T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BSM 498 BİTİRME ÇALIŞMASI

KRİPTO CÜZDANI

G171210009 - Arif DAMAR
G171210028 - Ömer Çağrı ŞAYİR

Fakülte Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ
Tez Danışmanı : Öğr.Gör.Dr. Yüksel YURTAY

2021-2022 Bahar Dönemi

SİMGELER VE KISALTMALAR LİSTESİ

BTC	: Bitcoin
ETH	: Ethereum
JS	: JavaScript
TS	: TypeScript
CRUD	: Create, Read, Update, Delete

ÖZET

Anahtar kelimeler: Blockchain (Blok zincir), Kripto Varlıkları, Kripto Borsaları, Web Teknolojileri

Kripto para, tamamen dijital, şifrelenmiş, sanal para birimidir. Kripto para birimlerine sahip olmanın yöntemi reel para birimleri (Türk Lirası, Dolar, Euro vb.) ile satın almaktır. Bu satın alma işleminin gerçekleştirildiği sitelere "Kripto Para Borsası" denmektedir.

Kripto Cüzdanı projesi, kullanıcıların üyesi olduğu farklı Kripto Borsalarındaki farklı Kripto Varlıklarını tek bir uygulama/arayüz üzerinden takip edebilme ihtiyacına çözüm olmayı hedeflemiştir. Bu uygulama sayesinde birden fazla Kripto Borsa kullanan kişiler, bütün Kripto Varlıklarını tek arayüz üzerinden takip edebilmektedir.

Bu tasarım çalışmasında, kullanıcıların Kripto Varlıklarını takip edebileceği en büyük Kripto Borsalarından olan Binance ve KuCoin borsaları bulunmaktadır. Kullanıcılar, arayüz üzerinden istedikleri borsayı seçerek o borsada bulunan Kripto Varlıklarını görüntüleyebilir ve kendi cüzdanlarına istedikleri miktarda Kripto Varlık ekleyerek takibini sağlayabilmektedirler.

GİRİŞ

Bu projede kripto varlık sahibi kullanıcıların varlıklarının bulunduğu farklı borsalardaki varlıklarını tek bir uygulamada takip etmeleri amaçlanmıştır. Bu uygulama sayesinde tek bir arayüzde borsa fark etmeksizin tüm kripto varlıkların anlık durumu ve kâr/zarar oranı takip edilebilmektedir.

Bu tasarım projesinde, kullanıcıların Kripto Varlıklarının bulunabileceği en büyük 3 Kripto Varlık Borsasının ikisi olan Binance ve KuCoin borsaları bulunmaktadır. Uygulamayı kullanacak olan kullanıcılar bu iki borsada olan tüm varlıkları uygulama içerisindeki Cüzdanlarına ekleyebilir ve Dashboard üzerinden kâr/zarar durumlarını takip edebilirler.

Bu projenin alternatifi olarak CoinStats ve Kubera kabul edilebilir. Bu sistemlerden CoinStats daha çok borsalar ile doğrudan bir bağlantı kurarak entegrasyonunu sağlarken bu projenin kapsamında güvenlik sebebiyle böyle bir entegrasyon bulunmuyor. Kubera ise yine entegrasyon sağlamasının yanında sisteme kayıt olurken ödeme gerektirirken bu projenin gelir modeli, kayıt olma ve sistemin temel özelliklerinin kullanılması noktasında ücretsiz olmayı ve ücretlendirmeyi ise daha fazla sayıda Kripto Varlık takibi, değişim takibi gibi gelişmiş özellikleri kullanma noktasında yapmayı amaçlamaktadır.

Bu projenin geçen dönemki Bilgisayar Mühendisliği Tasarımı projesinden farkları ise frontend tarafında daha iyi, minimal, modern, yeni nesil ve responsive tasarıma sahip olması, history sekmesi ile kullanıcıların kendilerine özel hazırladıkları portföylerin zaman içerisinde yaşadığı değişiminin takibinin ve analizinin sağlanması. Ayrıca daha fazla Kripto Varlık Borsası eklenecek ve bu borsalardan socket yardımıyla anlık değişimler gösterilerek uygulamaya real-time özellikler eklenecek. Backend tarafında ise öncelikle daha fazla Kripto Varlık Borsası ile entegrasyon sağlanacak.

Kullanıcıların portföylerinde bulunan Kripto Varlıkların zaman içerisinde değişimlerinin takibinin ve analizinin sağlanabilmesi için belirli zaman aralıklarıyla çalışıp, Kripto Varlıkları ve karşılıklarını kayıt edecek bir servis geliştirilecek.

Gelecek 4 hafta içerisinde;

- Frontend tarafında iyileştirme yapılacaktır. Tasarım Tamamen değiştirilip responsive hale getirilecektir. Örneğin sol tarafta olan Sidebar silinip Navbar olarak üst tarafa alınacaktır. Bu tasarım ile daha sade ve daha kullanışlı bir deneyim planlanmaktadır.
- Backend tarafında ise daha fazla Kripto Para Borsalarıyla entegrasyon sağlanmaya çalışılacak ve belirli aralıklarla çalışıp, kullanıcıların anlık cüzdan değişimlerini kayıt altına alacak bir sistem tasarlanacak. Bu sayede kullanıcılar cüzdanlarının zaman içerisinde değişimini takip edebiliyor olacaklar.

Teknoloji Gereksinimleri

Projede kullanılması gereken teknolojiler şu şekildedir;

- Backend
    - Node.js
    - Express.js
    - TypeScript
    - Session Authentication
    - CronJob
- Frontend
    - Node.js
    - React.js
    - Tailwind CSS
    - TypeScript

    - Mantine
    - React-Router
    - Redux
- Database
    - NoSQL
    - MongoDB

Maliyet Raporu

Personel Bütçe Raporu

Personel	Görev	Zaman	Maaş	Toplam Gider
Arif DAMAR	CTO - Backend Dev.	12 Ay	10.500₺	126.000₺
Ömer Çağrı ŞAYİR	CEO - Frontend Dev.	12 Ay	10.500₺	126.000₺

Giderler Bütçe Raporu

Alet / Teçhizat / Yazılım / Yayın Adı	Adet	Teknik Özellik	Kullanım Amacı	Birim Fiyatı	Toplam Tutar
Apple MacBook Pro Bilgisayar	2	M1 Pro işlemcili 1 TB SSD, 16 GB RAM	Geliştirme	35.000₺/adet	70.000₺
Monitör	2	Widescreen 27 inç	Ofis Kullanımı	2.500₺/adet	5.000₺
Ofis	12 Ay	50 m²	Ofis Kullanımı	5.000₺/ay	60.000₺
Masa	2	150cmX70 cm	Ofis Kullanımı	500₺/adet	1.000₺
Koltuk	2	Ergonomik, tekerlekli	Ofis Kullanımı	1.200₺/adet	2.400₺
Domain Hizmeti	12 Ay	Alan Adı	Web Sitesi	16,6₺/ay	200₺

			Alan Adı		
Azure	12 Ay	Hosting	Backend ve Frontend Projelerini Barındırma	1.000₺/ay	12.000₺
Github Codespaces	12 Ay (1992 iş saati)	2 Core CPU	Ortak ve Güvenli Geliştirme Ortamı	2,5₺/saat	4.980₺

Backend Geliştirme

Backend geliştirmesi TypeScript. MongoDB, JOI kütüphanesi, NodeJS kullanılarak yapılmıştır.

Veritabanı Modelleri

Veritabanı için şu ana kadar MongoDB kullanıldı. MongoDB'nin Bulut Tabanlı uygulaması MongoDB Atlas üzerinde çalışıldı.

Projede var olan veritabanı modelleri ve bu modellerdeki alanlar şunlardır:

- User (Kullanıcı) Modeli
  - email
  - password
  - walletId

- UserCrypto Modeli
  - walletId
  - exchangeId
  - symbol
  - amount
  - firstPrice
  - lastPrice

- Wallet Modeli
  - cryptoIds
  - balance

- Exchange Modeli
  - name
  - baseApi
  - symbolListEndpoint
  - priceEndpoint
  - symbols
  - logoUrl

Authentication

1- Authentication için Register (kayıt olma), Login (giriş yapma), Logout (çıkış yapma) ve o an giriş yapmış kullanıcının bilgilerinin bulunduğu Me adında dört adet endpoint tanımlanmıştır.

```
import express from "express";
import { AuthController } from "../../controllers/auth";
import { Auth } from "../../middlewares/auth";

const router = express.Router();
const authController = new AuthController();

router.route("/register").post(authController.register);
router.route("/login").post(authController.login);
router.route("/logout").post(Auth.authenticate, authController.logout);
router.route("/me").get(Auth.authenticate, authController.me);

export default router;
```

Şekil 1.1.1. Auth endpointlerinin belirtildiği kod tanımı

2- Register ve Login endpointlerine her kullanıcı erişebilirken, Logout ve Me endpointlerine sadece giriş yapmış kullanıcıların erişebilmesi Auth sınıfının authenticate Middleware'i ile sağlanmıştır.

```
export class Auth {
    public static async authenticate(request: Request, response: Response, next: NextFunction) {
        try {
            const userId = request.session.userId;

            if (userId) {

                const user = await User.findById(userId);

                if (!user) {
                    return response.status(404).send(new FailureResult('User not found'));
                }

                response.locals.user = user;
                return next();
            } else {
                return response.status(401).json(new FailureResult('Unauthenticated'));
            }

        } catch (error) {
            console.log(error);
        }
    }
}
```

Şekil 1.1.2. Auth.authenticate middleware'inin belirtildiği kod tanımı

Register

Register (kayıt olma) için kullanıcıdan gelen e-mail ve şifre tutulmuştur. Önce bu veriler JOI kütüphanesi yardımı ile önceden belirlenen kurallara uyup uymadığı kontrol edilmiştir (validation). Daha sonra veritabanında aynı e-mail olup olmadığı kontrol edilmiştir. Eğer bu e-mailde bir kullanıcı bulunduysa "Bu e-mailde bir kullanıcı zaten mevcuttur. Lütfen başka e-mail ile kayıt olmayı deneyiniz." uyarısı gönderilmiştir.

Eğer böyle bir e-mail mevcut değilse önce 0 balance ile bir Wallet (cüzdan) oluşturuluyor. Daha sonra kullanıcının şifresi kriptoloji ile alakalı kütüphaneler ile (bcrypt) şifrelenip, e-mail, şifreli parola ve yeni oluşturulan cüzdan ID'si ile beraber yeni bir kullanıcı oluşturuluyor.

```
public async register(request: Request, response: Response) {
  try {
    const { email, password } = request.body;

    await userRegisterValidator.validateAsync({ email, password });

    const existingUser = await User.findOne({ email });

    if (existingUser) {
      return response.status(400).send(new FailureResult("A user with this email already exists. Try signing in."));
    }

    const userWallet = await Wallet.create({ balance: 0 });

    const hashedPassword = await bcrypt.hash(password, 12);
    const result = await new User({
      email,
      password: hashedPassword,
      walletId: userWallet.id,
    }).save();

    if (result != null) {
      return response.status(201).send(new SuccessResult("Registered successfully", null));
    }

    return response.status(500).send(new SuccessResult("Registeration failed", null));
  } catch (error: any) {
    if (error.isJoi) {
      return response.status(400).send(new FailureResult("Validation error: " + error.message));
    }

    console.log(error);
    return response.status(500).send(new FailureResult("Something went wrong."));
  }
}
```

Şekil 1.1.2. Register endpointinin kod tanımı

Login

Login için kullanıcıdan gelen e-mail ve şifre öncelikle JOI kütüphanesi yardımı ile doğrulanıyor. Daha sonra bu e-mail'de bir kullanıcı var mı diye Veritabanına bakılıyor. Eğer bu verilerde bir kullanıcı yok ise "Kullanıcı bulunamadı" şeklinde hata dönülüyor.

E-mail var ise bu kullanıcının parolası ile kullanıcının girdiği parola karşılaştırılıyor. Yanlışsa "Girilen bilgiler hatalı" şeklinde bir hata dönülüyor. Doğruysa "Giriş başarılı" şeklinde 200 kodlu bir cevap gönderilir.

```
public async login(request: Request, response: Response) {
  try {
    const { email, password } = request.body;

    await userLoginValidator.validateAsync({ email, password });

    const existingUser = await User.findOne({ email });

    if (!existingUser) {
      return response.status(401).json(new FailureResult("User not found"));
    }

    const isPasswordCorrect = await bcrypt.compare(password, existingUser.password);

    if (!isPasswordCorrect) {
      return response.status(401).json(new FailureResult("Invalid credentials"));
    }

    request.session.userId = existingUser._id.toString();

    return response.status(200).json(new SuccessResult("Login successful", null));
  } catch (error: any) {
    if (error.isJoi) {
      return response.status(400).send(new FailureResult("Validation error: " + error.message));
    }

    console.log(error);
    return response.status(500).json(new FailureResult("Something went wrong."));
  }
}
```

Şekil 1.1.3. Login endpointinin kod tanımı

Job Handler Servisi

Bu servis; TypeScript, MongoDB, Cron Expression ve NodeJS kullanılarak geliştirilmiştir. Servisin amacı; belirli zaman aralıklarıyla çalışıp, kullanıcıların cüzdanlarında bulunan kripto varlıkların anlık değerlerini kontrol ederek, kullanıcılara toplam cüzdan değerlerinin zamana bağlı değişimini takip edebilmelerine olanak sağlamaktır.

Veritabanı Modelleri

Job Handler servisi düzenli aralıklarla çalıştığında hangi Job'ların çalışacağını belirlemek adıma aşağıdaki model tasarlanmıştır. Her Wallet oluşturulduğunda o Wallet'ın referans verdiği bir de RecurringJob oluşturulur. Böylece her RecurringJob sadece 1 Wallet ile ilişkilendirilmiş olur.

- RecurringJob
    - _id
    - schedule // cron expression
    - lastRunAt
    - nextRunAt
    - beingTriggered
    - beingTriggeredAt
    - enabled
- Wallet
    - cryptoIds
    - balance
    - recurringJobId

İşleyiş

1- Uygulama her 10 saniyede bir çalışıp, veri tabanındaki RecurringJob modelinde bulunan kayıtlardan nextRunAt alanındaki tarih verisi, uygulamanın çalışma tarihinden küçük veya eşit olan kayıtları seçer.

```
async function getJobsToRun() {
  const now = new Date();
  const twoMinutesAgo = new Date(now.getTime() - 2 * 60 * 1000);

  await RecurringJobModel.updateMany(
    {
      nextRunAt: { $lte: new Date() },
      enabled: true,
      $or: [{ beingTriggered: false }, { beingTriggered: true, beingTriggeredAt: { $lt: twoMinutesAgo }
}],
    },
    {
      $set: { beingTriggered: true, beingTriggeredAt: now, lastRunAt: now },
    }
  );

  const jobsToRun = await RecurringJobModel.find({ beingTriggeredAt: now });
  return jobsToRun;
}
```

Şekil 2.1.1. getJobsToRun fonksiyonu tanımı

2- Seçilen tüm Job'ların paralel olarak çalışabilmesi için tüm Job'lar maplenerek bir Promise dizisi içine alınır ve sonrasında bu dizi await edilir.

```
async function checkAndRunRecurringJobs() {
  const jobsToRun = await getJobsToRun();

  console.log("running " + jobsToRun.length + " jobs");

  const jobPromises = jobsToRun.map((job) => {
    return handleRecurringJob(job);
  });

  await Promise.all(jobPromises);
}
```

Şekil 2.1.2. Job'ların paralel olarak çalıştırılmasını sağlayan kod tanımı

3- Her bir Job'ın tek tek işlendiği handleRecurringJob fonksiyonunda öncelikle Job'a bağlı olan Wallet ve sonrasında da o Wallet'a dahil olan UserCrypto'ları veri tabanından çekip, var olup olmadıkları kontrol edilir.

```
async function handleRecurringJob(job: IRecurringJobDocument) {
  const wallet = await Wallet.findOne({ recurringJobId: job._id });

  if (!wallet) {
    return false;
  }

  const walletCryptos = await UserCrypto.find({ walletId: wallet._id }).populate({
    path: "exchange",
    select: "-symbols -createdAt -updatedAt",
  });

  if (!walletCryptos || walletCryptos.length === 0) {
    return false;
  }
  .
}
```

Şekil 2.1.3. wallet ve walletCryptos kontrollerinin yapıldığı kod tanımı

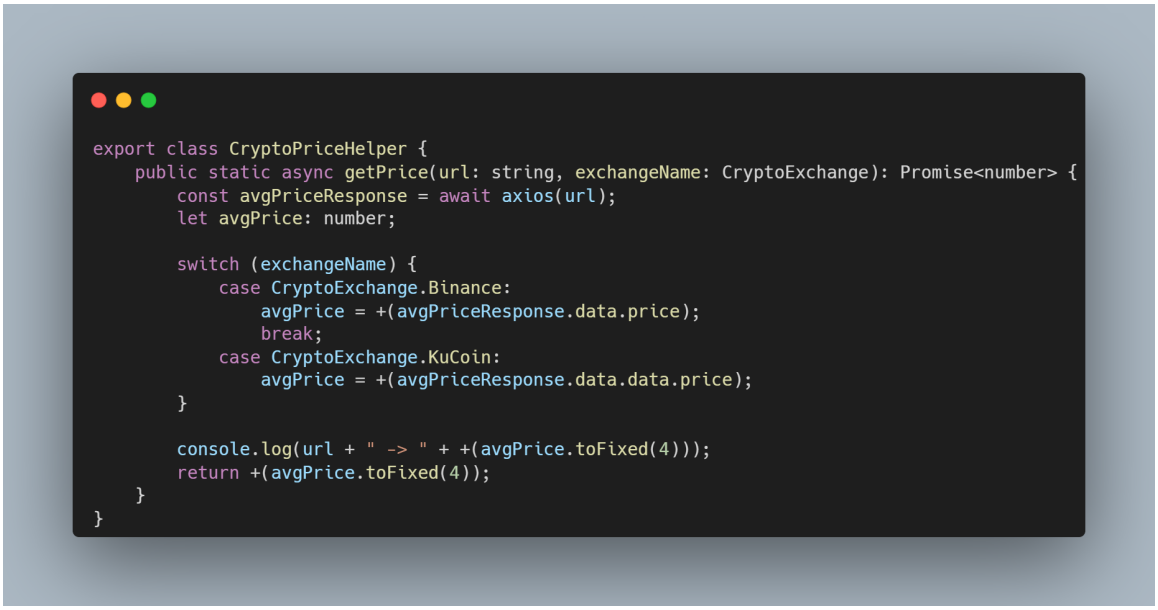
4- Bu verilerin varlığını doğruladıktan sonra her bir walletCrptos için tek tek anlık price değer alınır ve ilgili alanlar güncel veriye uygun olacak şekilde güncellenir.

```
async function handleRecurringJob(job: IRecurringJobDocument) {
  .
  wallet.balance = 0;
  for (const crypto of walletCryptos) {
    const currentPrice = await CryptoPriceHelper.getPrice(
      crypto.exchange.baseApi + crypto.exchange.priceEndpoint + crypto.symbol,
      crypto.exchange.name
    );

    wallet.balance += +(currentPrice * crypto.amount).toFixed(4);
    wallet.balance = +wallet.balance.toFixed(4);
    crypto.lastPrice = currentPrice;
    await crypto.save();
  }
  await wallet.save();
  .
}
```

Şekil 2.1.4. Her crypto için price değerinin güncellendiği kod tanımı

5- Güncel price değerinin alınması için CryptoPriceHelper sınıfının kullanılan getPrice adındaki statik metodu kullanılır.



```
export class CryptoPriceHelper {
    public static async getPrice(url: string, exchangeName: CryptoExchange): Promise<number> {
        const avgPriceResponse = await axios(url);
        let avgPrice: number;

        switch (exchangeName) {
            case CryptoExchange.Binance:
                avgPrice = +(avgPriceResponse.data.price);
                break;
            case CryptoExchange.KuCoin:
                avgPrice = +(avgPriceResponse.data.data.price);
        }

        console.log(url + " -> " + +(avgPrice.toFixed(4)));
        return +(avgPrice.toFixed(4));
    }
}
```

Şekil 2.1.5. Her crypto için price değerinin güncellendiği kod tanımı

6- Güncel price değeri kaydedildikten sonra Job'ın nextRunAt ve beingTriggered alanları güncellenir. Böylece ilgili Job'ın işinin tamamlandığı ve sıradaki çalışma için nextRunAt alanındaki tarihin gelmesi beklendiği belirtilmiş olur.

```
async function handleRecurringJob(job: IRecurringJobDocument) {
  .
  .
  const parsed = parseExpression(job.schedule);
  const nextRunAt = parsed.next().toDate();

  job.beingTriggered = false;
  job.nextRunAt = nextRunAt;
  await job.save();

  return true;
}
```

Şekil 2.1.6. Job'ın alanlarının güncellendiği kod tanımı

Frontend Geliştirme

Frontend (Ön yüz) geliştirmesi React, TypeScript, TailwindCss, Mantine.dev, Toaster kütüphaneleri yardımıyla yapılmıştır.

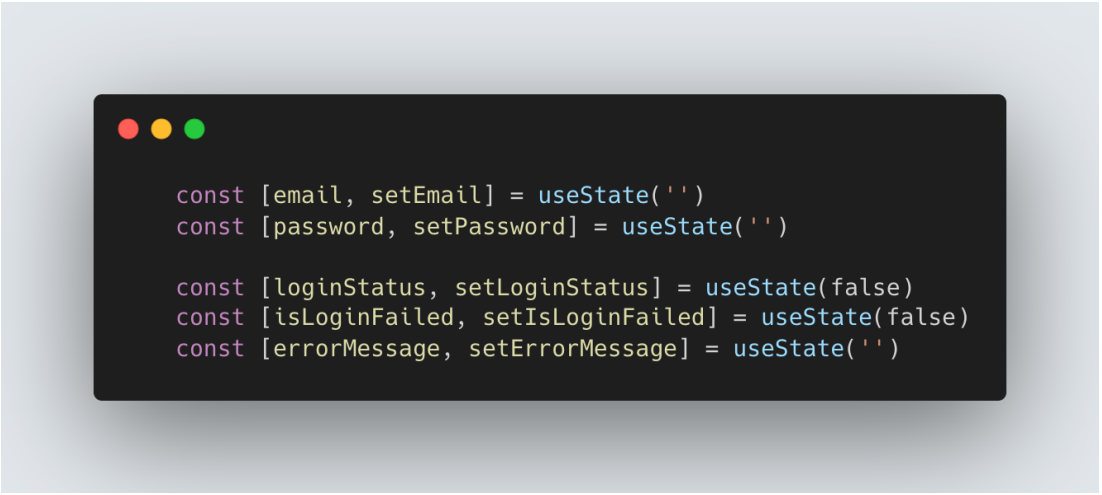
Register (Kayıt Ol) Sayfası

Bu sayfada gösterilen form vesilesi ile kullanıcıdan alınan E-mail ve Parola bilgileri ile -eğer o maile kayıtlı başka bir hesap yoksa- kullanıcı oluşturulur.

Bu sayfanın tasarımı için Tailwind Css kullanılmıştır.

Login (Giriş Yap) Sayfası

Bu sayfada gösterilen form vesilesi ile kullanıcıdan alınan E-mail ve Parola bilgileri server tarafına gönderilir. Eğer veritabanında kayıtlı bir e-mail varsa ve bu e-mailin parolası ile kullanıcının girdiği parola eşlenir ise kullanıcı direkt olarak Dashboard (Gösterge Paneli) sayfasına yönlendirilir. Eğer bilgilerde yanlışlık var ise Toaster kütüphanesi yardımı ile oluşturulan hata mesajları sayfanın sağ üst tarafında birkaç saniye görülür.



Şekil 3.2.1. Login sayfası değişken ve statelerin tanımı

```
const login = async () => {
        try {
            const response = await axios.post(`/api/auth/login`, {
                email,
                password,
            })

            setLoginStatus(response.data.status)
            if (response.data.status) {
                history.push('/dashboard')
            } else {
                console.log(response.data.message)
            }
        } catch (error: any) {
            console.log('ErrorX:', error.response.data)
            setErrorMessage(error.response.data.message)
            setIsLoginFailed(true)
        }
    }
```

Şekil 3.2.2. Login sayfası için ana login fonksiyonu

```
const checkIsLoggedIn = async () => {
    try {
        const response = await axios.get(`/api/auth/me`)
        if (response.data.status) {
            history.push('/dashboard')
        }
    } catch (error) {
        console.log('ErrorY: ', error)
    }
}
```

Şekil 3.2.3. Login sayfası için giriş yapılıp yapılmadığının kontorolünü yapan fonksiyon

Navbar Yapısı

Navbar, kullanıcı başarı ile giriş yaptıktan sonra public olmayan protected sayfaların hepsinde sayfanın üst tarafında görülmektedir.

En sol tarafta uygulamanın simgesi, ortanın biraz solunda menüler ve en sağda kullanıcı bilgileri ile Logout butonu vardır.

Navbar tasarımı Tailwind Css ile yapılmıştır.