Syllabus

<https://github.com/kagantemizkan/GrapColorTimetable>

<https://github.com/kagantemizkan/kou-syllabus>

Eraycan Çoban  
Bilişim Sistemleri Mühendisliği  
Kocaeli ÜniversitesiKocaeli,Türkiye  
[eraycancoban@gmail.com](mailto:eraycancoban@gmail.com)

Kağan Temizkan  
Bilişim Sistemleri Mühendisliği  
Kocaeli ÜniversitesiKocaeli,Türkiye  
k[aganntemizkan@gmail.com](mailto:kaganntemizkan@gmail.com)

Fatih Enes Kılıç  
Bilişim Sistemleri Mühendisliği  
Kocaeli ÜniversitesiKocaeli,Türkiye  
fatiheneskilic9@gmail.com

***Özet*—Welsh-Powell algoritmasının ders programdaki çakışmaları önlemek için nasıl kullanılabileceğini ve bunu web üzerinde uygulanmış halinin açıklandığı akademik bir projenin makalesi.**

***Keywords—graph renklendirme algoritmaları, node.js, welsh-powell, react,tailwind,vite,sql***

# Graph Renklendirme

Graf teorisi, nesneler arasındaki ilişkileri modelleyen bir matematik dalıdır. Graf renklendirme, bir grafa renk atama işlemidir, genellikle şu kurallara tabidir:

* Her bir düğüm (node) yalnızca bir renge sahip olabilir
* Komşu düğümler aynı renge sahip olamaz

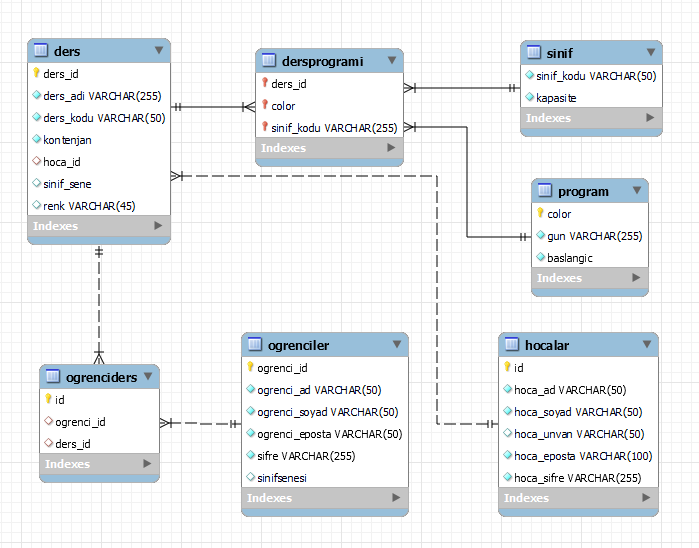
Welsh-Powell algoritması, bir grafın düğümlerini renklendirmek için kullanılan bir algoritmadır. Temel amacı, bir grafa mümkün olan en az sayıda renk kullanarak düğümleri renklendirmektir. Algoritma şu adımları içerir:

* Düğümleri derecelerine göre sırala. (Düğümün derecesi, o düğüme bağlı olan kenar sayısını ifade eder.)
* En yüksek dereceli düğümü ilk renge (renk 1) atayın.
* Bir sonraki düğümü renklendirmek için, henüz renklendirilmemiş düğümler arasından en yüksek dereceli olanı seçin ve önceki renklerle çakışmayacak şekilde bir renk atayın.

Bu işlemi tüm düğümler renklenecek şekilde tekrarlayın.

Bu şekilde, mümkün olan en az sayıda renk kullanarak grafı renklendirmiş olursunuz. Welsh-Powell algoritması, pratikte birçok uygulama alanında kullanılan etkili bir renklendirme algoritmasıdır.

# SQL Şeması



Ders :Derslerin tutulduğu tablo

dersprogrami:Welshpowell sonucu derslerin renklendirilmiş halinin tutulduğu tablo.

ogrenciDers:Hangi öğrencinin hangi dersi aldığını gösteren tablo

ogrenciler:Öğrencilerin bilgisinin tutulduğu tablo

hocalar:Hocaların bilgisinin tutulduğu tablo

sinif:Sınıfların bilgisinin tutulduğu tablo

Program: Renklerin saat-gün kombinasyonu olarak tutulduğu tablo.

# Backend

Backend temelinde 3 parçaya ayrılıyor kullanıcının giriş çıkış işlemleri,kullanıcının ders seçmesi ders eklemesi ve programın renklendirilmesi.Bu kısımda ilk ikisi üzerinden basit bir örnek verirken.Renklendirme kısmında biraz dikkatlice inceleyeceğiz.

## Register,Login,Logout Fonlsiyonları

register fonksiyonu:

export const register = (req, res) => {

const userType = req.body.userType; // Kullanıcı tipini al

// Kullanıcı var mı kontrol et

const q = `SELECT \* FROM ${userType === 'student' ? 'ogrenciler' : 'hocalar'} WHERE ${userType === 'student' ? 'ogrenci\_eposta' : 'hoca\_eposta'} = ?`;

db.query(q, [req.body.email], (err, data) => {

if (err) return res.json(err);

if (data.length) return res.status(409).json("Bu email adresi kullanılıyor");

// Şifreyi hashle

const salt = bcrypt.genSaltSync(10);

const hash = bcrypt.hashSync(req.body.password, salt);

// Kayıt

const insertTable = userType === 'student' ? 'ogrenciler' : 'hocalar';

const q2 = `INSERT INTO ${insertTable} (${userType === 'student' ? 'ogrenci\_ad, ogrenci\_soyad, ogrenci\_eposta, sifre' : 'hoca\_ad, hoca\_soyad, hoca\_eposta, hoca\_unvan, hoca\_sifre'}) VALUES (?)`;

const values = userType === 'student'

? [req.body.ad, req.body.soyad, req.body.email, hash]

: [req.body.ad, req.body.soyad, req.body.email, req.body.unvan, hash];

db.query(q2, [values], (err, data) => {

if (err) return res.json(err);

return res.status(200).json(`${userType === 'student' ? 'Öğrenci' : 'Kullanıcı'} eklendi`);

});

});

};

Kullanıcı tipini (student veya hoca) alır.

Veritabanında bu kullanıcı tipine uygun tabloyu seçmek için SQL sorgusu oluşturur.

Kullanıcının var olup olmadığını kontrol eder.

Eğer kullanıcı varsa, "Bu email adresi kullanılıyor" hatası döner.

Kullanıcı yoksa, şifreyi bcrypt kullanarak hashler.

Ardından kullanıcının türüne bağlı olarak ilgili tabloya yeni kayıt ekler.

Başarılı bir kayıt gerçekleşirse "Öğrenci eklendi" veya "Kullanıcı eklendi" mesajıyla birlikte 200 OK durumuyla bir yanıt döner.

login fonksiyonu:

export const login = (req, res) => {

const userType = req.body.userType; // Kullanıcı tipini al

const database = userType === 'student' ? 'ogrenciler' : 'hocalar'

const emailField = userType === 'student' ? 'ogrenci\_eposta' : 'hoca\_eposta';

const passwordField = userType === 'student' ? 'sifre' : 'hoca\_sifre';

const q = `SELECT \* FROM ${database} WHERE ${emailField} = ?`;

db.query(q, [req.body.email], (err, data) => {

if (err) return res.json(err);

// Kullanıcı bulunamadığında

if (data.length === 0) {

return res.status(404).json("Kullanici bulunamadi");

}

const isPasswordCorrect = bcrypt.compareSync(

req.body.password,

data[0][passwordField]

);

// Parola doğrulama hatası

if (!isPasswordCorrect) {

return res.status(400).json("Parola hatası");

}

const token = jwt.sign({ id: data[0].user\_id }, config.secret, {

expiresIn: 86400,

});

// password alanını filtreleyerek kullanıcı bilgilerini al

const { [passwordField]: password, ...other } = data[0];

res

.cookie("access\_token", token, {

httpOnly: true,

})

.status(200)

.json(other);

});

};

Kullanıcı tipini, veritabanı adını, e-posta alanını ve şifre alanını belirler.

Belirtilen e-posta adresi ile ilgili kullanıcıyı veritabanından sorgular.

Kullanıcı bulunamazsa "Kullanıcı bulunamadi" hatası döner.

Kullanıcı bulunursa, girilen şifre ile veritabanındaki hashlenmiş şifre karşılaştırılır.

Şifre doğru değilse "Parola hatası" hatası döner.

Şifre doğruysa, bir JWT (JSON Web Token) oluşturulur ve kullanıcıya döner. Ayrıca, kullanıcı bilgileri filtrelenerek (şifre hariç) JSON formatında yanıt verilir.

logout fonksiyonu:

export const logout = (req, res) => {

res.clearCookie("access\_token",{

sameSite:"none",

secure:true

}).status(200).json("User has been logged out.")

};

Kullanıcının çıkış yapmasını sağlar.

access\_token adlı çerez temizlenir.

Başarılı bir çıkış gerçekleşirse "User has been logged out." mesajıyla birlikte 200 OK durumuyla bir yanıt döner.

## AddLesson

addLesson fonksiyonu:

export const addLesson=(req,res)=>{

const q="Select \* From ders Where ders\_kodu = ?"

var randomColor = getRandomColor();

db.query(q,[req.body.dersKodu], (err, data) => {

if (err) return res.json(err);

if (data.length) return res.status(409).json("Bu ders zaten eklenmiş");

})

const q2=`INSERT INTO Ders (ders\_adi, ders\_kodu, kontenjan, hoca\_id, sinif\_sene,renk) VALUES (?)`

const values = [

req.body.dersAd,

req.body.dersKodu,

req.body.kontenjan,

req.params.id,

req.body.sene,

randomColor

]

db.query(q2, [values], (err, data) => {

if (err) return res.json(err);

return res.status(200).json("ders eklendi");

})

}

Veritabanında, belirtilen ders koduna sahip bir dersin olup olmadığını kontrol etmek için bir sorgu yapılır (SELECT \* FROM ders WHERE ders\_kodu = ?).

Eğer ders zaten mevcutsa, "Bu ders zaten eklenmiş" hatası ile birlikte 409 Conflict durumuyla bir yanıt döner.

Dersin veritabanına eklenmesi için bir ekleme sorgusu yapılır (INSERT INTO Ders (ders\_adi, ders\_kodu, kontenjan, hoca\_id, sinif\_sene, renk) VALUES (?)).

req.params.id kullanılarak hoca\_id parametresi alınır.

getRandomColor() fonksiyonu, rastgele bir renk üretmek için çağrılır ve bu renk, yeni dersin renk alanına eklenir.

Ders eklenirse, "ders eklendi" mesajı ile birlikte 200 OK durumuyla bir yanıt döner.

Ders eklenirken kullanılan renk, getRandomColor() fonksiyonu tarafından rastgele oluşturulmuş bir renktir. Bu renk,kullanıcının arayüzünde dersi temsil etmek için kullanılır.

Kontenjan, dersin öğrenci alabileceği maksimum sayıyı ifade eder.

Hata durumlarında res.json(err) kullanılarak hata detayları JSON formatında yanıt olarak gönderilir. Bu, hataların daha iyi teşhis edilmesine yardımcı olabilir. Ancak, üretim ortamında hata mesajlarını mümkünse sınırlamak veya genelleştirmek daha güvenli olabilir.

## Welsh-Powell

export const komsulukCikar = (req, res) => {

// Öğrenci komşulukları

const deleteQuery = `Delete from dersprogrami`

db.query(deleteQuery, (err, data) => {

if (err) {

return res.json(err);

}

console.log("silindi")

})

const ogrenciQuery = `

SELECT ogrenci\_id, GROUP\_CONCAT(ders\_id) AS komsuluk

FROM ogrenciders

GROUP BY ogrenci\_id;`;

// Hoca komşulukları

const hocaQuery = `

SELECT hoca\_id, GROUP\_CONCAT(ders\_id) AS komsuluk

FROM ders

GROUP BY hoca\_id;`;

const q = `SELECT COUNT(DISTINCT ders\_id) AS sayi

FROM ders;`

const q2 = `SELECT COUNT(DISTINCT sinif\_kodu) AS sayi

FROM sinif;`

// Her iki sorguyu da paralel olarak çalıştır

db.query(q, (err, data) => {

if (err) {

return res.json(err);

}

const maxSayi = data[0].sayi

db.query(ogrenciQuery, (ogrenciErr, ogrenciData) => {

db.query(hocaQuery, (hocaErr, hocaData) => {

if (ogrenciErr || hocaErr) {

return res.json(ogrenciErr || hocaErr);

}

// Veriyi birleştir

const birlesikVeri = {

ogrenci: ogrenciData,

hoca: hocaData,

};

const q = `Select sinif\_kodu from sinif `

db.query(q, (err, results) => {

if (err) {

return err

}

db.query(q2, (err, data) => {

if (err) {

return res.json(err);

}

const siniflar = results.map((row) => row.sinif\_kodu);

console.log(siniflar)

const sinifSayi = data[0].sayi

const sonuc = kaydedilenSayilariBul(birlesikVeri, maxSayi);

const coloredNodes = welshPowell(sonuc, sinifSayi);

console.log(coloredNodes)

const insertQuery = 'INSERT INTO dersProgrami (ders\_id, color,sinif\_kodu) VALUES ?';

let value=assignClassCodeAndColor(siniflar,coloredNodes)

console.log(value)

const values = Object.entries(value).map(([dersId, { color, sinif\_kodu }]) => [parseInt(dersId), color, sinif\_kodu]);

db.query(insertQuery, [values], (err, sonuc) => {

if (err) {

console.error('Veri eklenirken hata oluştu:', err);

throw err;

}

console.log('Veri başarıyla eklendi');

return res.json("başarı")

// Veritabanı bağlantısını kapat

resolveConflicts()

});

})

})

// Komsulukları hesapla

});

});

})

};

function kaydedilenSayilariBul(veri, maxSayi) {

const kaydedilenSayilar = {};

for (let i = 1; i <= maxSayi; i++) {

kaydedilenSayilar[i] = [];

for (const kategori in veri) {

for (const eleman of veri[kategori]) {

const komsuluk = eleman.komsuluk.split(',').map(Number);

if (komsuluk.includes(i)) {

const digerSayilar = komsuluk.filter((sayi) => sayi !== i);

kaydedilenSayilar[i] = kaydedilenSayilar[i].concat(digerSayilar);

}

}

}

// Duplicateleri temizle

kaydedilenSayilar[i] = [...new Set(kaydedilenSayilar[i])];

}

return kaydedilenSayilar;

}

function welshPowell(graph, maxSameColorNeighbors) {

const sortedNodes = Object.keys(graph).sort((a, b) => graph[b].length - graph[a].length);

const colors = {};

for (const node of sortedNodes) {

const neighborColors = new Set(graph[node].map(neighbor => colors[neighbor]));

let color = 1;

while (neighborColors.has(color) || countSameColorNeighbors(graph[node], colors, color) >= maxSameColorNeighbors) {

color++;

}

colors[node] = color;

}

return colors;

}

// Düğümün belirli bir renkteki komşu sayısını sayan yardımcı fonksiyon

function countSameColorNeighbors(neighbors, colors, color) {

return neighbors.reduce((count, neighbor) => {

return colors[neighbor] === color ? count + 1 : count;

}, 0);

}

function assignClassCodeAndColor(classes, distribution) {

let result = {};

Object.keys(distribution).forEach((key) => {

let randomClassIndex;

do {

randomClassIndex = Math.floor(Math.random() \* classes.length);

} while (result[key]);

result[key] = {

color: distribution[key],

sinif\_kodu: classes[randomClassIndex]

};

});

return result;

}

function resolveConflicts() {

let conflictsExist = true;

while (conflictsExist) {

const conflictQuery = `

SELECT

\*

FROM

dersProgrami

WHERE

(color, sinif\_kodu) IN (

SELECT

color,

sinif\_kodu

FROM

dersProgrami

GROUP BY

color, sinif\_kodu

HAVING

COUNT(\*) > 1

);

`;

const countClassesQuery = `

SELECT COUNT(DISTINCT sinif\_kodu) AS sayi

FROM sinif;

`;

db.query(conflictQuery, (err, results) => {

if (err) {

console.error(err);

return;

}

if (results.length > 0) {

const conflictList = results.map(row => ({ color: row.color, sinif\_kodu: row.sinif\_kodu }));

db.query(countClassesQuery, (err, data) => {

if (err) {

console.error(err);

return;

}

const totalClasses = data[0].sayi;

// Çakışan sınıfları çöz

resolveConflictsInClasses(conflictList, totalClasses);

});

} else {

conflictsExist = false;

}

});

}

}

function resolveConflictsInClasses(conflictList, totalClasses) {

conflictList.forEach(conflict => {

const { color, sinif\_kodu } = conflict;

// Rastgele bir başka sınıf seç

let randomClass ;

do {

randomClass = Math.floor(Math.random() \* totalClasses) + 1;

} while (randomClass === excludedClass);

// Çakışan sınıfı yeni sınıf ile değiştir

const updateQuery = `

UPDATE dersProgrami

SET sinif\_kodu = '${newClass}'

WHERE color = '${color}' AND sinif\_kodu = '${sinif\_kodu}';

`;

db.query(updateQuery, (err, results) => {

if (err) {

console.error(err);

return;

}

console.log(`Conflict resolved: ${sinif\_kodu} in ${color} color replaced with ${newClass}.`);

});

});

}

Öğrenci ve hoca komşulukları temsil eden veritabanındaki ders programı bilgileri (dersprogrami tablosu) silinir.

Öğrenci ve hoca komşulukları ayrı ayrı sorgulanarak, her öğrencinin komşu dersleri (komsuluk) toplanır.

Ders sayısı ve sınıf sayısı belirlenir.

Kaydedilen sayıları bulan kaydedilenSayilariBul fonksiyonu çağrılır.

Welsh-Powell algoritması ile renklendirme yapılır. Her dersin bir rengi olacak şekilde renklendirme yapılır.

Renklenmiş düğümlerle sınıf kodları eşleştirilir ve assignClassCodeAndColor fonksiyonu ile renklenmiş düğümlere rastgele sınıf kodları atanır.

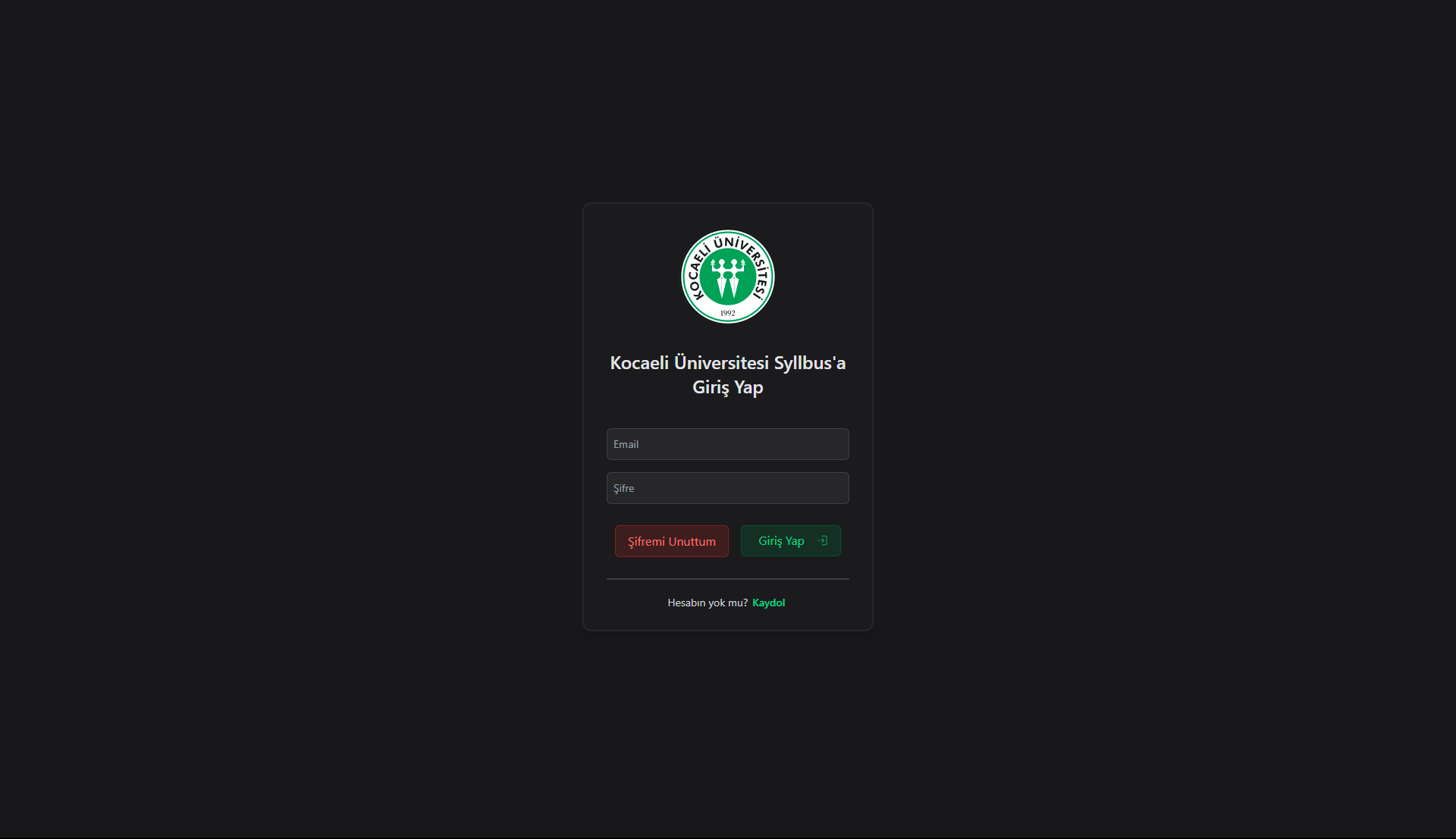
Çakışan sınıflar kontrol edilir ve çözülmesi için resolveConflicts fonksiyonu çağrılır.

Çakışan sınıfların çözülmesi için resolveConflictsInClasses fonksiyonu çağrılır.

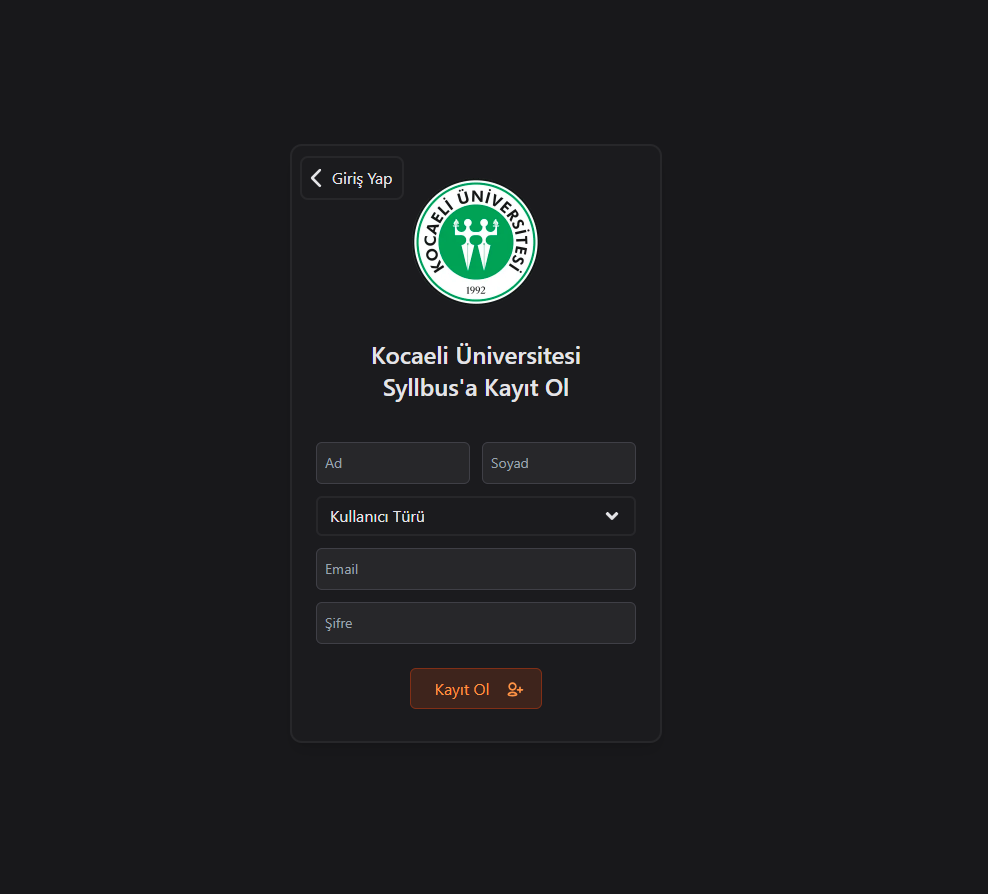
Çözülen veri, dersProgrami tablosuna eklenir.

# FrontEnd

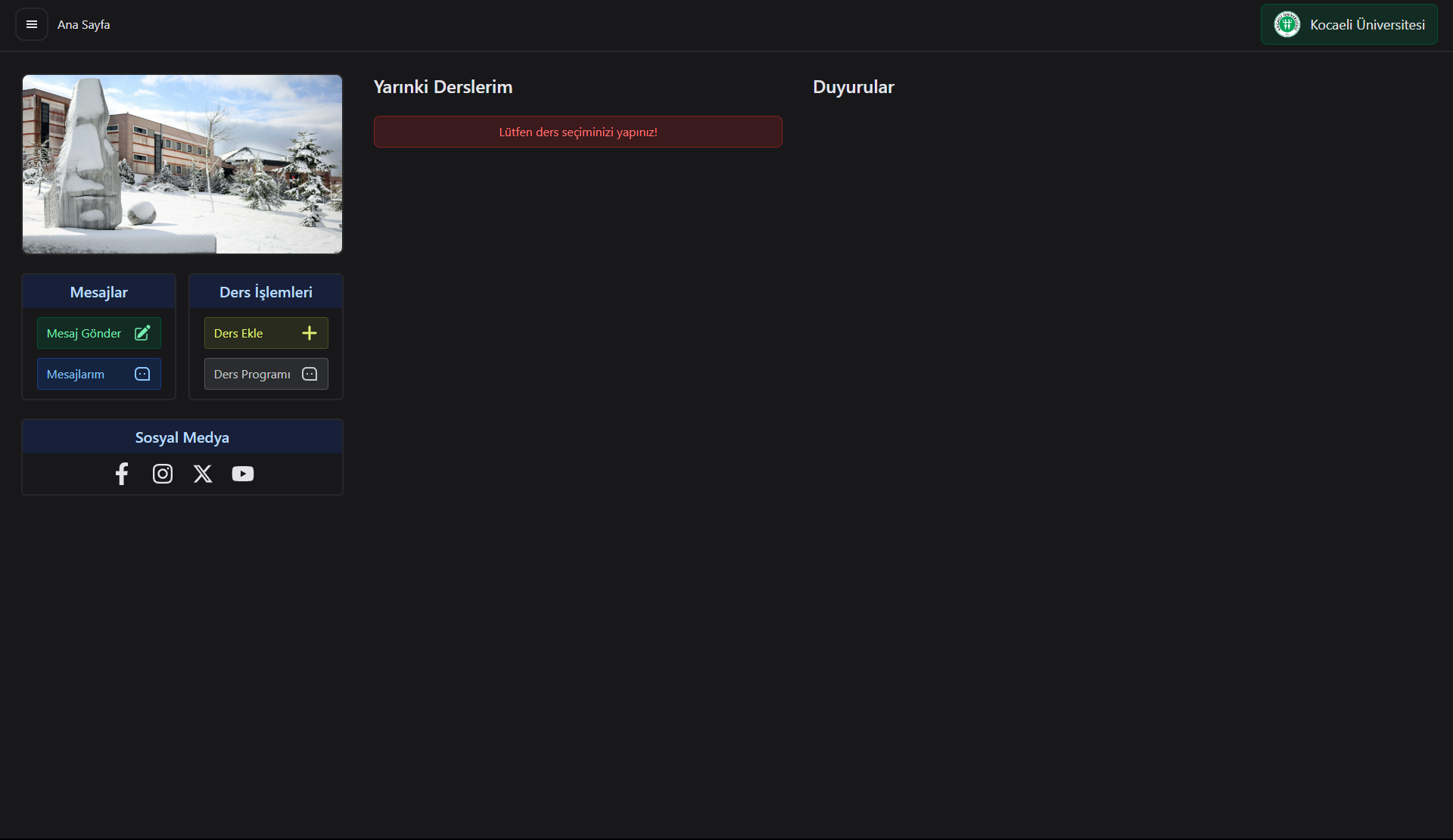
## Login ve Register



Kullanıcılar e-posta ve şifrelerini girerek sisteme giriş yapabilirler.Kayıtları olmadığı takdirde sisteme aşağıda gösterilen sayfadan kayıt olaHbilirler.

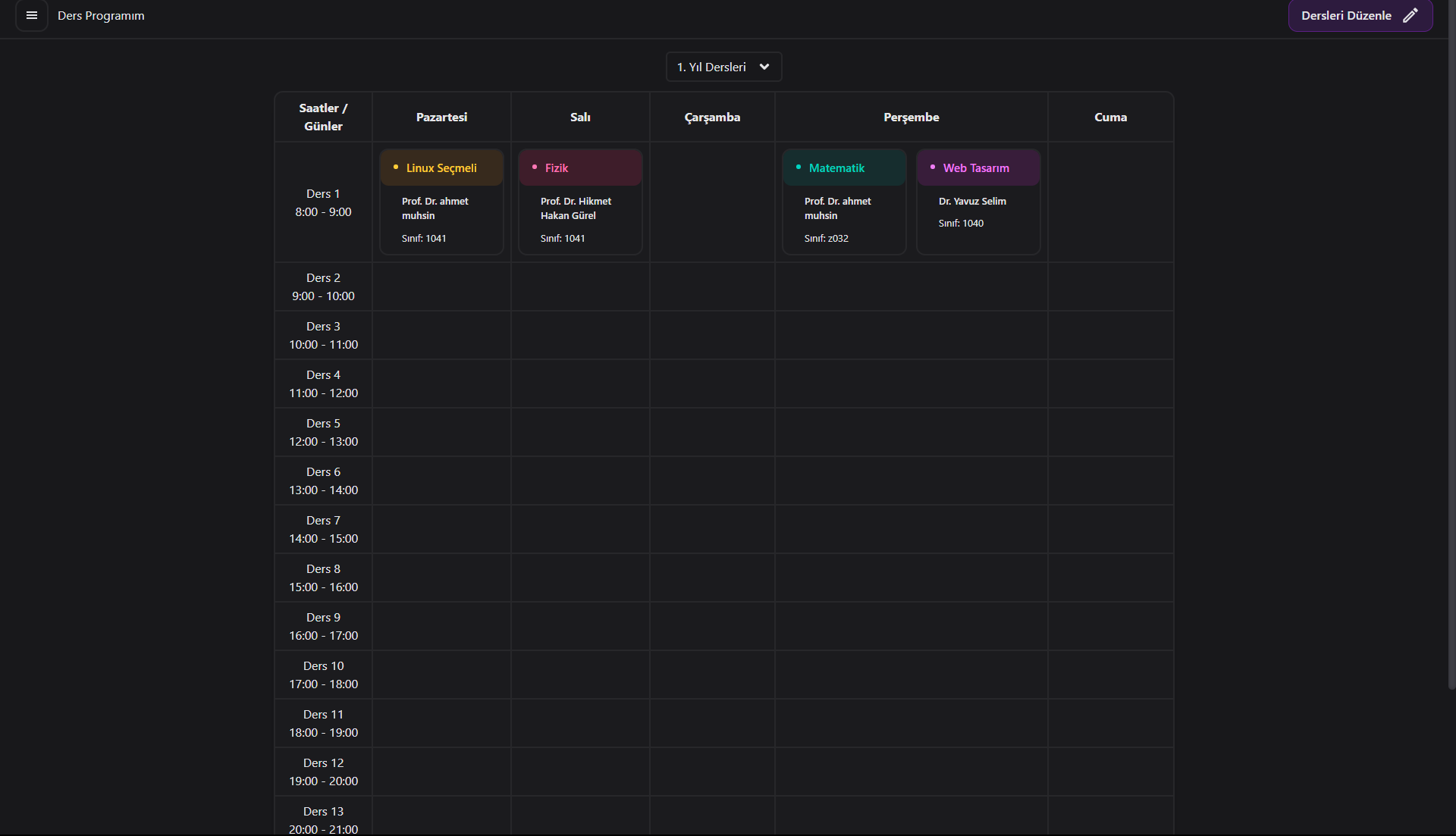


## Anasayfa



Yarınki Derslerim kısmından dersleri görebilir hocaysanız ders ekle kısmından ders ekleyebilirsiniz.

## Program Sayfası ve Ders Seçimi



Buradan sınıf senelerini seçerek ders programlarına bakabilirsiniz.

Ayrıca öğrenciler aşağıdaki gibi ders seçim ekranından ders seçip onaylayabilir.

