ANGULAR 2+

KONSUMOWANIE USŁUG

Waldemar Korłub

Zaawansowane programowanie w języku skryptowym

KASK ETI Politechnika Gdańska

Konsumowanie usług

- Aplikacja front-endowa wykorzystuje usługi po stronie serwera (back-endu)
  - Pobieranie danych
  - Zapisywanie informacji wprowadzonych przez użytkownika
  - Framework Angular dostarcza klienta HTTP w postaci klasy HttpClient
    - Instancja klasy HttpClient może zostać wstrzyknięta jako zależność poprzez argumenty konstruktora

Konsumowanie usług

- Instancje klasy HttpClient mogą zostać wstrzyknięte do klas komponentów
  - …i tam wykorzystane do pobierania/zapisywania danych
- Te same usługi aplikacji serwerowej są często wykorzystywane w wielu widokach
  - …i w różnych kontekstach
  - Pobranie danych książki przyda się w widoku szczegółów książki, w widoku edycji i np. w widoku zamówienia
- Wywołania klienta HTTP w klasie komponentu są trudne do wielokrotnego wykorzystania (ang. *code reuse*)
  - Komponent jest związany z konkretnym fragmentem interfejsu
- Wywołania usług sieciowych typowo umieszcza się w klasach serwisowych

Klasa serwisowa


```
@Injectable()
export class BooksService {
 private booksUrl = '/books’;
 private baseUrl = 'https://back-end.example.com';

 constructor(private http: HttpClient) { }

 public getBooks(limit?: number): Observable<Book[]> {
   let params = new HttpParams();

   if (limit) params =
     params.set('limit', limit.toString());

   return this.http.get<Book[]>(
    this.baseUrl + this.booksUrl, {params: params});
 }
}
```

Klasa serwisowa – inne żądania


```
@Injectable()
export class BooksService {
  //...

  public getBook(id: string): Observable<Book> {
    return this.http.get<Book>(
      this.baseUrl + this.booksUrl + '/' + id);
  }

  public save(book: Book): Observable<Object> {
    return this.http.put(
      this.baseUrl + this.booksUrl + '/' + book.id, book);
  }
}
```

Observable

- Klasa Observable pochodzi z bibliteki RxJS
- Biblioteka RsJS jest implementacją specyfikacji ReactiveX
  - API umożliwiające programowanie asynchroniczne z wykorzystaniem obserwowanych strumieni danych
- Żądania HTTP są wysyłane asynchronicznie
  - Oczekiwanie na odpowiedź nie blokuje interfejsu użytkownika
- Obiekt Observable umożliwia nasłuchiwanie na odpowiedź z serwera bez blokowania interfejsu

Observable – najważniejsze metody

- subscribe(next?, error?, complete?)
  - Nasłuchiwanie na rezultat asynchronicznej operacji
    - Next – operacja do wykonania w razie sukcesu
    - Error – operacja do wykonania w razie błędu
    - Complete – operacja do wykonania po zakończeniu asynchronicznej operacji (niezależnie od wyniku: sukces/błąd)
- map(project)
  - Mapowanie rezultatu operacji przed przekazaniem wyniku do kolejnego nasłuchującego obiektu
- catch(selector)
  - Obsługa błędów

Konsumowanie usług – komponent


```
@Component({ ... })
export class BookDetailsComponent implements OnInit {
  book: Book;

  constructor(private bookService: BooksService,
              private route: ActivatedRoute) { }

  ngOnInit() {
    const id = this.route.snapshot.paramMap.get('id');
    this.bookService.getBook(id).subscribe(
      book => this.book = book
    );
  }
}
```

Przecięcia

- Pewne aspekty powtarzają się dla wielu różnych żądań HTTP emitowanych przez aplikację
  - np. aspekt kontroli dostępu
  - Po zalogowaniu użytkownika informacja poświadczająca jego tożsamość powinna być dołączona do wszystkich dalszych żądań
  - Można to robić przy każdym wywołaniu klienta HTTP z osobna
    - …ale skutkuje to dużą ilością powtórzonego kodu
      - Jeśli w przyszłości będzie trzeba coś zmienić/poprawić, to zmiany będą konieczne w wielu miejscach aplikacji
- Przecięcia umożliwiają zarządzanie globalnymi aspektami wszystkich żądań HTTP

Przecięcia


```
@Injectable()
export class AuthInterceptor implements
                              HttpInterceptor {

  constructor(private router: Router) {
  }

  intercept(request: HttpRequest<any>,
            next: HttpHandler):
                              Observable<HttpEvent<any>> {
    //...
  }

}
```

Rejestrowanie przecięcia

- W definicji modułu:

```
@NgModule({
  imports: [ ... ],
  declarations: [ ... ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptor,
      multi: true
    },
    ...
  ],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

Przecięcie: metoda intercept


```
intercept(request: HttpRequest<any>, next: HttpHandler):
                                        Observable<HttpEvent<any>> {
 if (this.getAccessToken() != null) {
   request = request.clone({
     setHeaders: {
       Authorization: `Bearer ${this.getAccessToken()}`
     }
   });
 }
 return next.handle(request).do(() => {},
  (err: any) => {
   if (err instanceof HttpErrorResponse && err.status==401)
     this.router.navigate(
      ['sign-in’], {queryParams: {tokenExpired: true}});
   });
}
```

Pytania?