMessage(socket)**: Yeni bir mesajı hedef kullanıcıya iletir ve mesaj içeriğiyle birlikte kullanıcıya bilgi gönderir.

**onBroadcastMessage(socket)**: Grup mesajlarını tüm grup üyelerine iletir ve gönderenin bilgilerini paylaşır.

**onTypingStatus(socket)**: Kullanıcının yazma durumunu (`yazıyor`) diğer katılımcılara bildirir.

**messageSocketHandlers(socket)**: Tüm mesajlaşma ile ilgili event handler'larını (`onMessageReceived`, `onSendMessage`, vb.) socket'e bağlar.

**getGroupSlug(groupName)**: Grup adını temizleyip özel karakterlerden arındırılmış bir oda adı oluşturur.

### 7.3.3. FriendSocket Dosyası

**onFriendRequest(socket)**: Yeni arkadaşlık isteği alındığında mevcut istekleri kontrol eder. İstek zaten reddedilmişse, yeniden gönderir. Yoksa yeni bir istek oluşturur ve alıcıya bildirir.

**friend_request_rejected**: Arkadaşlık isteği reddedildiğinde durumu günceller.

**friend_request_accepted**: Arkadaşlık isteği kabul edildiğinde, bireysel bir sohbet oluşturur ve her iki kullanıcıyı da sohbete ekler.

**friendSocketHandlers(socket)**: Arkadaşlık işlemleriyle ilgili tüm event handler'ları socket'e bağlar.

### 7.3.4. ChatSocket Dosyası

**chatSocketHandlers(socket)**: Chat ile ilgili tüm socket eventlerini yönetir.

**create_chat**:

- Yeni bir grup sohbeti oluşturur.
- Sohbeti başlatan kullanıcıyı ve arkadaş listesindeki kullanıcıları sohbete ekler.
- Kullanıcıları ilgili odaya bağlar ve gruba katılmalarını sağlar.

**joinRoom**: Kullanıcıyı belirtilen sohbet odasına bağlar ve katıldığını onaylar.

**createRoomSlug**: Grup adı için URL-dostu bir slug oluşturur. Türkçe karakterleri değiştirir ve özel karakterleri kaldırır.

### 7.3.5. OnlineUsers Dosyası

**addOnlineUser(userId, socketId)**: Bir kullanıcı çevrimiçi olduğunda, userId ve socketId bilgisini onlineUsers haritasına ekler.

**removeOnlineUser(socketId)**: Verilen socketId'ye sahip kullanıcıyı onlineUsers haritasından kaldırır. Silinen kullanıcının userId'sini döndürür (isteğe bağlı).

**getSocketId(userId)**: Belirtilen userId'ye karşılık gelen socketId'yi döndürür.

**getUserId(socketId)**: Belirtilen socketId'ye karşılık gelen userId'yi döndürür.

## 7.4. Middleware Klasörü

### 7.4.1. Auth Dosyası

**authenticateToken**:

- auth_token çerezleri alır ve doğrular.
- geçerli değilse 403 hatası döner.
- geçerliyse kullanıcı bilgilerini req.user'a ekler ve devam eder.

## 7.5. SocketListener Dosyası

Bu dosya, istemci tarafında:

- Gerçek zamanlı mesaj alıp göndermeyi sağlar.
- Mesaj durumlarını güncellemeyi sağlar.
- Grup mesajlarını yönetmeyi sağlar.
- Arkadaşlık isteklerini işleme almayı sağlar.
- WebRTC sayesinde birebir arama sağlanır.
- Oturum ve bağlantı durumlarını kontrol etmeyi sağlar.

## 7.6. Github Linki

chat_work_github_linki