# Time Table Creator Application

With graph coloring algorithm

Yunus Emre Gül
Bilişim Sistemleri Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
211307009@kocaeli.edu.tr

Yunus Emre Gölbaşı
Bilişim Sistemleri Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
211307043@kocaeli.edu.tr

Serhat Akçadağ
Bilişim Sistemleri Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
211307032@kocaeli.edu.tr

***Abstract*— The project described in this article involves the development of a timetable generation application using a graph coloring algorithm. Implemented with Next.js, the application aims to efficiently organize class schedules based on a set of predefined constraints. This article provides an overview of the project, detailing its objectives, methodology, and the technologies employed.**

***Keywords—Next.js, Graph Coloring, Time-Table Creator, Mongo DB, Javascript***

## I. GİRİŞ

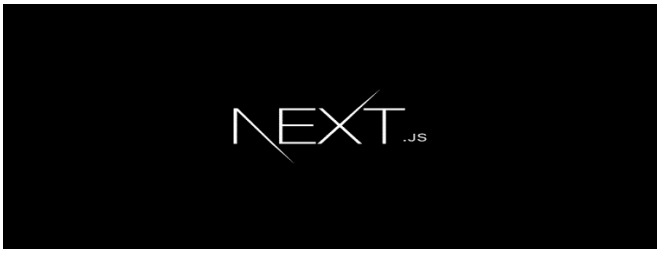
Eğitim kurumlarının karmaşık sınıf zaman çizelgelerini düzenlemesi ve optimize etmesi giderek daha zor bir hale gelmektedir. Bu ihtiyacı karşılamak amacıyla geliştirilen bu projede, Next.js kullanılarak bir web tabanlı uygulama oluşturulmuştur. Bu uygulama, çeşitli kısıtlamalara dayalı olarak sınıf zaman çizelgelerini düzenlemeyi ve optimize etmeyi amaçlamaktadır. Bu makalede, projenin geliştirilme aşamaları, kullanılan teknolojiler ve benimsenen metodoloji detaylı bir şekilde açıklanacaktır.

## II. KULLANILAN TEKNOLOJILER

### *A. Next.js*

Proje, React tabanlı Next.js çerçevesi üzerine inşa edilmiştir. Next.js'in öne çıkan özellikleri arasında sunucu tarafı renderleme, modüler yapı, ve gelişmiş yönlendirme sistemi bulunmaktadır. Bu çerçeve, uygulamanın hızlı ve etkili bir şekilde geliştirilmesine olanak tanımıştır.

Next.js'in sunucu tarafı renderleme yetenekleri, sayfaların hızlı bir şekilde yüklenmesine katkıda bulunurken, modüler yapısı ise kodun daha düzenli ve bakımı kolay olmasını sağlamaktadır. Ayrıca, gelişmiş yönlendirme sistemi sayesinde kullanıcılar arasında gezinme süreci daha kullanıcı dostu hale getirilmiştir.




### *B. MongoDB*

Veritabanı olarak MongoDB tercih edilmiştir. MongoDB, yapılandırılmamış veriyi yönetmek için esnek bir çözüm sunmaktadır. Projenin verileri, sınıf zaman çizelgeleri ve kısıtlamalar gibi, MongoDB üzerinde etkili bir şekilde depolanır ve alınır. Bu, uygulamanın ölçeklenebilirliğini artırır.


### *C. Javascript*

Proje, JavaScript dilinde yazılmıştır. JavaScript'in çok yönlü kullanımı, hem frontend hem de backend tarafında sorunsuz bir entegrasyon sağlar. Asenkron yapısı, uygulamanın birçok isteği aynı anda işleyebilmesine ve daha hızlı tepki vermesine olanak tanır.


## III. METODOLOJI

Projemizin geliştirilme süreci, Next.js, MongoDB ve graf renklendirme algoritması kullanılarak oluşturulan zaman çizelgesi oluşturma uygulamasının genel tasarımını içermektedir. Bu tasarım, projenin başlangıcından itibaren dikkatlice planlanan ve adım adım ilerleyen bir süreci kapsamaktadır.

Projemizin ilk aşaması, genel tasarımın belirlenmesiyle başlamıştır. Kullanıcı ihtiyaçları, projenin temel hedefleri ve beklenen sonuçlar dikkate alınarak, zaman çizelgesi oluşturma uygulamasının ana özellikleri belirlenmiştir. Bu aşamada, Next.js'in modüler yapısı ve MongoDB'nin veritabanı entegrasyonu, uygulamanın temel mimarisini şekillendirmek için kullanılmıştır.

Uygulamanın geliştirilme sürecinde, frontend tarafında Next.js kullanılarak kullanıcı arayüzü tasarlanmış ve geliştirilmiştir. Kullanıcı dostu bir arayüz, zaman çizelgesi oluşturma sürecini kullanıcılara anlaşılır ve etkili bir şekilde iletmek amacıyla özenle planlanmıştır. Arayüzdeki kullanıcı etkileşimleri, projenin başarılı bir şekilde tamamlanabilmesi için önemli bir faktör olmuştur.

Backend geliştirme aşamasında ise MongoDB, kullanıcılara ait sınıf zaman çizelgelerinin güvenli ve etkili bir şekilde depolanmasını sağlamıştır. Veritabanı entegrasyonu, uygulamanın veri yönetimi ihtiyaçlarını karşılayarak sistemin sağlam bir temel üzerine inşa edilmesini sağlamıştır.

Graf renklendirme algoritması, projenin anahtar bileşenlerinden biridir. Bu algoritma, çakışan derslerin zaman çizelgesinde aynı hücrede olmamasını sağlamak amacıyla entegre edilmiştir. Kullanıcı tarafından belirlenen kısıtlamalara göre efektif bir çözüm sunarak, zaman çizelgesi oluşturma sürecinde optimizasyon sağlamıştır.

Genel olarak, projenin tasarımı, kullanıcıların ihtiyaçlarına odaklanan, etkileşimli ve kullanıcı dostu bir zaman çizelgesi oluşturma uygulamasını hedeflemiştir. Bu tasarım süreci, başlangıçtan bitişe kadar dikkatlice planlanmış ve her aşamada projenin ana hedeflerine uygun olarak ilerlemiştir.

## IV. PROJENİN YAPIM AŞAMALARI

Bu bölümde projenin yapım aşamaları alt başlıklar halinde detaylandırılmıştır.

### *A. Genel Tasarım*

Proje, genel tasarım aşamasında başlamıştır. Kullanıcı ihtiyaçları ve projenin temel hedefleri belirlenerek, zaman çizelgesi oluşturma uygulamasının genel tasarımı şekillendirilmiştir. Bu aşamada, projenin ana özellikleri, kullanılacak teknolojilerin gereksinimleri, graf renklendirme algoritmasının gereksinimleri araştırılmış ve renklendirme algoritmasına göre oluşturulacak modellerin genel şablonu planlanmıştır.

### *B. Next.js Proje Şablonu Oluşturulması*

Next.js proje şablonu, projenin frontend kısmının temel altyapısının hazırlanmasını içerir. Bu aşamada, Next.js'in modüler yapısından yararlanarak, uygulamanın genel yapısını şekillendiren bir proje şablonu oluşturulmuştur. Bu şablonda, sayfa yönlendirmeleri, veri getirme metotları ve bileşen mimarisi gibi temel özellikler yer almaktadır. Next.js ile proje şablonu oluşturmak için ise öncelikle ortamınızda npm(node package manager) ve node.js kurulu olduğundan emin olun. Sonrasında ise "*npx create-next-app proje-adi*" komutu ile next.js şablonu oluşturabilir ve ilgili proje dizininde "*npm run dev*" komutu ile projenizi başlatabilirsiniz.

### *C. Veri Tabanı Entegrasyonu*

Proje geliştirme sürecinde veri tabanı entegrasyonu, MongoDB kullanarak güvenilir ve ölçeklenebilir bir veritabanı çözümü sunmak üzere tasarlanmıştır. Bu entegrasyonun temelini oluşturan Mongoose, MongoDB ile etkileşim kurabilmemizi ve uygulama içinde veritabanı operasyonlarını yönetmemizi sağlar. Örnek olarak Mongo DB ve Mongoose entegrasyonunu "*Figure 1*" üzerinde görebilirsiniz. Burada mongoose modülü içeri aktarılmış ve .env.local içerisinde gizlenmiş halde bulunan MONGO_URI bağlantısı kurulmuştur. Asenkron olarak oluşturulan connectDB metodu ise açık bir bağlantıyı tekrar oluşturmamak için asenkron olarak tanımlanmıştır.

```
const mongoose = require('mongoose');
const MONGO_URI = process.env.MONGO_URI
const connectDB = async () => {

    if(mongoose.connections[0].readyState) return;

    const conn = await mongoose.connect(MONGO_URI, {
        useNewUrlParser: true
    });
    console.log(`MongoDB connected: ${conn.connection.host}`);
}
module.exports = connectDB;
```

*Figure 1*

Mongoose, MongoDB'ye kolay bir erişim sağlayarak, şemalar, modeller ve sorgular üzerinde bir katman sağlar. Bu, verilerin düzenli bir şekilde depolanmasını ve veri yapısının daha etkili bir şekilde yönetilmesini mümkün kılar. Şemalar, veritabanındaki belgelerin yapısını tanımlayarak veri bütünlüğünü sağlar. Örnek olarak Mongoose ile oluşturulmuş ve proje içerisinde Class modelini temsil eden bir şemayı *Figure 2*' de görebilirsiniz.

```
const mongoose = require('mongoose');
const ClassSchema = new mongoose.Schema({
    code:{
        type: "String",
        maxlength: [8, "Subject code cannot be more than 8 characters."]
    }
}, {
    toJSON: { virtuals: true },
    toObject: { virtuals: true }
});
ClassSchema.virtual('givenSubjects',{
    ref: "GivenSubject",
    localField: "_id",
    foreignField: "class",
    justOne: false
})
const Class = mongoose.models.Class || mongoose.model("Class", ClassSchema);
module.exports = Class;
```

*Figure 2*

Proje içinde Mongoose kullanımı, kullanıcıların sınıf zaman çizelgelerini ve ilgili verilerini MongoDB veritabanında güvenli bir şekilde depolamasına olanak tanır. Sorguların ve veri manipülasyon operasyonlarının yönetimi Mongoose aracılığıyla yapılır, bu da projenin sağlam bir veri yönetimi altyapısına sahip olmasını sağlar.

Veri tabanı entegrasyonu aşaması, projenin backend kısmının güçlü ve ölçeklenebilir bir şekilde tasarlanmasını destekler. Mongoose kullanımı, veritabanı operasyonlarının basitleştirilmesini sağlayarak geliştirme sürecini hızlandırır ve projenin performansını artırır.

### *D. Modellerin Oluşturulması*

Uygulamanın temel veri modelleri, derslerin, sınıfların ,dersi veren hocaların ve verilen derslerin oluşturulan senaryoya göre yapılandırılmasını içerir. Veri tabanına uygun modellerin tasarımı ve oluşturulması bu aşamada gerçekleştirilmiştir. Dosya hiyerarşisi "Figure 3" ve örnek model ise "Figure 2" üzerinde gösterilmiştir.

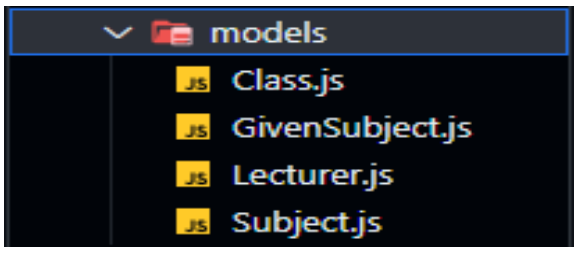



*Figure 3*

### E. API, Servis ve Controller Oluşturulması

Next.js projesi içinde, projenin ihtiyaçlarına uygun olarak RESTful API'lar oluşturulmuştur. Bu API'lar, kullanıcıların uygulama içindeki çeşitli işlemleri gerçekleştirmelerine olanak tanır. Örneğin, ders ekleme, sınıf zaman çizelgesi oluşturma gibi temel işlemleri gerçekleştirebilen API endpoint'leri tanımlanmış ve oluşturulmuştur. "Figure 4" API dosya içeriğini ve "Figure 5" ise projede bulunan "givenSubject" nesnesine ait bir route yapısını içermektedir.

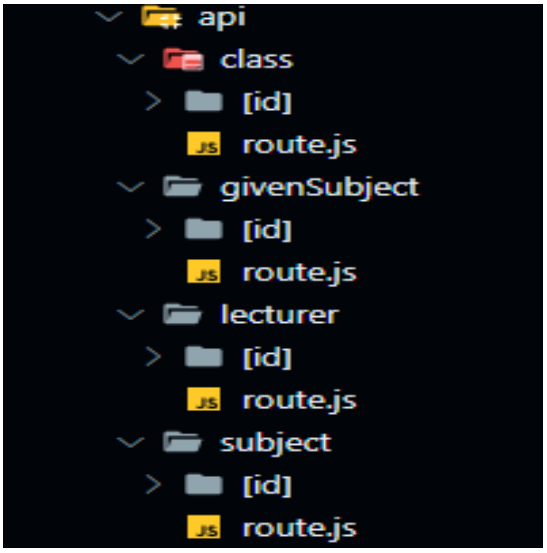



*Figure 4*

```
const givenSubjectController = require("@/app/controllers/GivenSubjectController");
export async function GET() {
  const res = await givenSubjectController.getAll({
    populate: ["subject", "lecturer"],
  });
  return res;
}
export async function POST(req) {
  const res = await givenSubjectController.insert(await req.json());
  return res;
}
export async function PUT(req) {
  const { id, ...data } = await req.json();
  const res = await givenSubjectController.update(id, data);
  return res;
}
export async function DELETE(req) {
  const { id } = await req.json();
  const res = await givenSubjectController.delete(id);
  return res;
}
```

*Figure 5*

Servisler, uygulamanın backend tarafında iş mantığı işlemlerini gerçekleştiren modüllerdir. Bu aşamada, belirlenen işlevselliğe uygun olarak gerekli servisler oluşturulmuş ve projeye entegre edilmiştir. Örneğin, zaman çizelgesi oluşturulması için kullanılan servis, derslerin çakışmamasını sağlamak üzere graf renklendirme algoritmasını içerebilir. Örnek olarak "givenSubjects" nesnesi için oluşturulmuş servis için tıklayınız.

Controller'lar, gelen istekleri yönlendiren ve ilgili servisleri çağıran modüllerdir. Her bir API endpoint'i için bir Controller oluşturularak, ilgili işlemlerin Servis tarafından gerçekleştirilmesi sağlanmıştır. Bu sayede, uygulamanın modüler bir yapıda olması ve işlevselliğin daha iyi yönetilmesi amaçlanmıştır Örnek olarak "givenSubject" nesnesi için oluşturulmuş olan controller için tıklayınız.

### F. Graf Algoritmasının Entegrasyonu

Projenin en kritik ve teknik detaylardan birini oluşturan graf renklendirme algoritması, çakışan derslerin zaman çizelgesinde aynı hücrede olmamasını temin etmek adına detaylı bir şekilde tasarlanmıştır. Bu aşama, zaman çizelgesi oluşturma sürecindeki karmaşıklığı ele almak ve çeşitli kısıtlamalara uygun renklendirme sağlamak amacıyla gerçekleştirilmiştir.

Algoritmanın başlangıcı, bir grafın oluşturulması ile başlar. Bu graf, derslerin ve çakışmalarının temsil edildiği bir veri yapısıdır. Graf oluşturulurken, dersler arasındaki zaman çakışmaları tespit edilir ve bu çakışan dersler arasında bir kenar oluşturulur. Grafın her düğümü, bir dersi temsil eder.

```
constructor(res, classes) {
    this.res = res;
    this.classes = classes;
    this.dugumler = new Map();

    this.res.forEach(givenSubject => {
      this.dugumEkle(givenSubject.id);
    });

    this.res.forEach(givenSubject => {
      this.res.forEach(otherGivenSubject =>
{
        if (
          givenSubject !== otherGivenSubject
&&
          this.zamanCakisiyor(givenSubject,
otherGivenSubject)
        ) {
          this.kenarEkle(givenSubject.id,
otherGivenSubject.id);
        }
      });
    });
    this.grafRenklendir();
  }
```

Algoritmanın ana adımı, zaman çakışmalarını tespit etmek ve grafı oluşturmaktır. Bu aşamada, her bir dersin diğer derslerle olan zaman çakışmaları kontrol edilir ve çakışma varsa graf üzerinde kenar eklenir.

```
zamanCakisiyor(ders1, ders2) {
    let smaller;
    let bigger;
    if (ders1.day === ders2.day) {
      if (ders1.startTime <=
ders2.startTime) {
        smaller = ders1;
        bigger = ders2;
      } else {
        smaller = ders2;
        bigger = ders1;
      }
```

```
      if( smaller.startTime <=
bigger.startTime &&
        smaller.endTime > bigger.startTime)
        {
          return true;
        }
      else{
        return false;
      }
    }
  }
```

Graf oluşturulduktan sonra, her düğümün komsularıyla olan ilişkileri belirlenir. Ayrıca, her düğümün derecesi hesaplanır. Bu derece, düğümün kaç tane komşusu olduğunu gösterir ve renklendirme yapılırken ilk olarak bu düğüme renk ataması yapılır.

```
dugumEkle(etiket) {
    this.dugumler.set(etiket, { komsular:
[], renk: null, derece: 0 });
  }
  kenarEkle(dugum1, dugum2) {
    this.dugumler.get(dugum1).komsular.push(
dugum2);
    this.dugumler.get(dugum1).derece++;
  }
```

Renklendirme aşamasında, her bir ders düğümü, BFS (Breadth-First Search) algoritması ile renklendirilir. Derecesi yüksek olan düğümler öncelikli olarak renklendirilir ve komşu düğümlerle çakışmamaya özen gösterilir.

```
grafRenklendir() {
    const renkler = [...new
Set(this.classes.map(x => x.id))];
    const ziyaretEdilenDugumler = new Set();

    const siraliDugumler =
[...this.dugumler.keys()].sort(
      (a, b) => this.dugumler.get(b).derece
- this.dugumler.get(a).derece
    );

    for (const dugum of siraliDugumler) {
      if (!ziyaretEdilenDugumler.has(dugum))
{
        this.bfsAlgoritmasi(dugum, renkler,
ziyaretEdilenDugumler);
      }
    }
  }
```

```
  bfsAlgoritmasi(baslangicDugumu, renkler,
ziyaretEdilenDugumler) {
    const kuyruk = [baslangicDugumu];
    const ziyaretEdilen = new Set();

    while (kuyruk.length > 0) {
      const mevcutDugum = kuyruk.shift();

      if (!ziyaretEdilen.has(mevcutDugum)) {
        const komsular =
this.dugumler.get(mevcutDugum).komsular;
        const kullanilanRenkler = new Set(
          komsular.map((komsu) =>
this.dugumler.get(komsu).renk)
        );

        for (const renk of renkler) {
          if (!kullanilanRenkler.has(renk))
{
            this.dugumler.get(mevcutDugum).r
enk = renk;
            break;
          }
        }
        ziyaretEdilen.add(mevcutDugum);
        ziyaretEdilenDugumler.add(mevcutDugu
m);
        kuyruk.push(...komsular.filter((koms
u) => !ziyaretEdilen.has(komsu)));
      }
    }
  }
```

Son olarak, her düğümün renk bilgisi alınarak renklendirilmiş grafi elde etmek mümkündür.

```
renklendirilmisGrafiGetir() {
    const dugumler =
[...this.dugumler.entries()];
    const renklendirilmisGraf = [];

    for (const [dugum, veri] of dugumler) {
      renklendirilmisGraf.push({
        dugum,
        renk: veri.renk,
        data: veri.data,
      });
    }
    return renklendirilmisGraf;
  }
```

### G. *Frontend Oluşturulması*

Uygulamanın kullanıcı arayüzü, Next.js'in modüler yapısından faydalanılarak oluşturulmuştur. React bileşenleri kullanılarak(Figure 6), modellerin ve servislerin entegrasyonu sağlanarak, kullanıcı dostu bir arayüz tasarlanmıştır. Frontend, kullanıcıların zaman çizelgesi oluşturma sürecini etkileşimli bir şekilde yönetmelerine olanak sağlayacak şekilde geliştirilmiştir(Figure 7).

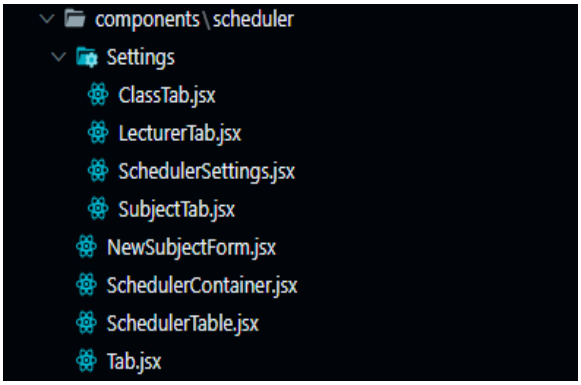



*Figure 6*

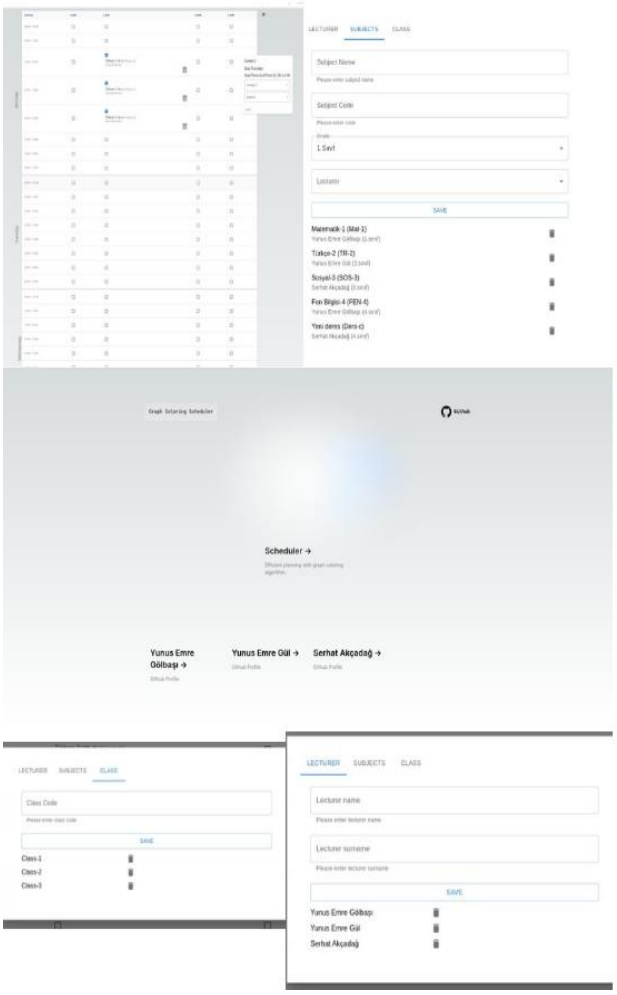



*Figure 7*

## V. KAYNAKÇA

Graf renklendirme algoritması ile ders program oluşturması amaçlanan web uygulamasında aşağıda bulunan kaynaklardan araştırmalar yapılmış ve sonuç olarak özgün bir proje oluşturulmuştur.

[1] Welsh-Powell Algoritması

[2] Breadth First Search Algoritması

[3] Next.js Dokümantasyonu

Kaynak kodlarını içeren Github linki için tıklayınız.