CENG 307

Blog Yönetim Sistemi

2025-2026 Dönemi Dönem Sonu Projesi

Proje Raporu

Tarih: Aral■k 2024

■Ç■NDEK■LER

1. PROJE ÖZET■

Proje Ad■: Blog Yönetim Sistemi

Bu proje, ö■retmenlerin (yazarlar) blog yaz■lar■ olu■turabildi■i ve ö■rencilerin (okuyucular) bu yaz■lar■ okuyup yorum yapabildi■i basit bir blog platformudur. Proje iki ana bile■enden olu■maktad■r: NestJS ile yaz■lm■■ backend API ve React ile yaz■lm■■ frontend kullan■c■ arayüzü.

Temel Özellikler:

• Kullan■c■ kay■t ve giri■ sistemi (JWT ile güvenli)

• ■ki farkl■ kullan■c■ rolü: Ö■retmen (Yazar) ve Ö■renci (Okuyucu)

• Blog yaz■lar■ için CRUD (Create, Read, Update, Delete) i■lemleri

• Kategori bazl■ yaz■ organizasyonu

• Yorum sistemi

• Yaz■ be■enme özelli■i

• Responsive ve kullan■c■ dostu arayüz

2. S■STEM M■MAR■S■

Proje Yap■s■:

Proje klasik client-server mimarisi kullanmaktad■r. Frontend ve backend tamamen ayr■ uygulamalard■r ve REST API üzerinden haberle■irler.

Katman	Teknoloji	Port
Frontend	React 18	3000
Backend	NestJS	5000
Veritaban■	SQLite	Dosya tabanl■
Kimlik Do■rulama	JWT	-

■leti■im Ak■■■:

1. Kullan■c■ frontend üzerinden i■lem yapar
2. Frontend, axios kullanarak backend'e HTTP iste■i gönderir
3. Backend, JWT token'■ do■rular ve yetki kontrolü yapar
4. Backend, veritaban■ i■lemini gerçekle■tirir (TypeORM ile)
5. Backend, sonucu JSON format■nda frontend'e döner
6. Frontend, sonucu kullan■c■ya gösterir

3. VER■TABANI TASARIMI

Entity ■li■ki Diyagram■ (ERD):

Projede 4 ana tablo (entity) bulunmaktad■r. Bu tablolar aras■ndaki ili■kiler a■a■■daki gibidir:

3.1. Users (Kullan■c■lar) Tablosu

Sütun	Tip	Aç■klama
id	INTEGER	Primary Key, otomatik artan
email	VARCHAR	Unique, kullan■c■ emaili
password	VARCHAR	Hash'lenmi■ ■ifre
name	VARCHAR	Kullan■c■ ad■ soyad■
role	VARCHAR	student veya teacher
createdAt	DATETIME	Kay■t tarihi

3.2. Categories (Kategoriler) Tablosu

Sütun	Tip	Aç■klama
id	INTEGER	Primary Key, otomatik artan
name	VARCHAR	Unique, kategori ad■
description	VARCHAR	Kategori aç■klamas■ (opsiyonel)
createdAt	DATETIME	Olu■turma tarihi

3.3. Posts (Blog Yaz■lar■) Tablosu

Sütun	Tip	Aç■klama
id	INTEGER	Primary Key, otomatik artan
title	VARCHAR	Yaz■ ba■l■■■
content	TEXT	Yaz■ içeri■i
imageUrl	VARCHAR	Resim URL (opsiyonel)
likes	INTEGER	Be■eni say■s■ (varsay■lan 0)
authorId	INTEGER	Foreign Key -> Users.id
categoryId	INTEGER	Foreign Key -> Categories.id
createdAt	DATETIME	Olu■turma tarihi
updatedAt	DATETIME	Güncellenme tarihi

3.4. Comments (Yorumlar) Tablosu

Sütun	Tip	Aç■klama
id	INTEGER	Primary Key, otomatik artan
content	TEXT	Yorum içeri■i
userId	INTEGER	Foreign Key -> Users.id
postId	INTEGER	Foreign Key -> Posts.id
createdAt	DATETIME	Olu■turma tarihi

■li■kiler:

• User - Posts: One-to-Many (Bir kullan■c■ birçok yaz■ yazabilir)

• User - Comments: One-to-Many (Bir kullan■c■ birçok yorum yapabilir)

• Category - Posts: One-to-Many (Bir kategoride birçok yaz■ olabilir)

• Post - Comments: One-to-Many (Bir yaz■da birçok yorum olabilir)

4. BACKEND (NestJS) DETAYLARI

4.1. Teknolojiler

• NestJS: Modern, ölçeklenebilir Node.js framework'ü

• TypeORM: Object-Relational Mapping kütüphanesi

• SQLite: Hafif, dosya tabanl■ veritaban■

• JWT (JSON Web Tokens): Güvenli kimlik do■rulama

• Bcrypt: ■ifre hashleme

• Passport: Kimlik do■rulama middleware'i

4.2. Proje Yap■s■

```
backend/
■■■ src/
■ ■■■ entities/ # Veritaban■ modelleri
■ ■ ■■■ user.entity.ts
■ ■ ■■■ post.entity.ts
■ ■ ■■■ comment.entity.ts
■ ■ ■■■ category.entity.ts
■ ■■■ controllers/ # API endpoint'leri
■ ■ ■■■ auth.controller.ts
■ ■ ■■■ post.controller.ts
■ ■ ■■■ comment.controller.ts
■ ■ ■■■ category.controller.ts
■ ■■■ guards/ # Yetkilendirme
■ ■ ■■■ jwt-auth.guard.ts
■ ■■■ strategies/ # Kimlik do■rulama
■ ■ ■■■ jwt.strategy.ts
■ ■■■ app.module.ts # Ana modül
■ ■■■ main.ts # Giri■ noktas■
■■■ package.json
```

4.3. Entity (Model) Aç■klamalar■

User Entity:

User entity, kullan■c■ bilgilerini tutar. TypeORM decoratorleri (@Entity, @Column vb.) kullan■larak veritaban■ ■emas■ tan■mlan■r. @OneToMany decorator'ü ile User'■n Posts ve Comments ile ili■kisi kurulur.

```
@Entity('users') export class User { @PrimaryGeneratedColumn() id: number; @Column({ unique: true }) email: string; @Column() password: string; @Column() name: string; @Column({ default: 'student' }) role: string; @OneToMany(() => Post, post => post.author) posts: Post[]; @OneToMany(() => Comment, comment => comment.user) comments: Comment[]; }
```

Post Entity:

Post entity, blog yaz■lar■n■ temsil eder. @ManyToOne decorator'ü ile User ve Category'ye ba■lan■r. @JoinColumn ile foreign key sütunu belirtilir. @OneToMany ile Comments ili■kisi kurulur.

```
@Entity('posts') export class Post { @PrimaryGeneratedColumn() id: number; @Column() title: string; @Column('text') content: string; @Column({ nullable: true }) imageUrl: string; @ManyToOne(() => User, user => user.posts) @JoinColumn({ name: 'authorId' }) author: User; @ManyToOne(() => Category, category => category.posts) @JoinColumn({ name: 'categoryId' }) category: Category; @OneToMany(() => Comment, comment => comment.post) comments: Comment[]; }
```

4.4. Controller Aç■klamalar■

Auth Controller:

Kullan■c■ kay■t ve giri■ i■lemlerini yönetir. @Post decorator'ü ile POST endpoint'leri tan■mlan■r. Bcrypt ile ■ifre hashlenir, JWT ile token olu■turulur.

```
@Controller('auth') export class AuthController { @Post('register') async register(@Body() body) { // ■ifreyi hashle const hashedPassword = await bcrypt.hash(body.password, 10); // Kullan■c■ olu■tur const user = this.userRepository.create({ email: body.email, password: hashedPassword, name: body.name, role: body.role }); await this.userRepository.save(user); // JWT token olu■tur const token = this.jwtService.sign({ id: user.id, email: user.email, role: user.role }); return { user, token }; } }
```

Post Controller:

Blog yaz■lar■ için CRUD i■lemlerini yönetir. @UseGuards(JwtAuthGuard) ile korumal■ endpoint'ler olu■turulur. Sadece yetkili kullan■c■lar i■lem yapabilir.

```
@Controller('posts') export class PostController { // Herkese aç■k @Get() async getAllPosts() { return await this.postRepository.find({ relations: ['author', 'category', 'comments'] }); } // Sadece teacher yaz■ yazabilir @Post() @UseGuards(JwtAuthGuard) async createPost(@Request() req, @Body() body) { if (req.user.role !== 'teacher') { throw new HttpException('Only teachers can create posts', 403); } const post = this.postRepository.create({ ...body, authorId: req.user.id }); return await this.postRepository.save(post); } }
```

5. FRONTEND (React) DETAYLARI

5.1. Teknolojiler

• React 18: Modern UI kütüphanesi

• React Router: Sayfa yönlendirme

• Axios: HTTP istekleri için

• CSS3: Stil ve düzen

• LocalStorage: Token ve kullan■c■ bilgilerini saklama

5.2. Proje Yap■s■

```
frontend/
■■■ src/
■ ■■■ components/ # React component'leri
■ ■ ■■■ Navbar.js # Üst menü
■ ■ ■■■ Login.js # Giri■ sayfas■
■ ■ ■■■ Register.js # Kay■t sayfas■
■ ■ ■■■ PostList.js # Yaz■ listesi
■ ■ ■■■ PostDetail.js # Yaz■ detay■
■ ■ ■■■ CreatePost.js # Yaz■ olu■turma
■ ■ ■■■ EditPost.js # Yaz■ düzenleme
■ ■ ■■■ *.css # Component stilleri
■ ■■■ services/
■ ■ ■■■ api.js # API ileti■imi
■ ■■■ App.js # Ana uygulama
■ ■■■ App.css # Global stiller
■ ■■■ index.js # Giri■ noktas■
■■■ package.json
```

5.3. API Service

api.js dosyas■, backend ile ileti■imi merkezi bir yerden yönetir. Axios interceptor kullanarak her istekte otomatik olarak JWT token eklenir.

```
const api = axios.create({ baseURL: 'http://localhost:5000/api' }); // Her istekte token ekle
api.interceptors.request.use((config) => { const token = localStorage.getItem('token'); if (token) {
config.headers.Authorization = `Bearer ${token}`; } return config; }); // API fonksiyonlar■ export const
postAPI = { getAll: () => api.get('/posts'), getById: (id) => api.get(`/posts/${id}`), create: (data) =>
api.post('/posts', data), update: (id, data) => api.put(`/posts/${id}`, data), delete: (id) =>
api.delete(`/posts/${id}`), like: (id) => api.post(`/posts/${id}/like`) };
```

6. API ENDPOINT'LER■ VE KULLANIMI

6.1. Auth Endpoints

Method	Endpoint	Aç■klama	Auth
POST	/api/auth/register	Yeni kullan■c■ kayd■	Hay■r
POST	/api/auth/login	Kullan■c■ giri■i	Hay■r

Register ■ste■i Örne■i:

```
POST /api/auth/register Content-Type: application/json { "name": "Ahmet Y■lmaz", "email": "ahmet@example.com", "password": "123456", "role": "student" } Yan■t: { "message": "User registered successfully", "user": { "id": 1, "name": "Ahmet Y■lmaz", "email": "ahmet@example.com", "role": "student" }, "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." }
```

6.2. Post Endpoints

Method	Endpoint	Aç■klama	Auth
GET	/api/posts	Tüm yaz■lar■ getir	Hay■r
GET	/api/posts/:id	Tek yaz■ detay■	Hay■r
POST	/api/posts	Yeni yaz■ olu■tur	Teacher
PUT	/api/posts/:id	Yaz■y■ güncelle	Yazar
DELETE	/api/posts/:id	Yaz■y■ sil	Yazar
POST	/api/posts/:id/like	Yaz■y■ be■en	Evet

Yaz■ Olu■turma ■ste■i Örne■i:

```
POST /api/posts Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9... Content-Type: application/json { "title": "React Hooks Nedir?", "content": "React Hooks, React 16.8 ile gelen...", "categoryId": 1, "imageUrl": "https://example.com/react.jpg" } Yan■t: { "message": "Post created successfully", "post": { "id": 1, "title": "React Hooks Nedir?", "content": "React Hooks, React 16.8 ile gelen...", "authorId": 2, "categoryId": 1, "likes": 0, "createdAt": "2024-12-18T10:30:00.000Z" } }
```

6.3. Comment Endpoints

Method	Endpoint	Aç■klama	Auth
GET	/api/comments/post/:id	Yaz■n■n yorumlar■	Hay■r
POST	/api/comments	Yeni yorum ekle	Evet
DELETE	/api/comments/:id	Yorumu sil	Evet

6.4. Category Endpoints

Method	Endpoint	Aç■klama	Auth
GET	/api/categories	Tüm kategoriler	Hay■r
GET	/api/categories/:id	Kategori detay■	Hay■r
POST	/api/categories	Yeni kategori	Teacher
PUT	/api/categories/:id	Kategori güncelle	Teacher
DELETE	/api/categories/:id	Kategori sil	Teacher

7. KULLANICI ROLLER■ VE YETK■LER

7.1. Student (Ö■renci/Okuyucu) Rolü

• Tüm blog yaz■lar■n■ görüntüleyebilir

• Yaz■ detaylar■n■ okuyabilir

• Yaz■lara yorum yapabilir

• Kendi yorumlar■n■ silebilir

• Yaz■lar■ be■enebilir

• Kategoriye göre filtreleme yapabilir

• Yaz■ olu■turamaz, düzenleyemez veya silemez

7.2. Teacher (Ö■retmen/Yazar) Rolü

• Student rolünün tüm yetkilerine sahiptir

• Yeni blog yaz■s■ olu■turabilir

• Kendi yaz■lar■n■ düzenleyebilir

• Kendi yaz■lar■n■ silebilir

• Yeni kategori olu■turabilir

• Kategorileri düzenleyebilir ve silebilir

• Tüm yorumlar■ yönetebilir (kendi yaz■lar■ndaki yorumlar■ silebilir)

7.3. Yetkilendirme Mekanizmas■

Yetkilendirme JWT (JSON Web Token) ile sa■lan■r. Kullan■c■ giri■ yapt■■■nda, backend bir token olu■turur ve bu token kullan■c■n■n id, email ve role bilgilerini içerir. Frontend her istekte bu token'■ Authorization header'■nda gönderir. Backend, JwtAuthGuard kullanarak token'■ do■rular ve kullan■c■n■n yetkisini kontrol eder.

8. COMPONENT AÇIKLAMALARI

Component	Aç■klama	Özellikler
App.js	Ana uygulama component'i	Routing, korumal■ route'lar
Navbar.js	Üst navigasyon çubu■u	Kullan■c■ bilgisi, giri■/ç■k■■
Login.js	Kullan■c■ giri■ sayfas■	Form validasyonu, token saklama
Register.js	Kullan■c■ kay■t sayfas■	Rol seçimi, form validasyonu
PostList.js	Blog yaz■lar■ listesi	Filtreleme, grid görünüm
PostDetail.js	Yaz■ detay sayfas■	Yorum sistemi, be■eni
CreatePost.js	Yeni yaz■ olu■turma	Sadece teacher, form
EditPost.js	Yaz■ düzenleme	Sadece yazar, form

8.1. PostList Component Detay■

PostList component'i, tüm blog yaz■lar■n■ grid format■nda gösterir. useEffect hook'u ile sayfa yüklendi■inde veriler çekilir. useState hook'u ile state yönetimi yap■l■r. Kategori filtresi ile yaz■lar filtrelenebilir.

```
function PostList() { const [posts, setPosts] = useState([]); const [categories, setCategories] =
useState([]); const [selectedCategory, setSelectedCategory] = useState('all'); useEffect(() => { //
Verileri yükle const loadData = async () => { const [postsRes, categoriesRes] = await Promise.all([
postAPI.getAll(), categoryAPI.getAll() ]); setPosts(postsRes.data); setCategories(categoriesRes.data); };
loadData(); }, []); // Filtreleme const filteredPosts = selectedCategory === 'all' ? posts :
posts.filter(post => post.categoryId === parseInt(selectedCategory)); return ( <div
className="post-list-container"> {/* Kategori filtresi */} <select value={selectedCategory} onChange={(e)
=> setSelectedCategory(e.target.value)}> <option value="all">Tümü</option> {categories.map(cat => ( <option
key={cat.id} value={cat.id}>{cat.name}</option> ))} </select> {/* Yaz■ kartlar■ */} <div
className="posts-grid"> {filteredPosts.map(post => ( <div key={post.id} className="post-card">
<h3>{post.title}</h3> <p>{post.content.substring(0, 150)}...</p> <span>❤■ {post.likes}</span> </div> ))}
</div> </div> ); }
```

8.2. PostDetail Component Detay■

PostDetail component'i, tek bir yaz■n■n tüm detaylar■n■ gösterir. useParams hook'u ile URL'den yaz■ ID'si al■n■r. Yorum ekleme, silme ve yaz■y■ be■enme fonksiyonlar■ bulunur.

```
function PostDetail() { const { id } = useParams(); // URL'den ID al const [post, setPost] =
useState(null); const [comment, setComment] = useState(''); const user =
JSON.parse(localStorage.getItem('user') || '{}'); const isAuthor = user.id === post?.authorId; useEffect(()
=> { loadPost(); }, [id]); const loadPost = async () => { const response = await postAPI.getById(id);
setPost(response.data); }; const handleLike = async () => { await postAPI.like(id); loadPost(); // Yenile
}; const handleAddComment = async (e) => { e.preventDefault(); await commentAPI.create({ content: comment,
postId: parseInt(id) }); setComment(''); loadPost(); }; return ( <div> <h1>{post?.title}</h1>
<p>{post?.content}</p> <button onClick={handleLike}>❤■ Be■en ({post?.likes})</button> {/* Yorum formu */}
<form onSubmit={handleAddComment}> <textarea value={comment} onChange={(e) => setComment(e.target.value)}
/> <button type="submit">Yorum Yap</button> </form> {/* Yorumlar */} {post?.comments?.map(c => ( <div
key={c.id}> <strong>{c.user?.name}</strong> <p>{c.content}</p> </div> ))} </div> ); }
```

9. KURULUM VE ÇALI■TIRMA

9.1. Gereksinimler

• Node.js (v16 veya üzeri)

• npm veya yarn

• Modern bir web taray■c■ (Chrome, Firefox, Safari)

9.2. Backend Kurulumu

```
# Proje dizinine git cd backend # Ba■■ml■l■klar■ yükle npm install # Uygulamay■ çal■■t■r npm run start:dev # Backend http://localhost:5000 adresinde çal■■acak
```

9.3. Frontend Kurulumu

```
# Proje dizinine git cd frontend # Ba■■ml■l■klar■ yükle npm install # Uygulamay■ çal■■t■r npm start # Frontend http://localhost:3000 adresinde aç■lacak
```

9.4. ■lk Kullan■m

1. Backend'i ba■lat■n (localhost:5000)

2. Frontend'i ba■lat■n (localhost:3000)

3. Taray■c■da localhost:3000 adresini aç■n

4. 'Kay■t Ol' butonuna t■klay■n

5. Teacher rolü ile bir hesap olu■turun

6. Giri■ yap■n

7. ■lk blog yaz■n■z■ olu■turun

9.5. Test Verileri Olu■turma

Sistemi test etmek için önce kategoriler olu■turman■z gerekir. Teacher rolü ile giri■ yap■n ve kategoriler olu■turun (örn: Teknoloji, E■itim, Sa■l■k). Ard■ndan bu kategorilerde blog yaz■lar■ olu■turun. Student rolü ile farkl■ bir hesap aç■p yorum yapabilirsiniz.

10. KODLAR VE EKLER

10.1. Veritaban■ Diyagram■ Aç■klamas■

Veritaban■ diyagram■ 4 ana tabloyu ve aralar■ndaki ili■kileri gösterir:

```
USERS CATEGORIES ■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■ ■ id (PK) ■ ■ id (PK) ■ ■ email ■
■ name ■ ■ password ■ ■ description ■ ■ name ■ ■ createdAt ■ ■ role ■ ■■■■■■■■■■■■■■■■
■ createdAt ■ ■ ■■■■■■■■■■■■■■■■ ■ ■ ■ ■ 1:N ■ 1:N ■ ■ ▼ ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ POSTS ■ ■ ■■■■■■■■■■■■■■■■ ■ ■ ■ id
(PK) ■ ■ ■ ■ title ■ ■ ■ ■ content ■ ■ ■ ■ imageUrl ■ ■ ■ ■ likes ■ ■ ■ ■ authorId(FK)
■■■■ USERS ■ ■ ■ categoryId(FK)■■■■ CATEGORIES ■ ■ ■ createdAt ■ ■ ■ ■ updatedAt ■ ■ ■
■■■■■■■■■■■■■■■■ ■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ ■ 1:N ▼
■■■■■■■■■■■■■■■■ ■ COMMENTS ■ ■ ■■■■■■■■■■■■ ■ ■ ■ id (PK) ■ ■ ■ ■ content ■ ■ ■ ■
userId(FK)■■■■ USERS ■ ■ postId(FK)■■■■ POSTS ■ ■ createdAt■ ■ ■ ■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■
```

10.2. JWT Token Yap■s■

JWT token üç bölümden olu■ur: Header, Payload ve Signature. Payload bölümünde kullan■c■ bilgileri bulunur. Token, secret key ile imzalan■r ve her istekte do■rulan■r.

```
Örnek Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwi
ZW1haWwiOiJhaG1ldEBleGFtcGxlLmNvbSIsInJvbGUiOiJ0 ZWFjaGVyIiwiaWF0IjoxNjM5MTI4MDAwfQ.abc123xyz
Token Payload (decoded): { "id": 1, "email": "ahmet@example.com", "role": "teacher", "iat":
1639128000, "exp": 1639732800 }
```

10.3. Güvenlik Önlemleri

• ■ifreler bcrypt ile hashlenip saklan■r (düz metin olarak saklanmaz)

• JWT secret key production ortam■nda de■i■tirilmelidir

• SQL Injection'a kar■■ TypeORM parameterized queries kullan■r

• CORS ayarlar■ sadece frontend domain'ine izin verir

• XSS sald■r■lar■na kar■■ input validasyonu yap■l■r

• Kullan■c■ rolleri backend'de kontrol edilir (frontend kontrolü yeterli de■ildir)

10.4. Geli■tirme Önerileri

• ■ifre de■i■tirme özelli■i eklenebilir

• Profil foto■raf■ yükleme eklenebilir

• Yaz■ taslak kaydetme özelli■i

• Yaz■ arama fonksiyonu

• Sayfalama (pagination) eklenerek performans art■r■labilir

• Email do■rulama sistemi

• ■ifremi unuttum özelli■i

• Admin paneli için ayr■ bir rol

• Yaz■ payla■ma (sosyal medya entegrasyonu)

• Markdown deste■i yaz■ içeri■i için

10.5. Sonuç

Bu proje, modern web teknolojileri kullanarak basit ama i■levsel bir blog platformu olu■turmay■ amaçlamaktad■r. NestJS ve React kombinasyonu, ölçeklenebilir ve bak■m■ kolay bir mimari sa■lar. Proje, CRUD i■lemleri, kimlik do■rulama, yetkilendirme ve kullan■c■ rolleri gibi temel web geli■tirme konseptlerini içermektedir.

Kodlar basit ve anla■■l■r tutulmu■tur, böylece sunumda sorulara rahatça cevap verilebilir. Her component ve her endpoint'in amac■ ve nas■l çal■■t■■■ aç■kça bellidir.