T.C.

TRAKYA ÜNİVERSİTESİ

TUNCA MESLEK YÜKSEKOKULU

TuChatt App

ARDA SEMİH ÇAVDAR - 1237305100

Bilgisayar Programcılığı

DANIŞMAN

ÖĞR. GÖR. İLKER HACIOĞLU

EDİRNE - 2025

[BİTİRME PROJESİ]
[TuChat App]
T.Ü. Tunca MYO
[Bilgisayar Programcılığı]

ÖZET

Bu proje, MERN Stack (MongoDB, Express.js, React, Node.js) ve Socket.io kullanılarak geliştirilen, gerçek zamanlı mesajlaşma sağlayan bir web uygulamasıdır. Projenin amacı, kullanıcıların güvenli ve hızlı bir şekilde anlık mesajlaşmasını sağlamak, çevrim içi durumlarını takip edebilmek ve kullanıcı deneyimini en üst seviyeye çıkarmaktır.

Geliştirme sürecinde öncelikle kimlik doğrulama ve yetkilendirme işlemleri tamamlanmıştır.

Bu kapsamda:

JWT (JSON Web Token) kullanılarak kullanıcı kimlik doğrulama sistemi oluşturulmuş,
Kayıt olma (signup), giriş yapma (login) ve çıkış yapma (logout) işlemleri backend tarafında geliştirilmiş ve ilgili route'lar (yol tanımlamaları) yapılandırılmıştır.
Kullanıcının oturum bilgileri güvenli bir şekilde saklanması amacıyla HTTP-Only Cookie ile JWT token yönetimi sağlanmıştır.
CSRF (Cross-Site Request Forgery) ve XSS (Cross-Site Scripting) saldırılarına karşı güvenlik önlemleri alınarak cookie yapılandırmaları yapılmıştır,
Socket.io entegrasyonu için hazırlıklara başlanmış, WebSocket bağlantısı için temel altyapı oluşturulmuştur.

Gelecek aşamalarda gerçek zamanlı mesajlaşma sisteminin tamamlanması, kullanıcı çevrim içi durumu yönetimi, mesajların veritabanında saklanması ve ön yüz (frontend) geliştirme işlemleri planlanmaktadır.

Yıl	:	2025
Sayfa Sayısı	:	11
Anahtar Kelimeler	:	Gerçek Zamanlı Mesajlaşma, MERN Stack, Socket.io, JWT, Authentication, WebSocket, Web Geliştirme, Güvenlik

İÇİNDEKİLER

BÖLÜM 1

GİRİŞ

Bu proje, MERN Stack (MongoDB, Express.js, React, Node.js) ve Socket.io kullanılarak geliştirilen bir gerçek zamanlı sohbet uygulamasının tasarımını ve implementasyonunu ele almaktadır. Projenin amacı, kullanıcıların hızlı ve güvenli bir şekilde anlık mesajlaşmalarını sağlamak, çevrim içi durumlarını takip edebilmek ve kullanıcı deneyimini optimize etmektir. Ayrıca, uygulama, kullanıcı doğrulama ve güvenlik süreçlerine odaklanarak, güvenli bir mesajlaşma altyapısı sunmayı hedeflemektedir.

Bu proje, günümüzde dijital iletişimin önemli bir aracı haline gelen anlık mesajlaşma uygulamalarının güvenli, hızlı ve kullanıcı dostu bir alternatifini sunmayı amaçlamaktadır. Proje, özellikle Socket.io teknolojisini kullanarak gerçek zamanlı veri iletimi sağlar ve bu alandaki mevcut uygulamaların daha verimli ve güvenli olmasına katkıda bulunmayı hedefler.

Bileşen	Teknoloji / Kütüphane	Açıklama
Frontend	React.js	Kullanıcı arayüzünü oluşturmak için kullanılır.
	Vite	React uygulamasını hızlı ve optimize bir şekilde çalıştırmak için tercih edilir.
	Tailwind CSS	Modern ve esnek bir stil sistemi sağlamak için kullanılır.
	Daisy UI	Kullanıcı dostu bileşenler ve stiller için kullanılır.
	Zustand	Global state yönetimi için kullanılacak hafif bir kütüphane.
	Axios / Fetch API	Backend ile haberleşmek için HTTP isteklerini yönetir.
	Socket.io-client	Gerçek zamanlı veri iletişimi için kullanılır.
Backend	Node.js	Sunucu tarafındaki işlemleri yönetmek için kullanılır.
	Express.js	API oluşturmak için kullanılan minimal ve hızlı bir framework.
	Socket.io	Gerçek zamanlı haberleşme ve mesajlaşma için kullanılır.
	JSON Web Token (JWT)	Kullanıcı kimlik doğrulaması ve yetkilendirme işlemleri için kullanılır.
	bcrypt.js	Kullanıcı şifrelerini güvenli bir şekilde saklamak için kullanılır.
	dotenv	Ortam değişkenlerini yönetmek için kullanılır.
	CORS	API erişim izinlerini kontrol etmek için kullanılır.
	Morgan	API isteklerini loglamak için kullanılır.
Database	MongoDB	NoSQL veritabanı olarak kullanılır.
	Mongoose	MongoDB ile Node.js arasında bağlantıyı yönetmek için kullanılır.
	Redis (Opsiyonel)	Kullanıcı oturumlarını ve önbellekleme işlemlerini hızlandırmak için kullanılabilir.

Projenin Teknik Altyapısı ve Mevcut Durumu

Bu çalışmada, modern web uygulamalarında yaygın olarak kullanılan teknolojiler tercih edilmiştir. Proje geliştirme süreci üç ana bileşen üzerinden ilerlemektedir: Frontend (istemci tarafı), Backend (sunucu tarafı) ve Veritabanı (database).

Frontend tarafında, kullanıcı dostu ve dinamik bir arayüz oluşturulması hedeflenmektedir. Bu kapsamda, React.js tercih edilmiştir. Arayüz tasarımı için Tailwind CSS ve Daisy UI entegrasyonu yapılmış, böylece esnek ve modern bir tasarım sağlanmıştır. Uygulamanın genel durum yönetimi için Zustand kullanılmaktadır. Gerçek zamanlı iletişim gerektiren durumlar için Socket.io-client entegrasyonu planlanmaktadır.

Backend tarafında, temel kimlik doğrulama süreçleri tamamlanmıştır. Node.js ve Express.js kullanılarak sunucu tarafı geliştirilmiş, JWT (JSON Web Token) ile kimlik doğrulama mekanizması oluşturulmuştur. Kullanıcı şifrelerinin güvenli bir şekilde saklanması için bcrypt.js kullanılmıştır. Ortam değişkenleri yönetimi için dotenv, API erişim izinleri için CORS, HTTP isteklerini loglamak için Morgan entegrasyonu sağlanmıştır. Şu ana kadar signup, login ve logout işlemleri tamamlanmış olup, WebSocket entegrasyonu için çalışmalar devam etmektedir.

Veritabanı tarafında, veri yönetimi için MongoDB tercih edilmiştir. Node.js ile MongoDB arasındaki etkileşimi kolaylaştırmak ve veri modellerini yönetmek amacıyla Mongoose kullanılmıştır. Veritabanı yapısı oluşturulmuş ve kullanıcı verilerinin güvenli bir şekilde saklanması sağlanmıştır.
Geliştirme sürecinde, hem istemci hem de sunucu tarafındaki bileşenlerin uyumlu bir şekilde çalışması hedeflenmektedir. Bir sonraki aşamada, gerçek zamanlı mesajlaşma sisteminin tamamlanması, online kullanıcı durumlarının yönetilmesi ve hata yönetimi mekanizmalarının güçlendirilmesi planlanmaktadır.

BÖLÜM 2

LİTERATÜR TARAMASI

2.1 Gerçek Zamanlı Mesajlaşma Uygulamaları

Gerçek zamanlı mesajlaşma, kullanıcıların anında birbirleriyle iletişim kurabilmesini sağlayan bir teknolojidir. Bugün, sosyal medya ve iş dünyasında en yaygın kullanılan iletişim araçlarından biri haline gelmiştir. WhatsApp, Telegram, Slack gibi popüler platformlar, dünya genelinde milyonlarca kullanıcıya hizmet vermektedir. Gerçek zamanlı mesajlaşma uygulamaları, kullanıcılar arasındaki etkileşimi hızlandırmak, bilgi paylaşımını kolaylaştırmak ve iş süreçlerini daha verimli hale getirmek için büyük bir öneme sahiptir. Bu tür uygulamalarda veri iletimi hızlı ve güvenli bir şekilde sağlanmalıdır. WebSocket gibi teknolojiler, sürekli açık bağlantılar üzerinden çift yönlü iletişimi mümkün kılarak, gerçek zamanlı veri iletiminin temelini oluşturur. WebSocket, TCP/IP protokolü üzerinden sürekli açık bağlantılar kurarak veri iletimini gerçekleştirir ve bu sayede geleneksel HTTP isteklerine kıyasla çok daha hızlı bir iletişim sağlar. Ancak, her ne kadar bu teknoloji verimli olsa da, güvenlik önlemleri ve kullanıcı gizliliği, kullanıcıların gerçek zamanlı mesajlaşma sistemlerini güvenle kullanabilmesi için kritik bir faktördür.

2.2 Socket.io ve WebSocket Teknolojileri

Gerçek zamanlı mesajlaşma uygulamalarında en yaygın kullanılan teknolojilerden biri Socket.io'dur. Socket.io, WebSocket teknolojisinin üzerine inşa edilmiş bir JavaScript kütüphanesidir ve web uygulamalarında gerçek zamanlı, iki yönlü iletişim sağlamak için yaygın olarak kullanılır. WebSocket bağlantılarının aksine, Socket.io, ağ kesintisi durumunda bağlantıyı yeniden kurabilir, bu da uygulamanın daha sağlam ve güvenilir olmasını sağlar. Socket.io'nun avantajlarından biri, tarayıcılar arası uyumluluğudur. Socket.io, farklı platformlarda çalışan tarayıcılar arasında sorunsuz bir iletişim sağlar. WebSocket ve Socket.io arasındaki fark, WebSocket'in daha temel ve düşük seviyeli bir iletişim protokolü olmasıdır, oysa Socket.io, WebSocket'i geliştirilmiş bir API ile daha kolay ve güçlü hale getirir. Socket.io, HTTP protokollerini de destekler, bu da onu daha esnek ve uygulamalara daha uygun hale getirir. Socket.io, ayrıca otomatik yeniden bağlantı, mesaj sıralama ve hata yönetimi gibi ek özelliklere sahiptir.

2.3 Güvenlik ve Kimlik Doğrulama

Gerçek zamanlı mesajlaşma uygulamaları, kullanıcıların kişisel bilgilerini ve mesajlaşmalarını içerdiğinden, güvenlik, bu tür sistemlerde kritik bir rol oynamaktadır. JSON Web Token (JWT), modern web uygulamalarında yaygın olarak kullanılan bir kimlik doğrulama ve yetkilendirme yöntemidir. JWT, kullanıcının kimliğini doğrulamak ve oturumları güvenli bir şekilde yönetmek için kullanılır. JWT token'ları, sunucu ile istemci arasında güvenli bir veri iletimi sağlar. Uygulamanın her iki tarafında da, şifreleme yöntemleriyle güvenlik sağlanır ve oturum süresi sınırlıdır. Bu sayede yetkisiz erişimlerin önüne geçilir.
Ayrıca, CSRF (Cross-Site Request Forgery) ve XSS (Cross-Site Scripting) gibi web güvenlik açıkları, mesajlaşma sistemlerinin önemli tehditlerindendir. CSRF, bir kullanıcının yetkisiz işlemler yapmasını sağlarken, XSS, kullanıcıların tarayıcılarında kötü amaçlı kod çalıştırılmasına neden olabilir. Bu nedenle, uygulama geliştirilirken bu tür saldırılara karşı alınacak önlemler, kullanıcı güvenliği için oldukça önemlidir. JWT ile birlikte HTTP-Only Cookies kullanarak bu tür saldırılara karşı ek koruma sağlanabilir. Bu yöntem, kullanıcı bilgilerini yalnızca tarayıcıda saklar ve kötü niyetli üçüncü şahısların erişimini engeller.

2.4 Gerçek Zamanlı Uygulamalarda Kullanıcı Durum Bilgisi

Gerçek zamanlı mesajlaşma uygulamalarında, kullanıcıların çevrim içi durumları (online/offline) önemli bir yer tutar. Kullanıcıların anlık olarak çevrim içi durumlarının izlenmesi, uygulamanın etkileşimli ve dinamik hale gelmesini sağlar. Çevrim içi durumu takip etmek için, Socket.io kullanılarak her kullanıcıya özel bir odak (room) tanımlanabilir. Bu sayede, her kullanıcı için bir bağlantı kurulabilir ve bu bağlantı üzerinden, kullanıcının çevrim içi durumu izlenebilir.
Çevrim içi durumu takip etmenin bir başka yolu ise, global state management sistemleri kullanmaktır. Bu tür sistemler, uygulamanın her yerinden ulaşılabilen bir durum yönetimi sağlar ve kullanıcının çevrim içi durumu gibi bilgiler, kullanıcı deneyimini zenginleştirir. Bu tür bir yönetim için Zustand gibi state yönetim kütüphaneleri kullanılabilir. Zustand, React ile birlikte entegre çalışarak, uygulamanın durumunu global bir şekilde yönetmek için oldukça basit ve etkili bir çözümdür.

2.5 Kullanıcı Arayüzü Tasarımı

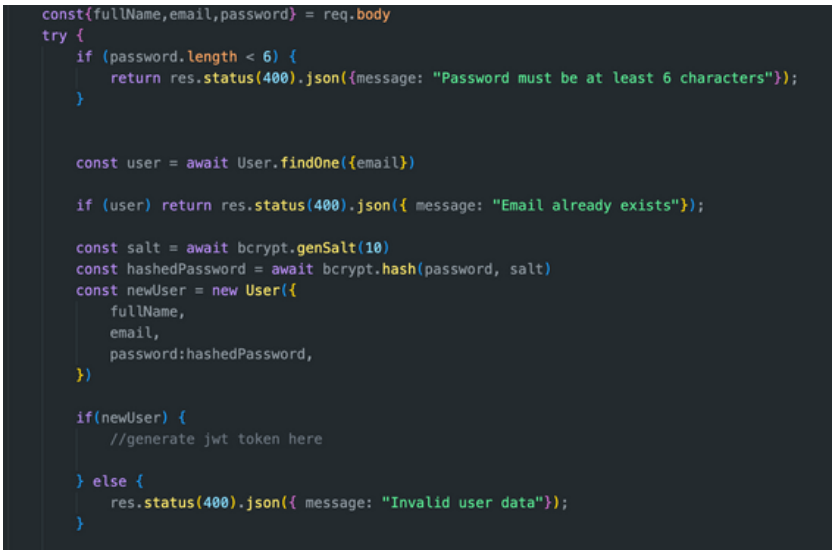
Gerçek zamanlı mesajlaşma uygulamaları, kullanıcı dostu bir arayüze sahip olmalıdır. Kullanıcıların hızlı ve verimli bir şekilde mesajlaşabilmesi için, mesajlaşma ekranları basit ve anlaşılır bir tasarıma sahip olmalıdır. TailwindCSS ve Daisy UI gibi modern CSS framework'leri, uygulama arayüzünün hızlı bir şekilde tasarlanmasını sağlar. Bu framework'ler, responsive (duyarlı) tasarımlar ve önceden yapılandırılmış bileşenlerle, geliştiricilerin zaman kazanmasını sağlar.
Ayrıca, kullanıcı deneyimini iyileştirmek için, gerçek zamanlı bildirimler, mesaj iletimi başarı göstergeleri (örneğin, "Mesaj Gönderildi" yazısı), sesli bildirimler ve emojiler gibi özellikler eklenebilir. Bu tür özellikler, kullanıcının etkileşimini artırır ve uygulamayı daha kullanıcı dostu hale getirir.

2.6 Sonuç

Gerçek zamanlı mesajlaşma uygulamaları, günümüzde hızla gelişen ve her geçen gün daha fazla kullanıcıya ulaşan önemli araçlardır. Bu uygulamalarda güvenlik, performans ve kullanıcı deneyimi en önemli unsurlardan biridir. MERN Stack ve Socket.io gibi teknolojilerin entegrasyonu, bu projelerde başarıyı elde etmenin anahtarıdır. Gerçek zamanlı veri iletimi için kullanılan bu teknolojiler, uygulamanın hem performansını artırır hem de güvenliğini sağlar. Ayrıca, kullanıcıların çevrim içi durumları ve mesajlaşma deneyimi gibi özellikler, uygulamanın kullanıcı dostu olmasına katkıda bulunur.

BÖLÜM 3

PROJE AŞAMALARI VE UYGULAMA GELİŞTİRME SÜRECİ



```
const{fullName,email,password} = req.body
try {
    if (password.length < 6) {
        return res.status(400).json({message: "Password must be at least 6 characters"});
    }


    const user = await User.findOne({email})

    if (user) return res.status(400).json({ message: "Email already exists"});

    const salt = await bcrypt.genSalt(10)
    const hashedPassword = await bcrypt.hash(password, salt)
    const newUser = new User({
        fullName,
        email,
        password:hashedPassword,
    })

    if(newUser) {
        //generate jwt token here

    } else {
        res.status(400).json({ message: "Invalid user data"});
    }
```

Kullanıcı Kayıt İşlemi

Bu bölümde, kullanıcı kayıt işlemini gerçekleştiren API route'u yer almaktadır. Kullanıcıdan alınan bilgilerle (tam ad, e-posta ve şifre) bir kullanıcı hesabı oluşturulmaktadır.

Kodun temel işleyişi şu adımları izler:

Şifre Uzunluğu Kontrolü: Kullanıcı tarafından girilen şifrenin güvenliğini sağlamak amacıyla, şifrenin en az 6 karakter uzunluğunda olup olmadığı kontrol edilmektedir. Eğer şifre belirtilen uzunluktan kısa ise, işlem durdurulmakta ve kullanıcıya bir hata mesajı gönderilmektedir.

E-posta Kontrolü: Kullanıcının girdiği e-posta adresinin veritabanında zaten kayıtlı olup olmadığı kontrol edilmektedir. Aynı e-posta adresine sahip başka bir kullanıcı varsa, işlem sonlandırılmakta ve kullanıcıya uygun bir hata mesajı verilmektedir.

Şifre Hash'leme: Kullanıcı şifresi, güvenlik amacıyla hash'lenmektedir. Bu işlemde, bcrypt kütüphanesi kullanılarak şifre hash'lenir ve şifre düz metin olarak kaydedilmez. Hash'leme, şifrelerin veritabanında güvenli bir şekilde saklanmasını sağlar.

Yeni Kullanıcı Oluşturma: Kullanıcının sağladığı verilerle (tam ad, e-posta ve şifre) yeni bir kullanıcı nesnesi oluşturulmaktadır. Bu kullanıcı nesnesi veritabanına kaydedilmeden önce, şifre güvenli bir şekilde işlenmiş ve e-posta doğrulaması yapılmıştır.

JWT Token Oluşumu: Eğer yeni kullanıcı başarıyla oluşturulursa, kullanıcının kimliğini doğrulamak ve oturum açmasını sağlamak için JWT (JSON Web Token) üretilmesi planlanmaktadır. Bu token, kullanıcının sonraki işlemlerinde kimliğini güvenli bir şekilde doğrulamak için kullanılacaktır.

KAYNAKÇA

React.js Documentation - React kütüphanesi hakkında resmi dokümantasyon.
https://react.dev/

Tailwind CSS Documentation - Tailwind CSS kütüphanesi hakkında resmi dokümantasyon.
https://tailwindcss.com/docs

Daisy UI Documentation - Daisy UI bileşenleri hakkında resmi dokümantasyon.
https://daisyui.com/docs/

Zustand State Management - React için global state yönetimi sağlayan Zustand kütüphanesi.
https://docs.pmnd.rs/zustand/getting-started/introduction

Socket.io Documentation - Gerçek zamanlı iletişim için Socket.io kütüphanesi.
https://socket.io/docs/v4/

Node.js Documentation - Node.js çalışma ortamı hakkında resmi dokümantasyon.
https://nodejs.org/en/docs

Express.js Documentation - Express.js framework'ü hakkında resmi dokümantasyon.
https://expressjs.com/en/starter/installing.html

JWT (JSON Web Token) Documentation - Kimlik doğrulama ve yetkilendirme için JWT kullanımı.
https://jwt.io/introduction

MongoDB Documentation - NoSQL veritabanı yönetimi için MongoDB resmi dokümantasyonu.
https://www.mongodb.com/docs/

Mongoose Documentation - Node.js ile MongoDB arasında köprü görevi gören Mongoose kütüphanesi.
https://mongoosejs.com/docs/

bcrypt.js Documentation - Şifreleme işlemleri için bcrypt.js kütüphanesi.
https://www.npmjs.com/package/bcrypt

dotenv Documentation - Node.js projelerinde ortam değişkenlerini yönetmek için dotenv kütüphanesi.
https://www.npmjs.com/package/dotenv

Morgan Documentation - HTTP isteklerini loglamak için kullanılan Morgan kütüphanesi.
https://www.npmjs.com/package/morgan