Projeden ne anladım nasıl planladım

Sosyal Medya ve Link Yönetim Uygulaması

1. Genel Bakış

Bu uygulama, kullanıcıların sosyal medya linklerini ve bu linklere ait bilgileri (sosyal medya adı, açıklama) yönetebileceği dinamik ve interaktif bir platform sağlar. Uygulama, Angular framework kullanılarak geliştirilmiş olup, modern web teknolojileri ile kullanıcı deneyimini en üst düzeye çıkarmayı amaçlamaktadır.

Projeye önce tablolara ve nasıl yönetim yapılacağına karar vererek başladım. Sonra bu proje için nodejs express ile endpointler oluşturdum bunları postman'de test ederek ilerledim. Son aşama olarak Frontend projesini önce şablonlar halinde sonrasında ise daha dinamik servisler haline kurmaya geçtim. Atomic design kullanıldığı için componentlerimi atomlara molecullere ve organismlara böldüm. ve bunları bir sayfada topladım. bu esnada farklı paketler farklı teknolojileri bir arada kullandım. en son aşamada ise manuel bir test yaparak projemi ilettim.

2. Teknolojiler

- **Frontend**: Angular
- **Backend**: Node.js ve Express.js
- **Veritabanı**: MySQL
- **Kimlik Doğrulama**: JWT (JSON Web Token), Bycrypt
- **Stil ve Tasarım**: CSS ve Paketler
- **API Entegrasyonu**:HttpClient

3. Uygulama Yapısı

3.1. Frontend (Angular)

- **Modüler Yapı**: Uygulama, Angular'ın modüler yapısını kullanarak bölümlere ayrılmıştır. Her bir modül, belirli bir işlevselliği kapsar (örneğin, kullanıcı yönetimi, sosyal medya link yönetimi).
- **Bileşenler (Components)**:
    - **Ana Sayfa**: Kullanıcıya sosyal medya linklerinin genel bir görünümünü sağlar.
    - **Link Yönetimi**: Sosyal medya linklerinin listelendiği, düzenlendiği ve silindiği component.
    - **Link Formu**: Yeni sosyal medya linklerinin eklendiği veya mevcut linklerin düzenlendiği form.
    - **Profil Yönetimi**: Kullanıcının profil bilgilerini güncellemesine olanak sağlar.
    - **Giriş / Kayıt**: Kullanıcıların giriş yapabileceği veya yeni bir hesap oluşturabileceği sayfalar.
- **Servisler (Services)**:
    - **Sosyal Medya Link Servisi**: Sosyal medya linkleri ile ilgili CRUD işlemlerini yönetir.
    - **Kullanıcı Servisi**: Kullanıcı yönetimi ve kimlik doğrulama işlemlerini yönetir.

- **Guards**:
    - **AuthGuard**: Kullanıcıların yetkilendirilmiş sayfalara erişimini sağlar.
    - **AuthService**:Token ve kimlik doğrulama işlemlerini yönetir

3.2. Backend (Node.js ve Express)

- **CRUD API'leri**:
    - **Link API**: Sosyal medya linkleri ile ilgili CRUD işlemlerini yönetir.
    - **Kullanıcı API**: Kullanıcı kayıt, giriş ve profil yönetimi işlemlerini yönetir.
- **Kimlik Doğrulama ve Yetkilendirme**:
    - **JWT**: Kullanıcı kimlik doğrulamasını ve yetkilendirmeyi yönetir.
    - **Middleware**: Yetkilendirme kontrolleri ve güvenlik önlemleri sağlar.
- **Veritabanı (MySQL)**:
    - **Kullanıcılar Tablosu**: Kullanıcı bilgilerini saklar.
    - **Sosyal Medya Linkleri Tablosu**: Sosyal medya linkleri ve ilgili bilgileri saklar.

3.3. Veritabanı Tasarımı

- **Kullanıcılar Tablosu**:
    - id: INT, PRIMARY KEY, AUTO_INCREMENT
    - username: VARCHAR
    - password: VARCHAR
    - email: VARCHAR
    - isActive: TINYINT (1: aktif, 0: pasif)
- **Sosyal Medya Linkleri Tablosu**:
    - id: INT, PRIMARY KEY, AUTO_INCREMENT
    - userId: INT, FOREIGN KEY (Kullanıcı ID'si)
    - name: VARCHAR
    - url: VARCHAR
    - description: TEXT
    - created_at: TIMESTAMP
    - updated_at: TIMESTAMP
- **Role Tablosu:** admin = 1 ,user =2

3.4. Kullanıcı Yetkilendirme ve Roller

- **Roller**:
    - **Admin**: Tüm sosyal medya linklerini görebilir, düzenleyebilir ve yönetebilir. sosyal medya linklerinin sahibini görebilir ve bilgilerinde düzenleme yapabilir.
    - **Kullanıcı**: Sadece kendi sosyal medya linklerini görebilir ve düzenleyebilir.

4. Kullanıcı Arayüzü (UI/UX)

- **Responsive Tasarım**: Uygulama, mobil ve masaüstü cihazlar için uyumlu ve duyarlı bir tasarıma sahip olacaktır.
- **Kullanıcı Dostu Arayüz**: Kullanıcıların işlemlerini kolayca yapabilmesi için sade ve anlaşılır bir arayüz tasarlanacaktır.
- **Form Doğrulama**: Kullanıcı verilerinin doğruluğunu kontrol etmek için form doğrulama özellikleri uygulanacaktır.

5. Güvenlik

- **Veri Şifreleme**: Kullanıcı parolaları bcrypt ile şifrelenecektir.
- **Yetkilendirme**: JWT tabanlı kimlik doğrulama ile kullanıcı yetkilendirmesi sağlanacaktır.

API Dökümantasyonu

Genel Bakış

Bu doküman, sosyal medya yönetim uygulaması için oluşturmuş olduğum API'leri ve kullanılan paketleri açıklar. API'ler, kullanıcı kayıt ve giriş işlemleri, sosyal medya bağlantıları ekleme, güncelleme ve silme işlemleri ile kullanıcı bilgilerini getirme işlevlerini içerir.

Kulllandığım paketler

- **express**: Web sunucusu ve API geliştirme için kullanılan bir Node.js çatısı. HTTP isteklerini ve yanıtlarını kolayca yönetmenizi sağlar.
- **mysql2**: MySQL veritabanı ile etkileşim kurmak için kullanılan bir Node.js modülü. Veritabanı bağlantıları ve sorgularını yönetir.
- **body-parser**: HTTP isteklerinden gelen gövde verilerini (body) analiz etmek için kullanılır. Bu paket artık express.json() ve express.urlencoded() fonksiyonları ile sağlanmıştır, bu yüzden body-parser kullanımı gerekmez.
- **bcrypt**: Şifreleri güvenli bir şekilde hash'lemek ve doğrulamak için kullanılan bir kütüphane.
- **cors**: Cross-Origin Resource Sharing (CORS) yapılandırması için kullanılan bir paket. Farklı kökenlerden gelen HTTP isteklerini yönetir.
- **jsonwebtoken**: JSON Web Token (JWT) oluşturmak ve doğrulamak için kullanılır. Kullanıcı kimlik doğrulama ve yetkilendirme işlemlerinde kullanılır.

API Endpoints

1. Kullanıcı Kayıt (Signup)

- **Yöntem**: POST
- **URL**: /signup

Request:
```
{
 "username": "string",
 "email": "string",
 "password": "string"
}
```

- **Success response**: 200 OK - Kullanıcı başarıyla kaydedildi.
- **Error response**: 400 Bad Request - Eksik alanlar varsa veya 500 Internal Server Error - Sunucu hatası.

İşlev: Kullanıcı adı, e-posta ve şifre ile yeni bir kullanıcı oluşturur. Şifre bcrypt ile hash'lenir.

2. Kullanıcı Giriş (Login)

- **Yöntem**: POST
- **URL**: /login

Request:
```
{
 "email": "string",
 "password": "string"
}
```

- **Success response**: 200 OK - JWT token ile birlikte giriş başarılı.
- **Error response**: 400 Bad Request - Eksik alanlar varsa, 401 Unauthorized - Geçersiz e-posta veya şifre.

Açıklama: Kullanıcı e-posta ve şifre bilgilerini doğrular. Başarılı girişte JWT token döner.

3. Sosyal Medya Bağlantılarını Getir (Get Social Links)

- **Yöntem**: GET
- **URL**: /get-social-links/:userId?
- **Success response**: 200 OK - Sosyal medya bağlantıları listesi.
- **Error response**: 403 Forbidden - Yetki hatası, 500 Internal Server Error - Sunucu hatası.

İşlev: Kullanıcının sosyal medya bağlantılarını getirir.

*Adminler tüm bağlantıları görebilir,

diğer kullanıcılar sadece kendi bağlantılarını görebilir.

**Burada userId yi token içerisinden decoder ile alıyorum aynı zamanda kullanıcının göndeerdiği parametre ile kıyaslayıp doğruluğunu sağlıyorum.

4. Sosyal Medya Bağlantısı Ekle (Add Social Link)

- **Yöntem**: POST
- **URL**: /add-social-link

Request:
```
{
 "social_media_name": "string",
 "social_media_link": "string",
 "description": "string"
}
```

- **Success response**: 200 OK - Sosyal medya bağlantısı başarıyla eklendi.
- **Hata Yanıtı**: 400 Bad Request - Eksik alanlar varsa, 500 Internal Server Error - Sunucu hatası.

Açıklama: Kullanıcının sosyal medya bağlantısı eklemesini sağlar. Kullanıcının rolüne göre updated_by_role_id güncellenir. bu şekilde bu güncellemeyi adminin mi yoksa userın kendisinin mi yaptığını anlayabiliriz.

5. Sosyal Medya Bağlantısını Güncelle (Update Social Link)

- **Yöntem**: PUT
- **URL**: /update-social-link/:id

Request:

```
{
 "social_media_name": "string",
 "social_media_link": "string",
 "description": "string"
}
```

- **Success response**: 200 OK - Sosyal medya bağlantısı başarıyla güncellendi.
- **Error Response**: 403 Forbidden - Yetki hatası, 404 Not Found - Bağlantı bulunamadı, 500 Internal Server Error - Sunucu hatası.

Açıklama: Var olan bir sosyal medya bağlantısını günceller. Yetkilendirilmiş kullanıcılar bu işlemi gerçekleştirebilir. yani u işlemi hem admin rolu olan hem de userın kendisi değiştirebilir.

6. Sosyal Medya Bağlantısını Devre Dışı Bırak (Deactivate Social Link)

- **Yöntem**: PATCH
- **URL**: /delete-social-link/:id/deactivate
- **Response success**: 200 OK - Sosyal medya bağlantısı başarıyla devre dışı bırakıldı.
- **Error Response**: 403 Forbidden - Yetki hatası, 404 Not Found - Bağlantı bulunamadı, 500 Internal Server Error - Sunucu hatası.

Açıklama: Sosyal medya bağlantısını devre dışı bırakır. Kullanıcının rolüne göre isActive alanını 0 yapar.

**Genel olarak sql kayıtlarında delete işlemi tercih edilmediği için bunun yerine isActive ile yönetim sağlarız. Get Social Links'de olduğu gibi get işleminde sadece isActive değeri 1 olanlar döndürülür.

7. Kullanıcı Bilgilerini Getir (Get User Info)

- **Yöntem**: GET
- **URL**: /get-user-info
- **Başarı Yanıtı**: 200 OK - Kullanıcı bilgileri.
- **Hata Yanıtı**: 404 Not Found - Kullanıcı bulunamadı veya devre dışı, 500 Internal Server Error - Sunucu hatası.

Açıklama: JWT token ile kimlik doğrulaması yapılmış kullanıcı bilgilerini getirir.

8. Kullanıcı Bilgilerini Getir (Get User Info)

- **Yöntem**: GET
- **URL**: get-username/:userId
- **Success response**: 200 OK - Kullanıcı bilgileri.
- **Error Response**: 404 Not Found - Kullanıcı bulunamadı veya devre dışı, 500 Internal Server Error - Sunucu hatası.

Açıklama: JWT token ile kimlik doğrulaması yapılmış kullanıcı bilgilerini getirir. Bu endpoint, belirtilen userId'ye sahip kullanıcının kullanıcı adını döndürür. Sadece admin rolü olan kullanıcılar veya kendi kullanıcı bilgilerini sorgulayan kullanıcılar bu endpoint'i kullanabilir.

9. Kullanıcı Bilgilerini Güncelle (Update User Info)

- **Yöntem:** PUT
- **URL:** /update-user-info
- **Success Response:** 200 OK - Kullanıcı bilgileri başarıyla güncellendi.
- **Hata Yanıtı:**
  - 400 Bad Request - Eksik zorunlu alanlar: username, email ve password gereklidir.
  - 404 Not Found - Kullanıcı bulunamadı veya devre dışı.
  - 500 Internal Server Error - Sunucu hatası.
- **Açıklama:** JWT token ile kimlik doğrulaması yapılmış kullanıcılar, kendi bilgilerini (kullanıcı adı, e-posta, şifre) güncelleyebilir. Şifre hashlenerek db'ye kaydedilir.

10. Admin Kullanıcı Bilgilerini Güncelle (Admin Update User)

- **Yöntem:** PUT
- **URL:** /admin-update-user/
- **Success Response:** 200 OK - Kullanıcı bilgileri başarıyla güncellendi.
- **Error Response:**
  - 400 Bad Request - Eksik zorunlu alanlar: username ve email gereklidir.
  - 404 Not Found - Kullanıcı bulunamadı veya devre dışı.
  - 500 Internal Server Error - Sunucu hatası.
- **Açıklama:** Admin rolüne sahip kullanıcılar, belirtilen userId'ye sahip kullanıcının kullanıcı adı ve e-posta bilgilerini güncelleyebilir. Admin yetkisine sahip olmanız gereklidir.

11. Kullanıcıyı Devre Dışı Bırak (Admin Deactivate User)

- **Yöntem:** PATCH
- **URL:** /admin-deactivate-user/
- **Success:** 200 OK - Kullanıcı ve sosyal bağlantılar başarıyla devre dışı bırakıldı.
- **Error:**
    - 403 Forbidden - Admin kullanıcısı devre dışı bırakılamaz.
    - 404 Not Found - Kullanıcı bulunamadı.
    - 500 Internal Server Error - Sunucu hatası.
- **Açıklama:** Admin rolüne sahip kullanıcılar, belirtilen userId'ye sahip kullanıcıyı devre dışı bırakabilir. Devre dışı bırakma işlemi, kullanıcının ve sosyal bağlantılarının aktiflik durumunu günceller. Admin kullanıcıları devre dışı bırakılamaz.

Güvenlik

- **JWT**: Kullanıcı kimlik doğrulama ve yetkilendirme için JSON Web Token (JWT) kullanılır. Token, Authorization başlığı ile gönderilir.
- **CORS**: Yalnızca http://localhost:4200 adresinden gelen isteklere izin verilir. Bu, Angular'da kullandığım adrestir ve API'nizin güvenli bir şekilde iletişim kurmasını sağlar.
- **bcrypt**: Şifrelerin güvenli bir şekilde saklanmasını ve doğrulanmasını sağlar.(şifrelerin databasede açık olarak kayıt edilmesi güvenlik sorunu oluşturur)

Veritabanı Yapısı

- **Users Tablosu**:
    - ***id***: Kullanıcı kimliği (Primary Key)
    - ***username***: Kullanıcı adı
    - ***email***: E-posta adresi
    - ***password***: Şifre (hashlenmiş)
    - ***isActive***: Kullanıcının aktif olup olmadığı
    - ***role_id***: Kullanıcının rolü (1 = Admin, 2 = Kullanıcı)
- **SocialLinks Tablosu**:
    - ***id***: Bağlantı kimliği (Primary Key)
    - ***user_id***: Sosyal medya bağlantısını ekleyen kullanıcının kimliği
    - ***social_media_name***: Sosyal medya adı
    - ***social_media_link***: Sosyal medya bağlantı URL'si
    - ***description***: Bağlantı açıklaması
    - ***isActive***: Bağlantının aktif olup olmadığı
    - ***updated_by_role_id***: Bağlantıyı güncelleyen kullanıcının rol kimliği
    - ***updated_at***: Güncelleme tarihi
- Role Tablosu: ***role_id***: Kullanıcının rolü (1 = Admin, 2 = Kullanıcı)

Frontend Uygulama Dökümantasyonu: SocialLinkManagement

Proje Özeti

Bu dökümantasyon, Angular 18 kullanılarak geliştirilmiş "SocialLinkManagement" uygulamasının yapılandırmasını ve kullanılan teknolojileri detaylandırır. Uygulama, sosyal medya linklerini yönetmek için tasarlanmış bir platformdur, atomic design ve solid prensiplerine uygun olarak yapılandırılmıştır.

Kullanılan Paketler

package.json dosyasında tanımlı olan ana paketler ve neden kullanıldıkları:

- ***@angular/ (örneğin, @angular/core, @angular/forms, @angular/router)***:* Angular'ın temel modülleri, uygulamanın ana yapısını oluşturur ve sağlam bir yapı sağlar. Bu modüller, Angular'ın güçlü özelliklerinden faydalanarak, dinamik ve ölçeklenebilir uygulamalar geliştirmeme olanak tanır.
- **@angular/material:** Bu kütüphane, uygulamama modern ve uyumlu UI bileşenleri eklememi sağlar. Material Design prensiplerine uygun, kullanıcı dostu arayüzler oluşturmak, uygulamanın estetik ve fonksiyonel özelliklerini artırır.
- ***@ngrx/ ve @ngxs/***:** Durum yönetimi konusunda farklı yaklaşımlar sunan bu kütüphaneler, uygulama durumunu yönetmek ve senkronize etmek için kullanılır. **@ngrx** ve **@ngxs** ile uygulamanın durumunu daha verimli ve düzenli bir şekilde yönetebiliyorum.
- **jwt-decode:** JSON Web Token (JWT) içindeki bilgileri decode etmek için kullanılır. Bu araç, güvenli kimlik doğrulama ve yetkilendirme işlemlerini uygulamamda güvenilir bir şekilde gerçekleştirmemi sağlar.
- **ngx-toastr:** Kullanıcılara anlık bildirimler (toast) göstermek için kullanıyorum. Bu, kullanıcı deneyimini artırarak, uygulamanın etkileşimini ve geri bildirim süreçlerini iyileştirir.
- **primeng ve primeicons:** UI bileşenleri ve ikonlar sağlar. PrimeNG ile zengin ve özelleştirilebilir arayüz bileşenleri ekleyerek, kullanıcı deneyimini daha da geliştiriyorum.
- **rxjs:** Asenkron veri akışlarını yönetir ve uygulamanın reaktif programlama özelliklerini destekler. RxJS, veri akışlarını etkili bir şekilde yönetmemi ve uygulamanın performansını optimize etmemi sağlar.

devDependencies dosyasında yer alan paketler:

- **@angular-devkit/build-angular**: Angular uygulamalarının build işlemlerini yönetir.
- **typescript**: TypeScript dilini destekler ve Angular projeleri için gereklidir.

  Bu projede, en güncel teknolojilere olan ilgimi ve hevesimi ortaya koyuyorum. Genellikle eski sürümlerle çalışmış biri olarak, Angular, Node.js, PrimeNG ve Material UI gibi modern paketleri kullanarak, teknoloji dünyasındaki yenilikleri takip ediyorum. Bu proje, hem büyük bir meydan okuma hem de teknolojiye olan tutkumun bir yansıması. Yeni teknolojilere olan hevesimle, sürekli öğrenmeye ve uygulamalarımı geliştirmeye devam ediyorum.

PrimeNG Kullanım Nedeni

PrimeNG, Angular projelerinde kullanılan güçlü ve zengin bir UI bileşen kütüphanesidir. projemde PrimeNG'yi tercih etmemin başlıca nedeni, Türk geliştiriciler tarafından oluşturulmuş olmasıdır. Bu durum, kütüphanenin yerel topluluk tarafından desteklenmesini ve sürekli güncellenmesini sağlar.

PrimeNG'nin sağladığı kapsamlı bileşenler ve özelleştirme seçenekleri, kullanıcı deneyimini geliştirmek için büyük avantaj sunar. Ayrıca, yerel katkılarla desteklenen bu kütüphane, projede kullanılan teknolojiler arasında gurur verici bir seçimdir.

Uygulama Yapılandırması

main.ts:

- Angular uygulamasını başlatmak için kullanılan dosya.
- bootstrapApplication ile uygulama başlatılır.
- HTTP istemcisi, durum yönetimi, yönlendirme ve animasyonlar için gerekli sağlayıcılar module sistemi yerine standalone çalıştığım için burada da yapılandırılması gerekiyor yapılandırılır.

style.css:

- Uygulamanın stil dosyaları burada tanımlanır. ngx-toastr, Angular Material ve PrimeNG stil dosyaları dahil edilmiştir.

Atomic Design Prensipleri

Uygulama, **atomic design** prensiplerine uygun olarak yapılandırılmıştır:

1. **Atoms**:
    - Basit UI bileşenleri (butonlar, input alanları, ikonlar) gibi en küçük yapı taşları. Örneğin, Angular Material ve PrimeNG bileşenleri bu kategoride yer alır.
2. **Molecules**:
    - Birkaç atomun birleşimiyle oluşan bileşenler. Örneğin, form grupları veya menüler.
3. **Organisms**:
    - Moleküller ve atomlardan oluşan daha karmaşık bileşenler. Örneğin, auth-form ve login-form componentleri.
4. **Modules**:
    - Sayfa düzenlerini tanımlar ve organizmaları belirli bir düzen içinde yerleştirir.

    - Belirli bir sayfayı oluşturur ve tüm bileşenleri, molekülleri ve organizmaları kullanarak son kullanıcıya sunar.

5. Sağlayıcılar ve Servisler

AuthGuard:

- Kullanıcıların belirli sayfalara erişimini kontrol eder.
- Kullanıcıların oturumlarının geçerliliğini kontrol eder ve yetkilendirilmemiş kullanıcıları login sayfasına yönlendirir.

AuthService:

- Kimlik doğrulama işlemlerini yönetir.
- Token doğrulama, kullanıcı giriş çıkış işlemleri ve token ile ilgili çeşitli işlemleri sağlar.
- jwt-decode kütüphanesi kullanılarak JWT token içeriği decode edilir ve geçerlilik kontrolü yapılır.

AuthInterceptor:

- HTTP isteklerine JWT token'ını ekler.
- Token varsa, istek başlıklarına Authorization header'ı ekler.

Pipe Kullanımı

Pipe, verileri bir biçimden diğerine dönüştürmek için kullanılan Angular bileşenleridir. Projemizde, Pipe kullanarak veri dönüşüm işlemlerini gerçekleştirdik:

Directive Kullanımı

Directive, DOM elemanlarının davranışlarını veya görünümlerini değiştirmek için kullanılır. Projemizde, Directive ile kullanıcı etkileşimlerini özelleştirdik:

Dinamik Popup Kullanımı

Dinamik Popup, kullanıcı etkileşimleri sırasında açılan ve kapatılan modüler bileşenlerdir. Bu bileşenleri SOLID prensiplerine uygun olarak geliştirdik:

Neden ngxs kullanmayı tercih ettim?

NGXS kullanmanın birkaç önemli nedeni vardır. İşte bunlar:

1. StateYönetimini Basit ve Anlaşılır Hale Getirir

NGXS, Angular uygulamalarında state yönetimini daha kolay ve anlaşılır hale getirir. Redux prensiplerine dayanan bir yapı ve sadece Angular için özel olarak optimize edilmiştir. Bu, durumun merkezi bir yerde tutulmasını ve güncellenmesini sağlar.

2. Otomatik Bağlantılar ve Depenc Yönetimi

NGXS, Angular ile uyumlu bir şekilde çalışır ve dependency injection (bağımlılık enjeksiyonu) gibi Angular özelliklerini kullanır. Bu, servisler ve state yönetimi arasında kolay bir entegrasyon sağlar.

3. Kolay Debugging ve Test Edilebilirlik

NGXS, geliştirme sürecinde debug yapmayı ve test etmeyi kolaylaştırır. Durum değişikliklerini izlemek için gelişmiş araçlar ve özellikler sunar. Ayrıca, state ve action'ları kolayca test edebilirsiniz.

4. State Yönetimi İçin İleri Düzey Özellikler

NGXS, çeşitli ileri düzey özellikler sunar:

- Selectors: State içerisindeki verileri seçmek ve işlemek için kullanılır sürekli bir data'ya subscribe işleminden daha performanslıdır.
- Actions: State'i güncellemek için kullanılan eylemler.
- Plugins: NGXS, çeşitli eklentiler ve middleware'lerle özelleştirilebilir.

5. Zamanı ve Eforu Azaltır

State yönetimini NGXS ile yapılandırmak, kodun daha düzenli ve bakımı daha kolay hale gelmesini sağlar. Tek bir yerden durum yönetimi yaparak, uygulamanızın karmaşıklığını azaltabildiğim için tercihlerim arasındadır.

6. Reaktif Programlama

NGXS, reaktif programlama ile uyumludur ve RxJS kütüphanesi ile entegrasyon sağlar. Bu, asenkron veri akışlarını yönetmeyi ve olay tabanlı programlamayı kolaylaştırır projemde birçok yerde tap gibi rxjs operatörlerinin kullanımını görebilirsiniz.

Model

Model, uygulamanın durumunu temsil eden veri yapılarını tanımlar. Örneğin, bir kullanıcı durumu için model aşağıdaki gibi olabilir:

UserStateModel
user: Kullanıcı bilgileri (örneğin, username, email)
isAuthenticated: Kullanıcının kimlik doğrulama durumu
error: Hata mesajları

2. State

State, uygulamanın durumunu yöneten NGXS bileşenidir. State, durumu saklar ve bu durumu güncelleyen action'ları işleyen kodları içerir.

UserState
@State<UserStateModel>: State dekoratörü ile tanımlanır.
Actions: State'i değiştiren eylemleri tanımlar ve bu eylemlere yanıt verir.
GetUsername: Kullanıcı adını almak için API çağrısı yapar.
SetIsAuthenticated: Kullanıcının kimlik doğrulama durumunu günceller.
Signup: Yeni kullanıcı kaydı oluşturur.

3. Actions

Actions, State'te yapılacak değişiklikleri temsil eden eylemlerdir. Her action, belirli bir işlemi tetikler ve sonuçları State'e yansıtır.

GetUsername: Belirli bir userId'ye göre kullanıcı adını alır.
SetIsAuthenticated: Kullanıcının kimlik doğrulama durumunu günceller.
Signup: Kullanıcı kaydını oluşturur.

4. Service

Service, API çağrılarını gerçekleştiren ve bu çağrılardan dönen verileri işleyen sınıftır. Genellikle HTTP istekleri gönderir ve gelen yanıtları döner.

UserService
signup(request): Yeni kullanıcı kaydı oluşturur.
getUsername(userId): Kullanıcı ID'sine göre kullanıcı adını alır.

5. API Döngüsü

API döngüsü, uygulamanın dış servislerle etkileşim kurma şeklidir.

Action: Kullanıcı bir action tetikler (örneğin, GetUsername).
Service: Action, ilgili service metodunu çağırır (örneğin, UserService.getUsername).
API İsteği: Service, belirlenen API endpoint'ine HTTP isteği gönderir.
API Yanıtı: API yanıtını alır ve service içinde işleyerek döner.
State Güncelleme: Service'den gelen yanıt, state'e aktarılır (örneğin, UserState içindeki ilgili action).
UI Güncelleme: State değişiklikleri, uygulamanın UI'sını günceller.