ANGULAR 2+

Waldemar Korłub

Aplikacje i Usługi Internetowe
KASK ETI Politechnika Gdańska

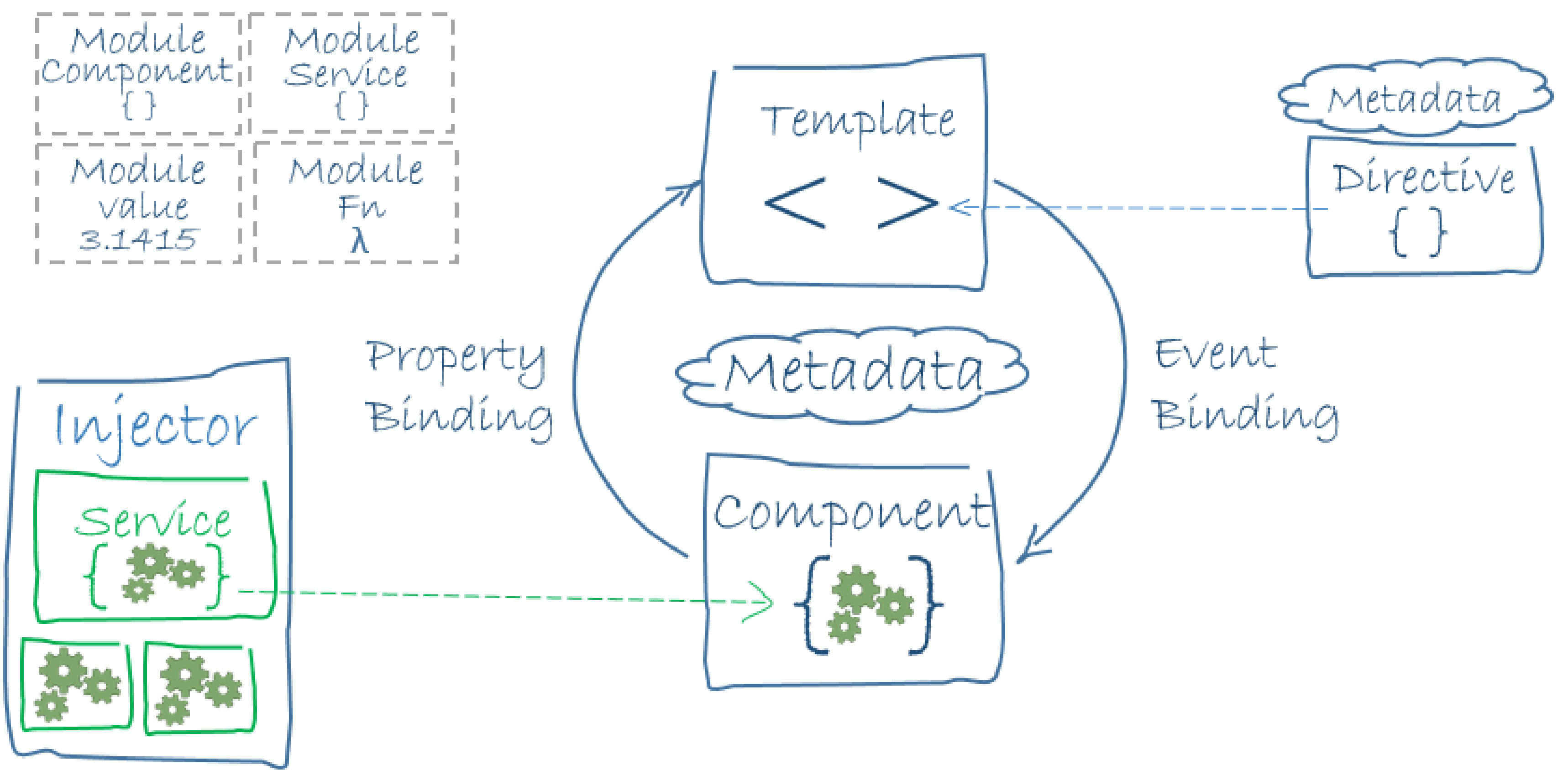
2

Architektura

Architektura frameworka Angular

- Aplikacja wykorzystuje język HTML i style CSS
- Poza standardowymi znacznikami HTML występują również dodatkowe specjalne znaczniki
- Dodatkowe znaczniki umożliwiają wykorzystanie *komponentów* definiowanych w projekcie
  - Komponent opisuje fragment widoku oraz jego zachowania
  - Komponent może być zbudowany z innych komponentów
    - Hierarchiczna struktura widoków
- Logika aplikacji implementowana w *serwisach*
- Komponenty i serwisy grupowane w *modułach*

Architektura



Architektura

- Moduły (ang. modules)
- Komponenty (ang. components)
- Szablony (ang. templates)
- Metadane (ang. metadata)
- Wiązanie danych (ang. data binding)
- Dyrektywy (ang. directives)
- Serwisy (ang. services)
- Wstrzykiwanie zależności (ang. dependency injection)

Moduły

- Aplikacje w Angularze dzieli się na moduły odpowiadające poszczególnym funkcjonalnościom
- Każda aplikacja posiada główny moduł o nazwie AppModule
  - Małe aplikacje mogą posiadać tylko jeden moduł, duże aplikacje – setki modułów
- Moduły definiowane jako klasy z dekoratorem @NgModule
  - Dekoratory umożliwiają dołączenie metadanych do klasy

NgModule – metadane

- Najważniejsze metadane dekoratora NgModule:
  - declarations – lista komponentów wykorzystywanych do budowania widoków aplikacji
  - exports – lista klas/komponentów, które powinny być dostępne do wykorzystania przez inne moduły
  - imports – lista modułów, których wyeksportowane klasy są wykorzystywane w module bieżącym
  - providers – lista dostawców umożliwiających budowanie instancji serwisów do wykorzystania w całej aplikacji (we wszystkich modułach)
  - bootstrap – komponent reprezentujący główny widok aplikacji (do niego ładowane są wszystkie inne widoki), używany tylko dla AppModule

Przykład AppModule (app.module.ts)


```
@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule
  ],
  declarations: [
    AppComponent,
    NavbarComponent,
    SignInComponent,
    BooksComponent,
    //...
  ],
  providers: [
    BooksService,
    AuthService,
    CartService,
    OrdersService,
    IdentityService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Moduły Angulara a moduły w języku JavaScript

- W języku JavaScript każdy plik jest *modułem*
- Moduły w języku JavaScript i moduły frameworka Angular to dwie osobne koncepcje
  - JavaScript: moduł = plik
  - Angular: moduł = zbiór komponentów i serwisów
- W typowym projekcie używamy obu mechanizmów modułów równocześnie
- Wspólne nazewnictwo (moduł/import/export) dla odrębnych mechanizmów może wywoływać dezorientację

Moduły w języku JavaScript

- Słowo kluczowe export w plikach (modułach) .js określa obiekty, które dany moduł udostępnia innym modułom (plikom .js):
  export class AppModule { }
  export class AppComponent {…}

- Inne pliki (moduły) .js mogą korzystać z tych obiektów korzystając z dyrektywy import:
  import {AppComponent} **from** **"./app.component"**;

Uruchomienie aplikacji za pomocą modułu AppModule

- Należy zaimportować klasę AppModule z pliku (modułu) JavaScript
- Wywołanie metody bootstrapModule inicjuje moduł Angulara:

```
import { AppModule } from './app/app.module';
platformBrowserDynamic().
    bootstrapModule(AppModule);
```

Komponenty (ang. components)

- Reprezentują fragmenty widoków składające się na interfejs aplikacji
- Definiowane jako klasy z dekoratorem @Component
- Są reprezentowane przez dodatkowe znaczniki umieszczane w kodzie HTML, np.:

```
<body>
  <my-app>Loading...</my-app>
</body>
```

Komponenty (ang. components)

- Komponent określa:
  - Szablon używany do zbudowania fragmentu widoku (tagi HTML obejmujące też tagi innych komponentów)
  - Dane do prezentacji (model)
  - Zachowania (funkcje obsługi zdarzeń)
- @Component – najważniejsze metadane:
  - selector – selektor CSS określający które znaczniki na stronie mają zostać wypełnione zawartością komponentu
  - templateUrl – ścieżka do pliku .html z szablonem
  - providers – lista dostawców serwisów, z których korzysta komponent

Przykładowy komponent


```
@Component({
  selector: 'app-books',
  templateUrl: './books.component.html',
  styleUrls: ['./books.component.css']
})
export class BooksComponent implements OnInit {

  @Input() limit: number;

  books: Observable<Book[]>;

  constructor(private booksService: BooksService) { }

  ngOnInit() {
    this.books = this.booksService.getBooks(this.limit);
  }
}
```

Szablony (ang. templates)

- Wykorzystywane przez komponenty do wygenerowania treści w oknie przeglądarki
- Składnia opiera się na składni języka HTML
- Zawierają dodatkowe elementy
  - Znaczniki umożliwiające dołączanie kolejnych komponentów do widoku
  - Dyrektywy umożliwiające sterowanie procesem generowania wynikowego kodu HTML

Przykładowy szablon


```
<tbody>
<tr *ngFor="let book of books | async; index as i">
  <td>{{i+1}}</td>
  <td>
    <a [routerLink]="['/books',book.id]">{{book.title}}</a>
  </td>
  <td>
    {{book.authors[0].name}} {{book.authors[0].surname}}
    <span *ngIf="book.authors.length > 1">
      <i>et al</i>
    </span>
  </td>
  <td>{{book.publicationDate | date}}</td>
  <td>{{book.amount}}</td>
</tr>
</tbody>
```

Wiązanie danych (ang. data binding)

- Bez wykorzystania wiązania danych deweloper musi ręcznie umieszczać w drzewie DOM dane uzyskane w kodzie JavaScript (np. wyniki żądania AJAXowego)
- …oraz ręcznie pobierać wartości z drzewa DOM w reakcji na zdarzenia emitowane przez użytkownika (np. keyup, click itd.)
- Przykładowo: walidacja formularza
  - Po kliknięciu przycisku przez użytkownika pobranie danych z pól, a następnie dodanie komunikatów błędów na stronie
- Wiązanie danych automatyzuje proces wymiany informacji pomiędzy komponentem i drzewem DOM

4 formy wiązania danych

- Interpolacja wartości: {{...}}
- Wiązanie właściwości elementów DOM: [property]
- Wiązanie obsługi zdarzenia: (event)
- Dwukierunkowe wiązanie danych: [(...)]

DOM

{{value}}

[property] = "value"

(event) = "handler"

[(ng-model)] = "property"

COMPONENT

Dyrektywy (ang. directives)

- Dokumenty HTML posiadają statyczną strukturę
- Szablony widoków Angulara są dynamiczne
  - Wynikowy HTML jest efektem przetworzenia szablonu zgodnie z umieszczonymi w nim dyrektywami
- Dwie grupy dyrektyw:
  - Strukturalne – modyfikują strukturę dokumentu poprzez dodawanie, usuwanie lub zamianę elementów, np.:
    - *ngFor – dodawanie elementów w pętli
    - *ngIf – warunkowe wyświetlenie elementu
  - Dyrektywy w postaci atrybutów – modyfikują wygląd lub zachowanie istniejących już elementów

Serwisy (ang. services)

- Logika aplikacji nie powinna być implementowana w klasach komponentów
  - Komponent działa w kontekście konkretnego szablonu widoku – trudno ponownie wykorzystać logikę zaszytą w klasie komponentu w innych miejscach aplikacji
  - Kontroler powinien definiować pola i metody na potrzeby wiązania danych, a logikę delegować do serwisów
- Serwisy implementują logikę aplikacji w sposób niezależny od interfejsu
  - Łatwe wykorzystanie w wielu różnych kontekstach
- Serwisy są dostarczane do komponentów na drodze wstrzykiwania zależności

Wstrzykiwanie zależności (ang. *dependency injection*)

Klasa nie odpowiada za pozyskanie swoich zależności,

zamiast tego

zależności są do niej dostarczane z zewnątrz.

Wstrzykiwanie zależności – konsekwencje

- Klasa nie musi wiedzieć jak pozyskać zależność
  - klasa nie jest uzależniona od sposobu pozyskania wymaganego komponentu (np. obiekt fabryki, builder, klasy narzędziowe)
  - klasa jest uzależniona wyłącznie od wymaganego komponentu
- Klasa nie musi podejmować decyzji, którą implementację zależności wykorzystać, jeśli jest ich wiele
  - …i nie jest związana na stałe z jedną konkretną implementacją!
  - komponent dostarczający zależność wybiera implementację i ją wstrzykuje
  - łatwość zamiany implementacji zależności wykorzystywanej przez klasę, bez konieczności modyfikowania samej klasy

Wstrzykiwanie zależności – konsekwencje

- Klasa nie musi zarządzać (inaczej: przejmować się) cyklem życia swoich zależności
  - ta odpowiedzialność spada na komponent dostarczający zależności
  - upraszcza to projekt klasy wykorzystującej zależności
- Wstrzykiwanie zależności ułatwia testowanie jednostkowe aplikacji
  - Możliwość odizolowania komponentu od jego zależności, wykorzystania obiektów-zaślepek (ang. mock)

Wstrzykiwanie w Angularze

- Wstrzykiwanie zależności jest nieodzownym mechanizmem frameworka Angular
  - Wykorzystywane wewnętrznie przez sam framework
  - …oraz w komponentach opracowywanych przez autora aplikacji opartej na Angularze
- Za dostarczenie zależności odpowiada komponent *injector*
  - *Injector* utrzymuje instancje serwisów, które mogą być wstrzykiwane
  - Instancje pozyskiwane są przy użyciu dostawców (ang. *providers*)
    - Należy w projekcie zdefiniować dostawców dla wymaganych serwisów
- Wstrzykiwanie przez parametry konstruktora

TypeScript:

Pola klasy w parametrach konstruktora

- Jawnie zdefiniowane pola z inicjalizacją w konstruktorze:

```
class Person {
  private firstName: string;
  private lastName: string;

  constructor(first: string, last: string) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

- Pola definiowane na poziomie konstruktora:

```
class Person {
  constructor(private firstName: string,
              private lastName: string) { }
}
```

Szablony i wiązanie danych

Interpolacja: {{…}}

- Umożliwia wyznaczenie wartości wykorzystywanej w widoku na podstawie wyrażenia, np.:
  <**h3**>{{imgTitle}}</**h3**>
  <**img src="{{imgUrl}}"**>
- Wyrażenie najczęściej odnosi się do pól/własności klasy komponentu
- Wyrażenie w nawiasach {{…}} jest konwertowane na ciąg znaków przed umieszczeniem w widoku

Wyrażenia (ang. *template expressions*)

- Wyrażenie może realizować dodatkowe operacje, np.:
  <p>Przekroczono próg o {{getScore()-getTreshold()}} punktów.</p>
- Wyrażenie nie powinno powodować efektów ubocznych
- Wyrażenie powinno być szybkie do wykonania
- Wyrażenia powinny być możliwie krótkie i proste
  - Złożoną logikę należy umieścić w metodzie komponentu i wywoływać gotową metodę w wyrażeniu
- Wyrażenie powinno być idempotentne

Wiązanie własności

- Wartości wyrażeń można również wiązać z właściwościami elementów drzewa DOM oraz właściwościami komponentów, np.:

```
<button [disabled]="isUnchanged">Cancel</button>
<img [src]="imgUrl">
<app-book-detail [book]="selectedBook">
```

- Takie wiązanie jest jednokierunkowe
  - Zmiana wartości wyrażenia powoduje zmianę wartości właściwości, ale nie odwrotnie
- Wiązania odnoszą się do właściwości elementów drzewa DOM a nie do atrybutów tagów HTMLowych

Właściwości węzłów DOM a atrybuty elementów HTML

- Niektóre atrybuty elementów HTML mają bezpośrednie odzwierciedlenie w właściwościach węzłów drzewa DOM, np. id, src
- Dla niektórych atrybutów nie istnieją odpowiadające im właściwości, np. colspan, aria
- Niektóre właściwości nie posiadają odpowiadających im atrybutów, np. textContent
- Niektóre atrybuty mają odzwierciedlenie we właściwościach, ale jest ono nieintuicyjne
  - np. disabled:
    - W HTMLu – sama obecność atrybutu powoduje nieaktywność elementu (bez względu na wartość)
    - W drzewie DOM – disabled=true powoduje nieaktywność

Wiązanie dla atrybutów

- Wiązania odnoszą się do właściwości elementów drzewa DOM a nie do atrybutów tagów HTMLowych
- ...poza jednym wyjątkiem:
  <td [attr.colspan]="getColsCount()">One-Two</td>
- Jeśli atrybut nie posiada odpowiednika wśród właściwości DOM jedyny sposób na jego zmianę, to bezpośrednia zmiana atrybutu – prefiks: attr.

Wiązanie właściwości CSS

- Obiektem wiązania mogą być klasy CSS, np.:
  ```
  <div [class.special]="isSpecial">...</div>
  ```

- Albo poszczególne właściwości CSS, np.:
  ```
  <button [style.color]="isSpecial ? 'red': 'green'">
  <button
    [style.background-color]="canSave ? 'cyan':'grey'">
  ```

{{...}} czy [property]="..."?

- Interpolacja {{...}} zawsze konwertuje wynik na postać tekstową
- Wiązanie właściwości [property] zachowuje typ wiązanego wyrażenia

Wiązanie zdarzeń

- Umożliwia wywoływanie funkcji obsługi zdarzeń zdefiniowanych w klasie komponentu w reakcji na akcje użytkownika, np.:

```
<button (click)="onSave()">Save</button>
```

Wiązanie dwukierunkowe

- Zmiana wartości związanego pola zmienia wartość właściwości
- Zmiana wartości właściwości zmienia wartość pola
- Szczególnie przydatne przy pracy z formularzami
  - Dwukierunkowe wiązanie zbudowane „ręcznie”:
```
<input [value]="name"
          (input)="name=$event.target.value" >
```
    - Różne tagi formularzy wymagają użycia różnych atrybutów
  - Z wykorzystaniem ngModel:
```
<input [(ngModel)]="name">
```

ngModel

- Reprezentuje wartość pola w formularzu
  - Umożliwia np. przypisanie wartości początkowej
- Równocześnie pozwala nasłuchiwać na zmiany
  - Zmiana wartości wykonana przez użytkownika jest automatycznie odzwierciedlona w polu klasy komponentu
- Obsługuje wszystkie podstawowe elementy formularzy HTML
  - Ukrywa różnice pomiędzy nimi, np.:
    - input → value
    - select/option → selected

Routing – nawigacja w aplikacji

Routing

- Typowe aplikacje internetowe składają się z wielu widoków, pomiędzy którymi nawiguje użytkownik
- RouterModule umożliwia definiowanie adresów, które spowodują wyświetlenie wybranych komponentów (widoków) aplikacji
- Znacznik <base href="/"> w sekcji <head> pliku index.html określa bazową ścieżkę dla adresów w ramach aplikacji

Przykładowy routing


```
const routes: Routes = [
  {path: '', redirectTo: '/sign-in', pathMatch: 'full'},
  {path: 'sign-in', component: SignInComponent},
  {path: 'dashboard', component: DashboardComponent},
  {path: 'books', component: BooksComponent},
  {path: 'books/:id', component: BookDetailsComponent},
  {path: 'books/:id/edit', component: BookEditComponent},
  {path: 'authors', component: AuthorsComponent},
  {path: 'cart', component: CartComponent}
];

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

Nawigacja

- Nawigacja z użyciem dyrektyw routerLink:

```
<a routerLink="/dashboard"
   routerLinkActive="active">Dashboard</a>
<a routerLink="/cart">Koszyk</a>
```

- Komponent zdefiniowany w routingu zostanie wyświetlony poniżej elementu: <router-outlet></router-outlet>

Programistyczna nawigacja


```
@Component({ ... })
export class CartComponent implements OnInit {

  constructor(private router: Router) {
  }

  placeOrder(): void {
    //...
    this.router.navigate(['orders']);
  }
}
```

Obsługa parametrów routingu

- Przykładowy routing z parametrami:

```
const routes: Routes = [
  //...
  {
    path: 'books/:id',
    component: BookDetailsComponent
  },
  {
    path: 'books/:id/edit',
    component: BookEditComponent
  }
];
```

Obsługa parametrów routingu

- W klasie komponentu:

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

  constructor(private router: Router,
              private route: ActivatedRoute) { }

  ngOnInit() {
    const id = this.route.snapshot.paramMap.get('id');
    this.book = //...wczytanie z back-endu...
  }
}
```

Nawigacja z parametrami

- W szablonie (books.component.html):

```
<table>
  <tbody>
  <tr *ngFor="let book of books; index as i">
    <td>{{i+1}}</td>
    <td>
      <a [routerLink]="['books', book.id]">
        {{book.title}}
      </a>
    </td>
  </tr>
  </tbody>
</table>
```

Nawigacja z parametrami

- Programistycznie w klasie komponentu:

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

  constructor(private router: Router,
              private route: ActivatedRoute) {
  }

  cancel(): void {
    this.router.navigate(['/books', this.book.id]);
  }

  save(): void {
    //...zapisanie zmian w formularzu...
    this.router.navigate(['/books', this.book.id]);
  }
}
```

Konsumowanie usług sieciowych

Konsumowanie usług sieciowych

- Dane prezentowane w aplikacji front-endowej typowo pobierane są z serwera (z back-endu)
- Informacje wprowadzane przez użytkownika są zapisywane na serwerze
  - Dane z formularzy na stronie, zawartość koszyka/zamówienia itd.
- Aplikacja back-endowa udostępnia swoje funkcje w postaci usług sieciowych (ang. *web services*)
- Front-end konsumuje usługi sieciowe
  - Wrócimy do tej kwestii po wprowadzeniu do usług sieciowych po stronie serwera

Dla porównania: AngularJS 1.x

AngularJS 1.x

- Oparty na języku JavaScript, a nie TypeScript
- Podstawowe składowe:
  - Widok – interfejs użytkownika, generowany na podstawie szablonu
  - Kontroler – obsługuje akcje użytkownika
  - Zasięg (ang. *scope*) – związany z widokiem, zawiera model danych oraz funkcje obsługi zdarzeń
  - Serwisy – obiekty pomocnicze, oferują funkcje do wykorzystania w kontrolerach (lub w innych serwisach)
  - Routing – pozwala na budowanie aplikacji składających się z wielu widoków

AngularJS 1.x – istotne koncepcje

- Wiązanie danych (ang. *data binding*) – łączenie informacji z modelu z elementami interfejsu
  - Może być dwukierunkowy
  - Aktualizacja modelu → aktualizacja interfejsu
  - Zmiana w interfejsie (np. edycja pola w formularzu) → aktualizacja modelu
- Wstrzykiwanie zależności
  - Dostarczanie wymaganych serwisów/obiektów/stałych do kontrolerów (lub innych serwisów)
  - Kontroler zamiast samodzielnie pobierać zależność, otrzymuje ją z zewnątrz

AngularJS 1.x – kontroler, zasięg

- Definicja kontrolera:

```
angular.module('starter.controllers', [])

.controller(
    'ProductsCtrl',
    function($scope, $http, baseUrl) {
        $http.get(baseUrl + '/products').then(
        function(response){
            $scope.products = response.data;
        });
    }
);
```

AngularJS 1.x – widoki

- Budowane w oparciu o szablony
- Odwołania do danych umieszczonych przez kontroler w zasięgu (scope) przy użyciu konstrukcji {{*wyrażenie*}}
- Instrukcje sterujące: ng-repeat, ng-if, ng-show, ng-hide

```
<ul ng-show="products.lenght">
    <li ng-repeat="product in products">
        {{product.name}}
    </li>
</ul>
```

AngularJS 1.x: przypisanie kontrolera do widoku

- Na poziomie szablonu:

```
<div ng-controller="ProductsCtrl">
...
</div>
```

- Lub na poziomie routingu:

```
$stateProvider
  .state('app.products', {
    url: '/products',
    templateUrl: 'templates/products.html',
    controller: 'ProductsCtrl'
  })
```

Bibliografia

- Wykorzystano materiały na licencji CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/) z oficjalnej dokumentacji frameworka Angular by Google (https://angular.io/docs/ts/latest/)

Pytania?