CustomSchedule-Graph-Coloring-for-Course-Management

Artun Kara
Bilişişim Sistemleri Mühendisliği
(Öğrenci)
Kocaeli Üniversitesi
Kocaeli Umuttepe
211307030@kocaeli.edu.tr

Tayfun Açıcı
Bilişişim Sistemleri Mühendisliği
(Öğrenci)
Kocaeli Üniversitesi
Kocaeli Umuttepe
211307030@kocaeli.edu.tr

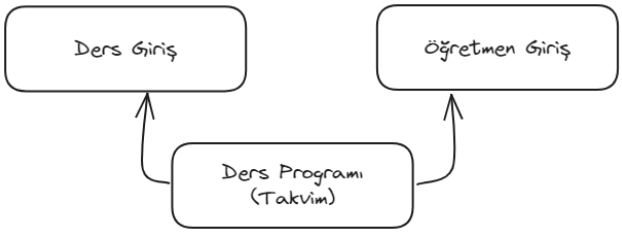
Elif Özkan
Bilişişim Sistemleri Mühendisliği
(Öğrenci)
Kocaeli Üniversitesi
Kocaeli Umuttepe
211307030@kocaeli.edu.tr

***Özet* — "CustomSchedule-Graph-Coloring-for-Course-Management" projesi, kurs planlaması için geliştirilen bir yazılım parçasıdır. Graf renklendirme algoritmalarını kullanarak çakışan dersleri azaltmayı ve öğrenci gereksinimlerine uygun zaman çizelgeleri oluşturmayı hedefler. Eğitim kurumlarının ders programlarını daha verimli ve esnek bir şekilde düzenlemelerine olanak tanır.**

***Abstract* — The 'CustomSchedule-Graph-Coloring-for-Course-Management' project is a software component developed for course scheduling. It aims to reduce overlapping classes and create timetables that suit students' needs by utilizing graph coloring algorithms. It allows educational institutions to organize their class schedules more efficiently and flexibly.**

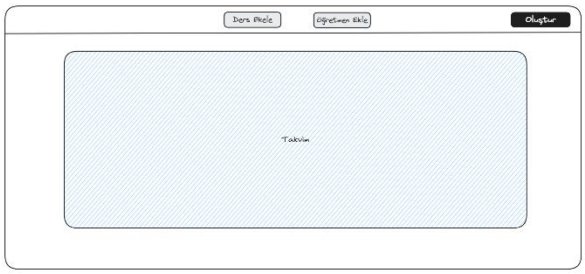
Keywords—component, formatting, style, styling, insert (*key words****)***

I. UI -UX ÇALIŞMLARI



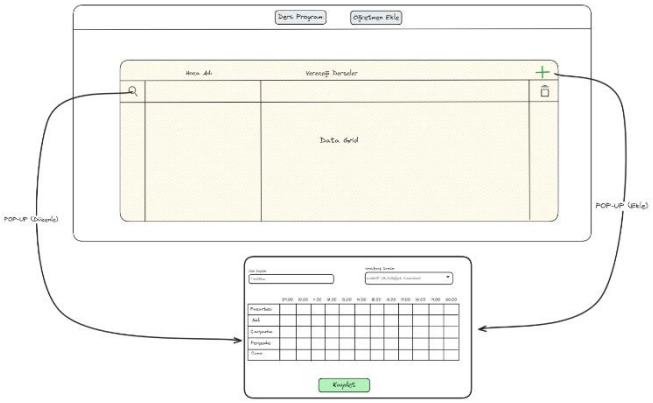
1 - Web Sayfası Akış Diyagramı

Eğitim kurumları, öğrencilerin başarısını artırmak ve verimliliği maksimize etmek için karmaşık programlarını düzenlemek zorundadır. Bu zorluğu hafifletmek ve ders programlarını optimize etmek için geliştirilmiş bir yaklaşım, "CustomSchedule-Graph-Coloring-for-Course-Management" adlı ders planlama uygulamasıyla hayat buluyor.



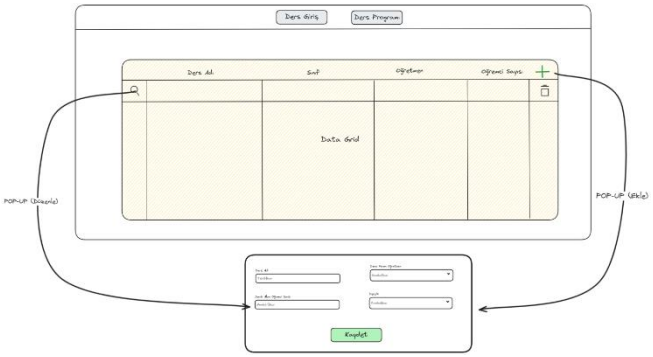
2 - Ders Programı Sayfası

Bu uygulama, ana sayfasında kullanıcılarına öğretmenlerin ve derslerin planlamasını bir araya getiren dinamik bir tablo sunuyor. Kullanıcılar, ders programı üzerinde hızlıca değişiklikler yapabiliyor, çakışmaları azaltabiliyor ve öğrenci gereksinimlerine uygun olarak dersleri düzenleyebiliyor. Ana sayfa üzerinden kullanıcılar, daha fazla detay eklemek veya değişiklik yapmak istediklerinde öğretmen eklemek veya yeni dersler oluşturmak için ilgili sayfalara geçiş yapabiliyorlar. Bu sayede, uygulama hem genel bir bakış sunuyor hem de detaylı düzenlemeler için kullanıcıları yönlendiriyor.



3 - Öğretmen Giriş

Öğretmen ekleme ve ders oluşturma sayfaları, kullanıcı dostu arayüzleriyle dikkat çekiyor. Kullanıcılar, yeni öğretmenlerin bilgilerini girmek veya yeni derslerin detaylarını belirlemek için kolayca yönlendiriliyorlar. Bu şekilde, kullanıcılar istedikleri özelleştirilmiş ders planlarını oluşturmak için gereken esnekliğe sahip oluyorlar.



4 - Ders Giriş

Sonuç olarak, "CustomSchedule-Graph-Coloring-for-Course-Management" projesi, eğitim kurumlarına dinamik, verimli ve özelleştirilebilir bir ders planlama deneyimi sunuyor. Bu yaklaşım, hem genel bakış sağlayarak hem de ayrıntılara inerek eğitim yöneticilerine ve planlamacılara değerli bir araç sunuyor. Bu sayede, kurumlar öğrencilerin ihtiyaçlarını daha iyi karşılayabilir ve ders programlarını daha etkin bir şekilde optimize edebilirler..

II. AR UYGULAMASI TASARIMI

A. *Genel Amacı*

Eğitim, teknolojinin nimetlerinden yararlanarak sürekli olarak gelişmekte ve dönüşmektedir. Son zamanlarda, sınıf ortamındaki dersleri daha etkin bir şekilde takip etmek ve canlı olarak izleme ihtiyacı, artırılmış gerçeklik (AR) teknolojisiyle birleşti. Bu ihtiyaçtan yola çıkarak geliştirilen bir proje, Unity motorunu ve ARFoundation ile ARKit gibi kütüphaneleri bir araya getirerek eğitimde yeni bir boyutun kapılarını aralıyor.

Bu projede, sınıflarda işlenen dersleri canlı olarak izleyebilmek amacıyla Unity motoru kullanıldı. Özellikle, ARFoundation ve ARKit gibi kütüphaneler, gerçek dünyayı sanal olarak genişletebilme ve canlı dersleri izleme konusunda büyük bir adım sağladı. Bununla birlikte, projenin temeli, veri tabanıyla bağlantı kurmayı mümkün kılan scriptlerin yazılmasıyla başladı.

İlk aşamada, veri tabanı ile etkileşimi sağlayacak scriptler geliştirildi. Ardından, projenin merkezinde yer alan istek alma ve veri çekme işlemlerini gerçekleştiren scriptler yazıldı. Bu adımların temel amacı, veri tabanından gelen canlı ders bilgilerini alarak bunları ekranda göstermeyi sağlamaktı.

Projenin ana hedefi, AR teknolojisinin gücünü kullanarak, sınıf ortamında gerçekleşen dersleri canlı olarak izleyebilmekti. Bu amaç doğrultusunda, scriptlerin yazılması ve veri tabanı ile iletişim kurulması projenin temelini oluşturdu. Elde edilen verilerin ekranda basit bir şekilde görüntülenmesi, bu teknolojinin eğitimdeki potansiyelini ve gelecekteki kullanım alanlarını gösterme adına önemli bir adım oldu.

Sonuç olarak, Unity motoru, ARFoundation ve ARKit gibi teknolojilerin birleşimiyle geliştirilen bu proje, eğitimde artırılmış gerçeklik uygulamalarının geleceğini şekillendirmeye aday. Canlı dersleri izleme ve sanal dünya ile gerçek dünyayı bir araya getirme amacıyla geliştirilen bu tür projeler, eğitimde teknolojinin sunduğu potansiyeli ortaya koymak açısından önemli bir adım niteliği taşıyor.

B. *Uygulama Arayüz*

5 - Ar Ugulama Arayüzü

Eğitimde artırılmış gerçeklik teknolojisinin kullanılmasıyla birlikte, bu teknolojinin kullanıcı dostu arayüzü ve pratik işlevselliği büyük önem taşıyor. Geliştirilen bir projede, sınıf takibini sağlamak için oluşturulan arayüz, kullanıcıya basit ve etkili bir deneyim sunmayı hedefliyor.

Uygulamanın ana arayüzü, "Sınıf Algıla" butonuyla dikkat çekiyor. Bu buton, kullanıcının kamerayı sınıf numarasına doğru tuttuğu zaman veri almak için projenin veri tabanına istek yapılmasını sağlıyor. Ardından, alınan veriler "classNumber" ve "classData" olarak ekranda basit bir şekilde gösteriliyor.

Bu sürecin yanı sıra, "Geri Dön" butonu kullanıcıya, mevcut sınıf izleme işlemini sonlandırıp uygulamanın ana menüsüne dönme olanağı tanıyor. Böylece kullanıcı, yeni bir sınıf izlemek veya farklı bir işlem yapmak için kolaylıkla uygulamaya geri dönebiliyor.

Ancak, kameranın yönü değiştirildiğinde tekrar algılama özelliği olmaması, kullanıcının uygulamadan çıkıp tekrar girmesi gerekliliğini doğuruyor. Bu durum, uygulamanın kullanımında bazı pratik kullanım sınırlamalarına neden olabiliyor.

Projede yer alan bu basit arayüz, artırılmış gerçeklik teknolojisinin eğitimdeki potansiyelini kullanıcının erişebilirliğini ve kullanım kolaylığını gözeterek sunmayı amaçlıyor. Ancak, kameranın yön değiştirmesiyle yeniden algılama yapma eksikliği, kullanıcı deneyimini tam anlamıyla sorunsuz hale getirebilme adına geliştirme fırsatları sunabilir.

Bu tür arayüzler, artırılmış gerçeklik teknolojisinin eğitimde kullanılmasını daha erişilebilir ve kullanıcı dostu hale getirerek, öğrenci ve eğitimcilerin bu teknolojiden daha fazla faydalanmasını sağlamayı hedefliyor.

C. *Benzer Mantıktaki Teknolojiler*

Günümüzde teknolojinin eğitimdeki rolü giderek artarken, artırılmış gerçeklik ve QR kod okuma gibi teknolojiler farklı amaçlar için benzer mantıkla kullanılmaktadır. Her iki teknoloji de kamera algılama, veri tabanı bağlantısı ve ekrana bilgi bastırma gibi temel işlevleri paylaşırken, farklı kullanım alanlarıyla dikkat çekerler.

Artırılmış gerçeklik uygulamalarında, kamera sınıfı algılayarak canlı dersleri izleme amacıyla kullanılırken, QR kod okuma uygulamalarında kamera, barkod etiketlerini okuyarak içerdikleri bilgileri elde etmek için kullanılır.

Her iki teknoloji de veri tabanı bağlantısıyla çalışır. Artırılmış gerçeklik uygulamalarında, sınıf bilgileri veya ders detayları gibi veriler veri tabanından istenir ve ekrana yansıtılır. QR kod okuma uygulamalarında ise QR kodun içeriği, kodun bağlı olduğu veri tabanından çekilir ve kullanıcıya sunulur.

Ekrana bilgi bastırma konusunda her iki teknoloji de benzer bir işlevselliğe sahiptir. Artırılmış gerçeklik uygulamalarında, veri tabanından alınan bilgiler kullanıcıya görüntülenir ve uygun bir arayüzle sunulur. QR kod okuma

uygulamalarında ise kodun içindeki bilgiler, kullanıcıya bir arayüz yardımıyla gösterilir.

Farklılık ise geri dönüş ve işleme yeteneğinde ortaya çıkar. Artırılmış gerçeklik uygulamalarında, sınıf değiştirme veya farklı işlemler için geri dönüş sağlayan butonlar bulunabilirken, QR kod okuma uygulamalarında genellikle farklı bir tarama yapabilmek için geri dönüş imkanı bulunur.

Sonuç olarak, artırılmış gerçeklik ve QR kod okuma teknolojileri, kamera algılama, veri tabanı bağlantısı ve bilgi sunma konularında benzer mantığı paylaşsa da, farklı kullanım amaçlarına hizmet ederek eğitimden perakende sektörüne kadar geniş bir yelpazede kullanılmaktadır. Bu teknolojiler, kullanıcıya farklı deneyimler sunarak teknolojinin gücünü günlük yaşamın bir parçası haline getirmektedir.

D. Verinin Alınması

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.XR.ARFoundation;

public class ArScript : MonoBehaviour
{

    public string webServiceURL = "";

    public string WebServiceURL
    {
        get => webServiceURL;
        set => webServiceURL = value;
    }

    void Start()
    {
        StartCoroutine(GetWebServiceData());
    }
    //Servisten veri almak için istekte bulun
    IEnumerator GetWebServiceData()
    {
        UnityWebRequest www = UnityWebRequest.Get(webServiceURL);

        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
        {
            Debug.LogError("Web servisi çalışmıyor " + www.error);
        }
        else
        {
            // Web servis yanıtı başarıyla alındı.
            Debug.Log("Web servisi çalışıyor: " + www.downloadHandler.text);
```

6 - Verinin Alınması İle İlgili Kod parçası

Bir projede, belirli bir servisle iletişim kurup veri almak amacıyla yazılan scriptin önemli bir kısmı, iletişim fonksiyonları ve koşulları içerir. Özellikle, GetWebServiceData() fonksiyonu kullanılarak servisle iletişim sağlanmaya çalışılırken, iletişimde başarısızlık durumunda hata mesajları kullanıcıya sunulur.

Scriptte, GetWebServiceData() fonksiyonu belirli bir URL üzerinden istek yaparak veri almayı hedefler. Ancak, servisle başarılı bir şekilde iletişim kurulamadığı zamanlarda, hata mesajları kullanıcıya gösterilir. Bu durum, iletişim hatası yaşandığında kullanıcının bilgilendirilmesini ve olası sorunların anlaşılmasını sağlar.

Özellikle, bu tür iletişim fonksiyonları ve hata kontrol mekanizmaları, yazılım geliştirme süreçlerinde önemli bir yer tutar. Hata mesajları, kullanıcı deneyimini iyileştirerek, olası problemlerin anlaşılmasına ve çözülmesine yardımcı olur. Ayrıca, servis bağlantısında yaşanan herhangi bir sorunun tespit edilmesine olanak tanır.

GetWebServiceData() fonksiyonu, belirli bir URL üzerinden veri isteği yaparak projenin ihtiyacına uygun şekilde servisle iletişim kurmayı hedefler. Bu iletişim sürecinde, başarısızlık durumunda hata mesajlarının kullanıcıya sunulması, uygulamanın kullanılabilirliği ve sorun giderme süreçlerinde önemli bir rol oynar.

Sonuç olarak, iletişim sağlama ve hata kontrolü, yazılım geliştirme süreçlerinde verimliliği ve kullanıcı deneyimini artırmayı amaçlar. GetWebServiceData() fonksiyonu gibi iletişim araçları, projelerin başarılı bir şekilde servislerle etkileşim kurmasına olanak tanırken, hata mesajları ise olası sorunları tespit etmeyi ve çözmeyi kolaylaştırır.

E. Verinin Ekranda Gösterilmesi ve Karşılaşılan Hatalar

```
private TextMeshProUGUI _resultText;

// Örnek API endpoint URL'si(Veri tabanı için)
private string apiUrl = "";
public Text classname;
public Text classdata;

void Start()
{
    // Butona tıklama olayını dinle
    _button.onClick.AddListener(RequestData);
}

// Butona tıklandığında çalışacak fonksiyon
private void RequestData()
{
    StartCoroutine(GetDataFromApi());
}

// Veri çekme işlemi
private IEnumerator GetDataFromApi()
{
    // İstek yap
    using (WWW www = new WWW(apiUrl))
    {
        yield return www;

        // İstek başarılı mı kontrol et
        if (www.error == null)
        {
            // Gelen veriyi ekrana yazdır
            _resultText.text = "Cevap: " + www.text;
            classname.text=_resultText.text;
            classdata.text=_resultText.text;
            Debug.Log(_resultText.text);
        }
```

7 - Verilerin Ekrana Getirilmesi (1)

Bir projenin önemli aşamalarından biri, alınan verilerin kullanıcıya görüntülenmesi için ayrı bir scriptin yazılmasıdır. Bu script, classname ve classdata gibi değişkenler aracılığıyla alınan verilerin ekrana basılmasını sağlar. Özellikle, bir butona tıklandığında çalışan bir fonksiyon tanımlanarak, veri tabanına istek yapılır ve istenilen veri çekilir.

Bu aşamada amaç, veri tabanından istenen bilgiyi almak ve kullanıcıya göstermektir. Script, istek yapıldığında gelen cevabı _resultText gibi değişkenlere atayarak ekrana aktarır. Bu sayede, kullanıcı alınan bilgileri okuyabilir ve istenilen verileri görsel olarak görebilir.

Verilerin ekrana aktarılması, projenin kullanılabilirliğini ve anlaşılırlığını artırmak adına önemlidir. Kullanıcıların istedikleri bilgilere kolaylıkla erişebilmesi, projenin kullanıcı dostu olmasını sağlar. Bu sayede, verilerin doğru şekilde görselleştirilmesiyle birlikte, kullanıcılar istedikleri bilgilere hızlıca ulaşabilirler.

Sonuç olarak, verilerin ekrana aktarılması için yazılan script, projenin veri tabanından alınan bilgileri kullanıcıya sunmayı hedefler. Bu sayede, kullanıcılar istedikleri bilgilere kolayca erişebilir ve projenin amacına daha rahatlıkla ulaşabilirler. Verilerin ekrana basılması, projenin kullanıcı deneyimini geliştirmek ve bilgi erişimini kolaylaştırmak için kritik bir adımdır.

```
}
//JSON Formatında verileri ekrana bastır
/*https:forum.unity.com/threads/solved-deserialize-json-in-unity-c.932298*/
public class ParseReceivedData
{
    public Dictionary<string, string> response_get;

    public static ParseReceivedData CreateFromJSON(string jsonString)
    {
        return JsonUtility.FromJson<ParseReceivedData>(jsonString);
    }

    public void PrintOutput()
    {
        foreach (KeyValuePair<string, string> keyValuePair in this.response_get)
        {
            Debug.Log(keyValuePair.Key + ": " + keyValuePair.Value);
        }
    }
}
```

8 - Verilerin Ekrana Getirlilmesi (2)

Projede, JSON formatındaki verilerin işlenebilmesi için ParseReceviedData adında bir nesne oluşturuldu. Bu nesne, aldığı verileri C#'ın Dictionary yapısını kullanarak anahtar-

değer çiftleri şeklinde saklamayı hedefledi. Ancak, bağlantı kurulmaya çalışıldığında (endpoint belirtildikten sonra) verilerin alınamaması ve uygulamanın hata vermesi, projenin beklenen şekilde çalışmadığını gösteriyor.

Bu aşamada, projenin bağlantıda yaşadığı sorunlar, veri işleme ve entegrasyon sürecinde zorluklar doğuruyor. JSON formatındaki verilerin işlenememesi, belirli bir bağlantı noktasında veya iletişim hatasında kaynaklanıyor olabilir. Bağlantı sağlanamadığında ise uygulama çalışamaz duruma geçiyor.

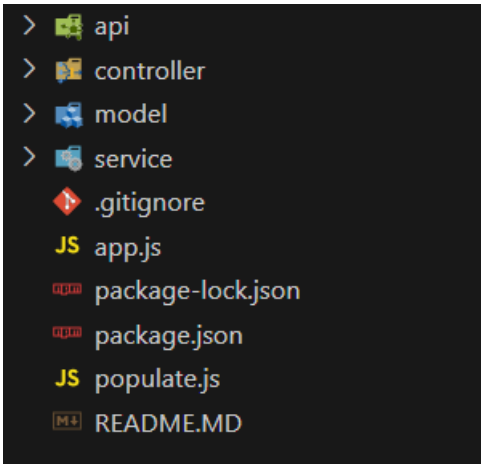
Özellikle, veri işleme aşamasında yaşanan sorunlar, projenin istenen şekilde çalışmamasına neden oluyor. JSON formatındaki verilerin doğru bir şekilde işlenememesi, uygulamanın istenilen verilere erişememesine ve hatalarla karşılaşmasına yol açıyor.

Bağlantı problemleri ve entegrasyon zorlukları, yazılım geliştirme süreçlerinde sıkça karşılaşılabilen durumlardır. Özellikle, projenin farklı bileşenleri arasında sağlanması gereken veri akışının kesintiye uğraması, projenin işlevselliğini etkileyebilir.

Sonuç olarak, projenin veri işleme süreçlerinde yaşanan bağlantı problemleri, JSON verilerinin doğru şekilde işlenememesine ve uygulamanın hatalarla karşılaşmasına neden oluyor. Bağlantı sorunları ve entegrasyon zorlukları, projenin istenen şekilde çalışmasını engelleyerek, geliştirme sürecinde dikkat gerektiren bir konu olarak öne çıkıyor. Bu tür zorlukların çözülmesi, projenin başarılı bir şekilde entegre olmasını ve istenen verileri doğru şekilde işlemesini sağlayabilir.

III. BACKEND

A. *Genel Dosyalama Yapısı*



9 - Genel Klasör Yapısı

Günümüz eğitim sistemi, teknolojinin sunduğu imkanları kullanarak sürekli olarak dönüşüyor. Bu bağlamda, eğitim yönetimi için geliştirilen bir Node.js projesi, API, controller, model (MongoDB), service klasörleri ve diğer temel bileşenlerle birlikte, projenin altyapısını oluşturuyor.

Bu proje, eğitim kurumları için geliştirilmiş bir backend uygulaması olarak öne çıkıyor. Temel bileşenler arasında bulunan API, projenin dış dünyayla iletişimini sağlayan bir arayüz olarak hizmet ediyor. Controller, gelen istekleri yöneterek uygun işlemleri gerçekleştiren modüllerden oluşuyor. Model klasörü, MongoDB veritabanı için veri yapısını tanımlayan ve işleyen yapıları içeriyor. Service klasörleri ise iş mantığı ve veri işleme süreçlerini yöneten, modüllerin işlevselliğini sağlayan önemli bileşenlerden biri olarak karşımıza çıkıyor.

Projede yer alan app.js dosyası, projenin ana giriş noktası olup, farklı modülleri bir araya getirerek çalışmasını sağlar. Package-lock.json ve package.json dosyaları ise projede kullanılan paketlerin ve bağımlılıkların listesini içerir. Readme dosyası ise projenin tanıtımı, nasıl kullanılacağı veya kurulacağı gibi temel bilgileri içerir.

Node.js ve MongoDB tabanlı backend projesi, eğitim yönetimi için esneklik, verimlilik ve özelleştirilebilirlik sağlar. Veri tabanıyla etkileşimde bulunarak çeşitli veri işleme süreçlerini yönetir ve dış dünya ile iletişim kurar. Bu sayede, eğitim kurumları kendi ihtiyaçlarına göre özelleştirilmiş çözümler elde edebilirler.

Sonuç olarak, Node.js ve MongoDB temelli backend projesi, eğitimde teknolojinin gücünü kullanarak daha verimli, kullanıcı dostu ve özelleştirilebilir bir yönetim sistemini mümkün kılar. API, controller, model, service klasörleri ve diğer bileşenler, projenin sağlam altyapısını oluşturur ve eğitim yönetimi için gerekli olan çeşitli işlevselliği sunar.

B. *App.js*

İlk olarak, gerekli modüller çağrılıyor. api, express, mongoose gibi modüller projede kullanılacak fonksiyonları ve yapıları içerirken, cors ve dotenv ise uygulamanın güvenlik ve konfigürasyon ayarlarını yönetmeye yardımcı olur.

```
const api = require('./api/index');
const express = require("express");
const mongoose = require("mongoose");
const app = express();
const cors = require("cors");
require("dotenv").config();
```

10 - App.js dosyası

Bu bölümde, express framework'ü kullanarak bir uygulama oluşturulur ve gerekli modüller tanımlanır. api değişkeni, projenin rotalarını içeren bir dosyaya yönlendirme yapar. Ardından, MongoDB veritabanına bağlanma işlemi gerçekleştirilir.

```
// middleware
const corsOptions = {
    origin: "http://localhost:3000" // frontend URI (ReactJS)
}
app.use(express.json());
app.use(cors(corsOptions));

// connect MongoDB
mongoose.connect(process.env.MONGODB_URI).then(() => {
    const PORT = process.env.PORT || 8000
    app.listen(PORT, () => {
        console.log(`App is Listening on PORT ${PORT}`);
    })
}).catch(err => {
    console.log(err);
});
```

11 - App.js Dosyası

Middleware'ler eklenir: express.json() ile gelen isteklerin JSON formatında işlenmesi sağlanır. cors middleware'i, belirtilen origin değerine sahip istekleri kabul eder. Ardından MongoDB veritabanı bağlantısı yapılır. Bağlantı başarılı olduğunda, belirtilen port üzerinden uygulamanın dinlemesi sağlanır. Bağlantı hatası durumunda ise hatanın konsola yazdırılması sağlanır.

```
// route
app.use('/', api);
```

12 - App.js Dosyası

Son olarak, belirlenen rotaları ('/') api değişkenine yönlendirir. Bu sayede, gelen istekler ilgili rotalara yönlendirilerek işlenir ve istenen işlemler gerçekleştirilir.

Bu kod, bir Node.js uygulamasının başlatılması ve temel ayarlarının yapılmasını içerir. Express, MongoDB ve diğer modüllerin kullanımıyla birlikte, bir backend uygulamasının temel yapı taşlarını oluşturur ve istemcilere hizmet vermek için hazır hale getirir.

C. API Klasörü

1) index.js

Bu kod, bir Express.js uygulamasında rotaları yönetmek için kullanılan Router modülünü tanımlar. express.Router() ile oluşturulan bir router nesnesi, belirli rotalara yönlendirmeler yapar ve istemcilere cevaplar döndürür.

```
const express = require("express")
const courses = require("./courses")
const instructor = require("./instructor")
const { createWeeklySchedule } = require("../controller/sllybusController")

const router = express.Router()
```

13 - index.js

Kodun bu bölümünde, express modülü çağrılır ve Router modülü kullanılarak router adında bir router nesnesi oluşturulur. Ardından, farklı rotalara yönlendirmeler yapacak olan courses, instructor ve createWeeklySchedule gibi fonksiyonlar veya dosyalar çağrılır ve ilgili değişkenlere atanır.

```
router.get("/", (req, res) => res.end("hello"))
router.get("/sllybus", createWeeklySchedule)
```

14 - index.js

Bu kısımda, oluşturulan router nesnesi üzerinde HTTP GET isteklerine yanıt verecek rotalar tanımlanır. Örneğin, / rotası bir GET isteği alır ve basit bir "hello" mesajıyla yanıt döndürürken, /sllybus rotası createWeeklySchedule fonksiyonuna yönlendirilerek ilgili işlev gerçekleştirilir.

```
router.use("/api/course", courses)
router.use("/api/instructor", instructor)
```

15 - index.js

Bu bölümde ise, /api/course ve /api/instructor rotaları belirli dosya veya fonksiyonlarla eşleştirilir. courses ve instructor değişkenleri, ilgili rotalara gelen isteklerin yönlendirileceği dosyaları veya fonksiyonları temsil eder. Bu sayede, ilgili rotalara gelen istekler ilgili dosyalara veya fonksiyonlara iletilir ve işlenir.

```
module.exports = router
```

16 - index.js

Son olarak, oluşturulan router nesnesi module.exports aracılığıyla dışa aktarılır. Bu sayede, bu dosya başka dosyalar tarafından require() fonksiyonuyla çağrılarak kullanılabilir hale gelir. Bu modül, ana uygulama dosyasında ilgili yere eklenerek rotaların yönetilmesini ve istemcilere cevap döndürülmesini sağlar.

2) instructor.js

```
const express = require("express")

const router = express.Router()

const {
  createInstructor,
  getInstructorByinstructorId,
  updateInstructor,
  deleteInstructor,
  getAllInstructors,
} = require("../controller/instructorController")

router.get("/", getAllInstructors)
router.post("/", createInstructor)
router.get("/:id", getInstructorByinstructorId)
router.post("/:id", updateInstructor)
router.delete("/:id", deleteInstructor)

module.exports = router
```

17 - instructor.js

Bu kod parçası, Express.js kullanarak rotaları yönetmek için Router modülünü kullanan bir dosyayı tanımlar. express.Router() ile oluşturulan router nesnesi, farklı HTTP istek metotlarını belirli işlevlere yönlendirir. instructorController dosyasından çağrılan fonksiyonlar, ilgili isteklerin karşılanmasını ve işlenmesini sağlar.

Bu kod bloğu içinde, HTTP GET, POST, PUT ve DELETE metotlarına karşılık gelen istekler için rotalar tanımlanmıştır. Örneğin, router.get("/", getAllInstructors) ifadesi, kök dizinine yapılan GET isteğini getAllInstructors işlevine yönlendirir ve tüm eğitmenleri getirir. Benzer şekilde, router.post("/", createInstructor) ifadesi, kök dizinine yapılan POST isteğini createInstructor işlevine yönlendirir ve yeni bir eğitmen oluşturur.

Her bir istek metodu belirli bir URL yapılandırması ile ilişkilendirilir. Örneğin, router.get("/:id", getInstructorByinstructorId) ifadesi, /:id parametresine sahip bir GET isteğini getInstructorByinstructorId işlevine yönlendirir ve belirli bir eğitmeni ID'ye göre getirir.

Son olarak, oluşturulan router nesnesi, module.exports ile dışa aktarılır. Bu sayede, bu dosya başka bir dosya tarafından çağrılarak kullanılabilir ve uygulamanın rotalarının yönetimini ve işlenmesini sağlar.

3) *courses.js*

```
const express = require("express")

const router = express.Router()

const {
  createCourse,
  getCourseByCourseId,
  updateCourse,
  deleteCourse,
  getAllCourses,
} = require("../controller/courseController")

router.get("/", getAllCourses)
router.post("/", createCourse)
router.get("/:id", getCourseByCourseId)
router.post("/:id", updateCourse)
router.delete("/:id", deleteCourse)


module.exports = router;
```

18 - courses.js

Bu kod, Express.js'in sağladığı Router modülü kullanılarak rotaların yönetildiği bir dosyayı tanımlar. express.Router() ile oluşturulan router nesnesi, farklı HTTP istek metodlarına karşılık gelen rotaları belirli işlevlere yönlendirir. courseController dosyasından çağrılan fonksiyonlar, ilgili isteklerin işlenmesini sağlar.

Her bir istek metodu belirli bir URL yapılandırması ile eşleştirilir. Örneğin, router.get("/", getAllCourses) ifadesi, kök dizine yapılan GET isteğini getAllCourses işlevine yönlendirir ve tüm dersleri getirir. Benzer şekilde, router.post("/", createCourse) ifadesi, kök dizine yapılan POST isteğini createCourse işlevine yönlendirir ve yeni bir ders oluşturur.

Her bir işlev, belirli bir işlemi gerçekleştirmek üzere tanımlanmıştır. Örneğin, getCourseByCourseId, belirli bir kursun ID'sine göre getirilmesi için tasarlanmıştır ve /:id parametresine sahip bir GET isteği ile çalışır.

Son olarak, oluşturulan router nesnesi, module.exports ile dışa aktarılır. Bu sayede, bu dosya başka bir dosya tarafından çağrılarak kullanılabilir ve uygulamanın rotalarının yönetimini ve işlenmesini sağlar.

D. *Controller Klasörü*

```
const courseService = require("../service/courseService")

exports.createCourse = async (req, res) => {
  try {
    const course = await courseService.createCourse(req.body)
    res.json({ data: course, status: "success" })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
}

exports.getCourseByCourseId = async (req, res) => {
  try {
    const course = await courseService.getCourseByCourseId(req.params.id)
    res.json({ data: course, status: "success" })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
}

exports.updateCourse = async (req, res) => {
  try {
    const course = await courseService.updateCourse(req.params.id, req.body)
    res.json({ data: course, status: "success" })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
}

exports.deleteCourse = async (req, res) => {
  try {
    const course = await courseService.deleteCourse(req.params.id)
    res.json({ data: course, status: "success" })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
```

19 - Örnek Bir Controller

Bu kod, bir Node.js uygulamasında farklı isteklere yanıt veren fonksiyonları içeren bir dosyayı tanımlar. Her bir fonksiyon, ilgili isteğe karşılık gelen işlevleri gerçekleştirir ve veritabanı işlemlerini courseService, instructorService ve sllybusService gibi ilgili servis dosyaları aracılığıyla gerçekleştirir.

createCourse, getCourseByCourseId, updateCourse, deleteCourse ve getAllCourses fonksiyonları, derslerle ilgili HTTP isteklerine yanıt verir. Her biri, ilgili istek metoduna göre ilgili servis fonksiyonlarını çağırır ve istenen işlemi gerçekleştirir. Benzer şekilde, createInstructor, getInstructorByinstructorId, updateInstructor, deleteInstructor ve getAllInstructors fonksiyonları da eğitmen verileriyle ilgili işlemleri gerçekleştirir.

Son olarak, createWeeklySchedule fonksiyonu, haftalık ders programı oluşturma işlemlerini yönetir. Bu fonksiyon, sllybusService ile işbirliği içinde çalışarak ilgili işlemi gerçekleştirir.

Her bir fonksiyon, try-catch bloğu içinde bulunur. try bloğu içinde ilgili servis fonksiyonu çağrılır ve başarılı bir şekilde işlem tamamlanırsa istemciye JSON formatında bir yanıt döndürülür. Eğer bir hata oluşursa, catch bloğu içinde hata durumuyla birlikte 500 hatasıyla birlikte istemciye geri dönüş yapılır (res.status(500).json({ error: error.message })). Bu yapı, istemciden gelen isteklere yanıt verirken hata durumlarında uygun bir geri bildirim sağlayarak güvenilir bir API sağlar.

E. Model Klosörü

```
const mongoose = require('mongoose');

const {Schema} = mongoose;

const instructorSchema = new Schema({
  isim: { type: String, required: true },
  haftaici_musait_gunler: { type: Array, default: [] },
  saat_araliklari: { type: Array, default: [] }
});

  module.exports = mongoose.model("Instructor", instructorSchema)
```

20 - Örnek Bir Model Dosyası

Bu kod parçası, MongoDB veritabanında kullanılacak olan veri modellerini oluşturan Mongoose şemasını tanımlar. Her bir şema, belirli bir veri tipi ve şema yapılarıyla ilgili gereksinimleri içerir.

courseSchema, bir kursun temsil edildiği şemayı oluşturur. Bu şema, dersin adı (ders_ismi), öğrenci sayısı (ogrenci_sayisi), sınıf bilgisi (sinif) ve kursu yöneten eğitmenin ID'si (egitmen_id) gibi özellikleri içerir. Her özellik, belirli bir veri tipi ve zorunluluk durumu (required: true) ile belirtilir. egitmen_id özelliği, Instructor adlı başka bir şemaya referans yapar.

instructorSchema ise bir eğitmenin temsil edildiği şemayı tanımlar. Bu şema, eğitmenin adı (isim), hafta içi müsait günleri (haftaici_musait_gunler) ve saat aralıkları (saat_araliklari) gibi özellikleri içerir. haftaici_musait_gunler ve saat_araliklari özellikleri, dizi şeklinde veri saklamak üzere tanımlanmıştır ve varsayılan olarak boş bir dizi olarak atanmıştır.

Her iki şema da, Mongoose'un model metodu kullanılarak bir MongoDB koleksiyonu oluşturmak üzere dışa aktarılır (module.exports = mongoose.model(...)). Bu şemalar, MongoDB'de ilgili koleksiyonları temsil eder ve bu şemalara uygun verilerin saklanmasını sağlar. Bu yapı, veritabanında tutulacak verilerin yapısını ve ilişkilerini belirleyerek veritabanı işlemlerini düzenler.

F. Service Klasörü

```
const Course = require("../model/course")
const Instructor = require("../model/instructor")

exports.createSllybus = async () => {
  const findCourseOrder = (numCourses, prerequisites) => {
    let countIncomingCounts = {}
    let result = []
    let possibleQueue = []
    let graph = {}

    for (let x = 0; x < prerequisites.length; x++) {
      let i = prerequisites[x]

      if (graph[i[1]] != null) {
        graph[i[1]].push(i[0])
      } else {
        graph[i[1]] = [i[0]]
      }

      if (countIncomingCounts[i[0]]) {
        countIncomingCounts[i[0]]++
      } else {
        countIncomingCounts[i[0]] = 1
      }
    }

    for (let i = 0; i < numCourses; i++) {
      if (!countIncomingCounts[i] || countIncomingCounts[i] == 0) {
        possibleQueue.push(i)
      }
    }

    while (possibleQueue.length !== 0) {
      let top = possibleQueue.shift(0)
      result.push(top)
```

21 - Örnek Bir Service Dosyası

Bu kod bloğu, belirli veri modelleri üzerinde veritabanı işlemlerini gerçekleştiren fonksiyonları içeren bir dosyayı tanımlar. courseModel ve instructorModel üzerinde Mongoose modelleri ile ilgili CRUD (Create, Read, Update, Delete) işlemlerini gerçekleştirir.

Her bir fonksiyon, ilgili veritabanı modelinde tanımlanan metotları kullanarak belirli işlemleri gerçekleştirir. Örneğin, createCourse, findCourseByID, updateCourse, deleteCourse ve getAllCourses, courseModel üzerinde ilgili işlemleri gerçekleştirirken, createInstructor, findInstructorByID, updateInstructor, deleteInstructor ve getAllInstructors fonksiyonları ise instructorModel üzerinde ilgili işlemleri gerçekleştirir.

Ayrıca, createSllybus fonksiyonu, belirli bir dönemde alınabilecek derslerin haftalık bir programını oluşturan karmaşık bir algoritma içerir. Bu fonksiyon, alınan derslerin önceki derslerinin belirlenmesi, müsaitlik durumları ve saat aralıklarına göre haftalık bir program oluşturur. Oluşturulan program, haftalık ders programı formatında weeklySchedule olarak döndürülür. Bu işlem sırasında, olası hatalar try-catch blokları içinde yönetilir ve hata durumunda boş bir nesne veya hata mesajı döndürülür. Bu algoritma, verilen koşullara göre ders programı oluştururken, eğitmenlerin müsaitlik durumları ve derslerin önceki derslerine bağlı olarak bir planlama yapar.

G. *App.js Dosyası*

```
const api = require('./api/index');
const express = require("express");
const mongoose = require("mongoose");
const app = express();
const cors = require("cors");
require("dotenv").config();

// middleware
const corsOptions = {
    origin: "http://localhost:3000" // frontend URI (ReactJS)
}
app.use(express.json());
app.use(cors(corsOptions));

// connect MongoDB
mongoose.connect(process.env.MONGODB_URI).then(() => {
    const PORT = process.env.PORT || 8000
    app.listen(PORT, () => {
        console.log(`App is Listening on PORT ${PORT}`);
    })
}).catch(err => {
    console.log(err);
});

// route
app.use('/', api);
```

22 - App.js Dosyası

Bu kod, bir Node.js uygulamasında kullanılan ana geliştirme dosyasını temsil eder. Öncelikle gerekli modüllerin import edilmesi yapılır; bunlar arasında express, mongoose (MongoDB için), cors (çapraz kaynak isteği engelleme) gibi modüller yer alır. Ardından, express uygulaması oluşturulur ve gerekli middleware'ler (express.json() ve cors) kullanılır.

corsOptions nesnesi, gelen isteklerin hangi kaynaklardan gelmesine izin verileceğini belirtir. Bu durumda, sadece http://localhost:3000 adresinden gelen isteklere izin verilir.

Daha sonra MongoDB'ye bağlanılır (mongoose.connect) ve bağlantı başarılı olduğunda belirtilen port numarasında (process.env.PORT veya varsayılan olarak 8000) bir HTTP sunucusu başlatılır (app.listen). Bağlantı başarısız olursa, hata konsola yazdırılır.

Son olarak, app.use('/', api) ifadesi, rotaları yönlendirmek için api dosyasını kullanır. Bu, uygulamanın ana rotasına gelen istekleri api dosyasına yönlendirir. Bu dosya genellikle farklı rotaları ve bu rotalara karşılık gelen işlevleri içerir. Bu yapı, istemci tarafından gönderilen istekleri ilgili yönlendirmelere göre işlemek ve yanıtlamak için kullanılır.

IV. DATABASE

```
// dersler koleksiyonu şeması
const derslerSchema = new Schema({
  ders_ismi: { type: String, required: true },
  ogrenci_sayisi: { type: Number, required: true },
  sinif: { type: String, required: true },
  egitimci_id: { type: mongoose.Types.ObjectId, ref: "eğitmenler" }
});

// eğitmenler koleksiyonu şeması
const eğitmenlerSchema = new Schema({
  isim: { type: String, required: true },
  haftaici_musayit_gunler: { type: Array, default: [] },
  saat_araliklari: { type: Array, default: [] }
});
```

23 - DataBase Kodları

Veritabanındaki şemalar, MongoDB'deki koleksiyonların yapısını tanımlayan yapıları ifade eder. İlgili kod parçaları, dersler ve eğitmenler adlı iki farklı koleksiyonun şemalarını oluşturur.

"Dersler" koleksiyonu şeması, her bir dersin belirli özelliklerini tanımlar. Bu özellikler şunlardır:

- ders_ismi: Dersin adı, String veri tipinde ve zorunlu bir alan (required: true).
- ogrenci_sayisi: Dersin öğrenci sayısı, Number veri tipinde ve zorunlu bir alan.
- sinif: Dersin yapıldığı sınıf bilgisi, String veri tipinde ve zorunlu bir alan.
- egitimci_id: Dersi veren eğitmenin ID'si, mongoose.Types.ObjectId veri tipinde ve "Eğitmenler" koleksiyonuna referans veren bir alan (ref: 'Eğitmenler').

"Eğitmenler" koleksiyonu şeması ise eğitmenlerin özelliklerini tanımlar. Bu özellikler şunlardır:

- isim: Eğitmenin adı, String veri tipinde ve zorunlu bir alan.
- haftaici_musayit_gunler: Eğitmenin hafta içi müsait olduğu günlerin listesi, Array veri tipinde ve varsayılan olarak boş bir liste (default: []).
- saat_araliklari: Eğitmenin müsait olduğu saat aralıklarının listesi, Array veri tipinde ve varsayılan olarak boş bir liste (default: []).

Bu yapılar, MongoDB'de tutulan veritabanının "dersler" ve "eğitmenler" adlı iki koleksiyonunun hangi özellikleri içereceğini belirler. Derslerin öğrenci sayısı, ders adı gibi bilgileri saklamak için dersler koleksiyonu, eğitmenlerin adı, müsait olduğu günler ve saat aralıkları gibi bilgileri saklamak için ise eğitmenler koleksiyonu kullanılır. Bu şemalar, veritabanındaki verilerin yapısını ve ilişkilerini tanımlar.

V. FRONTEND

A. *Front End ve Next js*

Next.js, React tabanlı bir JavaScript framework'ü olarak modern web uygulamalarının geliştirilmesine odaklanmıştır. Front-end geliştirmede kullanımı, bir dizi avantaj ve özellik sunar. Sayfa yönlendirmesi kolaylığı ile URL yönlendirmeleri basitçe gerçekleştirilebilir. Aynı zamanda kodu bölerek yükleme işlemlerini optimize eder, böylece sayfa yükleme sürelerini iyileştirir. Sunucu tarafında render edilirken dosyaların statik dosyalara dönüşmesini sağlayarak performans artışı sağlar. API desteği ile backend entegrasyonu oldukça kullanıcı dostudur.

Next.js, CSS modüllerini destekleyerek bileşen bazlı stil yönetimini kolaylaştırır. Bu özellik, stil çakışmalarını engeller ve modüler CSS kullanımını teşvik eder. Her bir bileşen kendi stilini içerir, böylece stil değişiklikleri belirli bir bileşeni etkiler ve genel stil yapılarını etkilemez. Bu, daha düzenli ve bakımı kolay bir stil yönetimi sağlar. Overall, Next.js, React ile birleştirerek geliştiricilere performans, entegrasyon kolaylığı ve stil yönetimi konusunda avantajlar sunar.

B. *Genel Sayfa Kısmı*

1) *Modal.jsx*

24 - Modal Sayfası

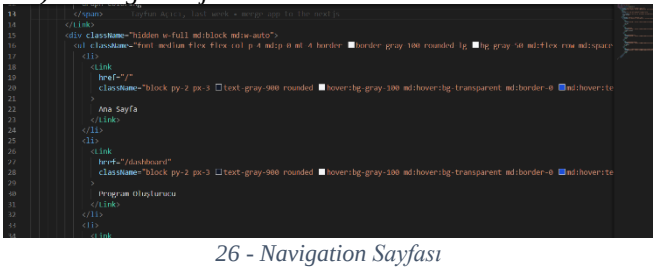
Projemizin front end kısmında react bağımlıklarını projemize dahil ettik. Daha sonrasında görünürlük durumuna bağlı olarak bir arayüz elemanını kullanıcının etkileşimine sunar. Kapatma butonu ve dışarı tıklandığında kapatma özellikleri, kullanıcı deneyimini geliştirmek için eklenmiştir. Bu yapı Bileşenin modüler yapısı, kullanım esnekliği sağlar ve uygulama içinde tekrar kullanılabilirliği artırır. Bu da kullanılabilirlik durumunu arttırır.

2) Layout.jsx

25 - Layoyt Sayfası

Projemizin bu kısmında gerekli font ayarlamalarını yapmaya devam ettik ve içinde HTML içeriklerini döndürebileceğimiz bir yapı kurmaya çalıştık. Yaptığımız Nextjs uygulamasında temel sayfa düzenini ve font ayarlarını yaparken Inter kütüphanesi ile fontu entegre ettik. RootLayout fonksiyonu ise temel HTML belgesi yapısını oluşturur ve sayfa içeriğini düzenler. Bu, genel bir sayfa şablonunu tanımlamak ve sayfa düzenini standartlaştırmak amacıyla kullanılabilir.

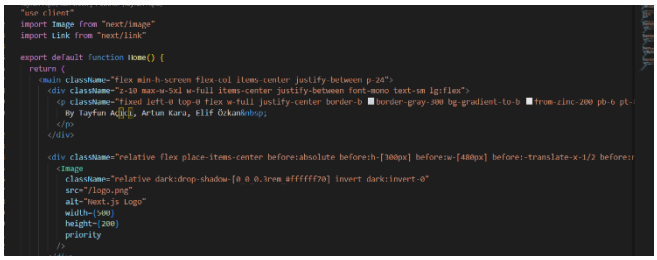
3) Navigation.jsx



26 - Navigation Sayfası

Projemizin bu kısmında kullanacağımız navigasyon kısmını tasarlamak istedik. Bileşen, genel bir sayfa düzeni oluşturur ve bağlantılar arasında gezinmeyi sağlar. Responsive tasarım, renk temaları ve stil özellikleri, kullanıcıların uygulama içinde etkileşimde bulunmasını kolaylaştırmak için kullanılmıştır. Next js kullanarak tasarladığımız projemizin bu kısmında sayfalar arası kolayca gezinmeyi kolaylaştırmak için 'Link' modülünü entegre ettik. Daha sonrasında Navigation fonksiyonun içine tanımladığımız <nav> elementi ile arka plan rengini, sınırları ve temaları belirtmek istedik. Fonksiyonun içinde tanımladığımız <Link> bileşeni ise başlık ve ana sayfa bağlantısını tanımlamamızı sağlıyor . Bu bileşeni tanımladığımız bir diğer kod bölümünde ise bağlantı verirken ve bağlantı görünümünü tanımlarken kullandık.

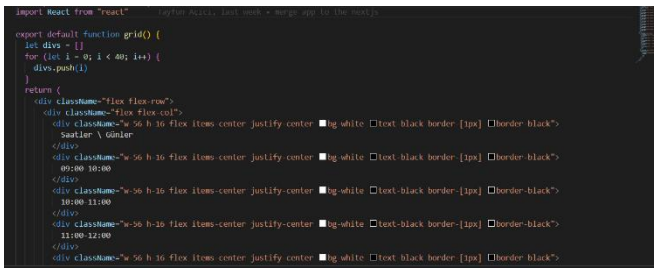
4) Page.jsx



27 - Page Sayfası

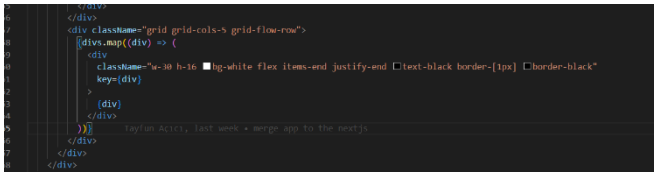
Projemizin bu kısmında ana sayfa bileşenimizi oluşturmak istedik. Sayfamız görsel öğeler, menü bağlantıları ve responsive tasarım içeren özellikler bulundurmaktadır. Öncelikle kullandığımız <main> etiketi ile sayfamızın ana içeriğini oluşturmaya başladık. Verdiğimiz className ile esnek bir ekran görüntüsü yaratmak istedik. Daha sonraki tasarım eklentileri ve bağlantıları ise kullandığımız etiketleri ve <Link> etiketleri ile sağlamak istedik. Bununla da kalmayarak sağladığımız bağlantılar ile sayfada yönlendirme yapılmasını sağladık.

C. Dashboard Kısmı



28 - Grid.jsx

Projemizin bu kısmında haftalık ders programını gösteren bir tablo oluşturmak istedik. Tanımladığımız for döngüsü ile 40 elemanlı bir dizi oluşturarak, programın saat aralıklarını temsil eden numaraları içermesini sağladık. Ana bileşen yapımızda bulunan etiketlerine flex özelliği vererek yatay bir görünüm oluşmasını sağladıktan sonra saatler ve günler için ayrı alanlar tanımladık ve onlara stil özellikleri verip renklendirdik.



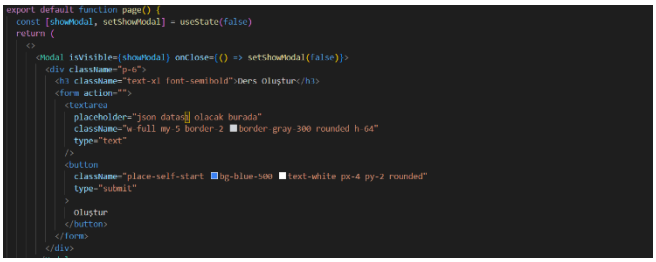
29 - Grid.jsx

Bu kısımda kullandığımız map fonksiyonu ile her saat aralığı için bir kutucuk oluşturduk. Genel anlamda bu bölümde saat aralıklarını ve ders numaralarını gösterdik.

D. *Dersler ve Eğitmenler*

30 - Grid.jsx

Projemizin bu bölümünde page() fonksiyonu ile hocaların ismi ve verdiği dersleri göstermek istedik. Önce tablomuzun başlığını tanımladık daha sonrasında ise tablo içeriğini ilave ettik.



31 - Grid.jsx

Bu bölüm ise dersler ile alakalı yeni ders eklemek için kullanıcıya bir model penceresi sunar. Showmodal adlı bileşen ile model penceresi görünümü kontrol edilir. İlk varsayılan değeri ise false olarak atanmıştır.

32 - Grid.jsx

Kodun bu kısmıyla ise ders detaylarını göstermek istedik. Tanımladığımız setShowModal() fonksiyonu ile model penceresini görünür hale getirip üzerine gelindiğinde butonun altı çizili şekilde görünmesini sağlıyoruz.

Kaynakça

[1] https://docs.unity3d.com/2020.1/Documentation/Manual/JSONSerialization.html

[2] https://forum.unity.com/threads/solved-deserialize-json-in-unity-c.932298/

[3] https://mongoosejs.com/docs/guide.html

[4] https://nextjs.org/docs

.