Zaawansowany Przewodnik po Angularze

Kacper Renkel

August 11, 2024

Contents

1 Wprowadzenie

Angular to platforma i framework do budowania złożonych aplikacji webowych. W tym dokumencie omówimy zaawansowane aspekty Angulara, takie jak Guardy, Interceptory, Serwisy, Moduły, Routing, oraz inne zaawansowane funkcje.

2 Struktura aplikacji Angular

Angular opiera sie na modularnej architekturze, co pozwala na łatwe zarzadzanie złożonymi aplikacjami poprzez podział na moduły.

2.1 Moduły

Moduły w Angularze to zbiory komponentów, serwisów, dyrektyw i innych zasobów, które sa grupowane razem w celu ułatwienia zarzadzania i organizacji kodu.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { SharedModule } from './shared/shared.module';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    SharedModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Listing 1: Definicja modułu w Angularze

3 Komponenty

Komponenty sa podstawowymi elementami aplikacji Angular, które odpowiadaja za interfejs użytkownika.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-hello-world',
  template: '<h1>Hello, World!</h1>',
  styleUrls: ['./hello-world.component.css']
})
```

```
export class HelloWorldComponent { }
```

Listing 2: Przykładowy komponent

4 Serwisy

Serwisy w Angularze to klasy, które zawieraja logike biznesowa i sa wstrzykiwane do komponentów lub innych serwisów za pomoca mechanizmu Dependency Injection.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class DataService {
  private data: string[] = [];

  addData(item: string) {
    this.data.push(item);
  }

  getData(): string[] {
    return this.data;
  }
}
```

Listing 3: Przykładowy serwis w Angularze

5 Routing

Routing w Angularze umożliwia nawigacje miedzy różnymi widokami aplikacji. Angular Router to elastyczne narzedzie pozwalajace na definiowanie tras, ochrone tras za pomoca Guardów oraz ładowanie modułów na żadanie (lazy loading).

5.1 Definiowanie tras

Trasy definiuje sie w pliku modułu routingowego, gdzie każda trasa jest przypisana do określonego komponentu.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Listing 4: Przykład konfiguracji tras

5.2 Lazy Loading

Lazy Loading pozwala na ładowanie modułów dopiero wtedy, gdy sa one potrzebne, co znaczaco poprawia wydajność aplikacji.

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  {
    path: 'admin',
    loadChildren: () => import('./admin/admin.module').then(m
        => m.AdminModule)
  },
];
```

Listing 5: Przykład Lazy Loading w Angularze

6 Guardy

Guardy w Angularze pozwalaja na kontrolowanie dostepu do określonych tras na podstawie warunków logicznych. Angular oferuje kilka typów Guardów:

- `CanActivate` – sprawdza, czy użytkownik może przejść do określonej trasy.
- `CanDeactivate` – sprawdza, czy użytkownik może opuścić określona trase.
- `CanLoad` – sprawdza, czy moduł może być załadowany na żadanie.

6.1 Implementacja CanActivate

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { AuthService } from './auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService, private
     router: Router) {}
```

```
  canActivate(): boolean {
    if (this.authService.isAuthenticated()) {
      return true;
    } else {
      this.router.navigate(['/login']);
      return false;
    }
  }
}
```

Listing 6: Przykład implementacji `CanActivate`

7 Interceptory

Interceptory w Angularze sa cześcia mechanizmu HTTP Client i umożliwiaja przechwytywanie oraz modyfikacje żadań i odpowiedzi HTTP.

7.1 Tworzenie Interceptora

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent
    } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  intercept(req: HttpRequest<any>, next: HttpHandler):
     Observable<HttpEvent<any>> {
    const authToken = 'my-auth-token';
    const authReq = req.clone({
      setHeaders: { Authorization: `Bearer ${authToken}` }
    });
    return next.handle(authReq);
  }
}
```

Listing 7: Przykład implementacji Interceptora

7.2 Rejestrowanie Interceptora

Aby zarejestrować Interceptor, należy dodać go do tablicy `providers` w module aplikacji.

```
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { AuthInterceptor } from './auth.interceptor';

@NgModule({
```

```
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor,
      multi: true },
  ],
})
export class AppModule { }
```

Listing 8: Rejestracja Interceptora

8 Formularze

Angular oferuje dwa podejścia do tworzenia formularzy: Template-Driven i Reactive Forms. Reactive Forms daja wieksza kontrole nad walidacja i stanem formularza.

8.1 Reactive Forms

Reactive Forms pozwalaja na bardziej dynamiczne zarzadzanie formularzami poprzez zdefiniowanie formy w TypeScript zamiast HTML.

```
import { Component } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/
   forms';

@Component({
  selector: 'app-registration',
  templateUrl: './registration.component.html'
})
export class RegistrationComponent {
  registrationForm = new FormGroup({
    username: new FormControl('', Validators.required),
    email: new FormControl('', [Validators.required,
      Validators.email]),
    password: new FormControl('', [Validators.required,
      Validators.minLength(6)]),
  });

  onSubmit() {
    console.log(this.registrationForm.value);
  }
}
```

Listing 9: Przykład Reactive Form

9 HttpClient

`HttpClient` w Angularze jest narzedziem do komunikacji z serwerami za pomoca żadań HTTP. Umożliwia obsługe różnych metod HTTP, takich jak `GET`, `POST`, `PUT`, `DELETE`.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ApiService {

  constructor(private http: HttpClient) { }

  getData(): Observable<any> {
    return this.http.get('https://api.example.com/data');
  }

  postData(data: any): Observable<any> {
    return this.http.post('https://api.example.com/data',
      data);
  }
}
```

Listing 10: Przykład użycia HttpClient w Angularze

10 Middleware w Angularze

Middleware w Angularze może być zaimplementowany jako Interceptor lub jako mechanizm Guard. Oba podejścia pozwalaja na przechwytywanie i manipulowanie żadaniami HTTP lub nawigacja.

11 JWT (JSON Web Token)

JWT jest standardem używanym do bezpiecznego przekazywania informacji miedzy serwerem a klientem. W Angularze JWT można łatwo zintegrować z aplikacja przy użyciu Interceptorów do automatycznego dołaczania tokenów do nagłówków żadań.

11.1 Implementacja JWT w Angularze

Aby zintegrować JWT w Angularze, należy stworzyć serwis do obsługi tokenów oraz Interceptor do automatycznego dołaczania tokenu do żadań HTTP.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class JwtService {
  private readonly TOKEN_KEY = 'auth-token';
```

```
  saveToken(token: string) {
    localStorage.setItem(this.TOKEN_KEY, token);
  }

  getToken(): string | null {
    return localStorage.getItem(this.TOKEN_KEY);
  }

  clearToken() {
    localStorage.removeItem(this.TOKEN_KEY);
  }
}
```

Listing 11: Przykład serwisu do obsługi JWT

12 Podsumowanie

Angular jest potężnym narzedziem do tworzenia złożonych aplikacji webowych. Dzieki zaawansowanym funkcjom takim jak Guardy, Interceptory, Lazy Loading, oraz rozbudowanemu systemowi modułów, Angular umożliwia tworzenie skalowalnych i dobrze zorganizowanych aplikacji. Dokument ten przedstawia kluczowe aspekty Angulara, które sa niezbedne dla każdego dewelopera pragnacego tworzyć nowoczesne i zaawansowane aplikacje.