convista:

Better People.
Better Transformation.

Angular

kurs wprowadzający

Wrocław / 17.05.2023

Agenda

01. Podstawowe pojęcia związane z Angularem

- Komponenty
- Moduły
- Biblioteki

02. Interakcja komponentów

- Text interpolation
- Property binding
- Event binding
- Inputs and Outputs
- Structural directives

03. Dependency Injection i routing

- Dependency Injection
- Routing

04. Project Introduction

convista:

Podstawowe pojęcia związane z Angularem

01

convista:

Komponenty

„Angular zawiera oparty na komponentach framework do budowania skalowalnych aplikacji internetowych”

Budowa komponentu

Selektor

Selektor CSS, który jednoznacznie identyfikuje komponent w kodzie HTML

Szablon

Kod HTML w wersji inline-template lub w osobnym pliku .html

Style

Kod CSS* w wersji: inline-template lub w osobnych plikach .css*

* CSS / SCSS / SASS / LESS

Budowa komponentu

```
interface Component {

    /* The CSS selector */
    selector?: string;

    /* The relative path or absolute URL of a template file for an Angular component */
    templateUrl?: string;

    /* An inline template for an Angular component */
    template?: string;

    /* One or more relative paths or absolute URLs for files containing CSS stylesheets */
    styleUrls?: string[];

    /* One or more inline CSS stylesheets to use in this component */
    styles?: string[];

}
```

Budowa komponentu

app.component.ts:

```
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: [ './app.component.css' ]
})
```

app.component.html:

```
<app-root></app-root>
```

Jak wygenerować komponent?

Sposób 1:

- *Wygenerować nowy komponent za pomocą CLI (Command Line Interface) poleceniem*
- *np.* `ng generate component my-component`

Sposób 2:

- Ręcznie utworzyć klasę komponentu i szablon HTML
- Zarejestrować komponent w module aplikacji


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

@Component({
  selector: 'app-my-modal',
  templateUrl: './my-modal.component.html',
  styleUrls: ['./my-modal.component.scss']
})
export class MyModalComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```


```
<p>my-modal works!</p>
```

convista:


CSS
3


Sass

Moduły

Moduły opis

Co to są moduły?

moduły w Angularze służą do organizowania kodu i dzielenia aplikacji na mniejsze, zarządzalne części.

Każdy moduł może zawierać składniki (components), serwisy (services), dyrektywy (directives), a także importować inne moduły.

Składają się z:

- Klas modułu, które definiują funkcjonalność
- Dekoratora @NgModule, który konfiguruje moduł

Jak tworzyć moduły w Angularze?

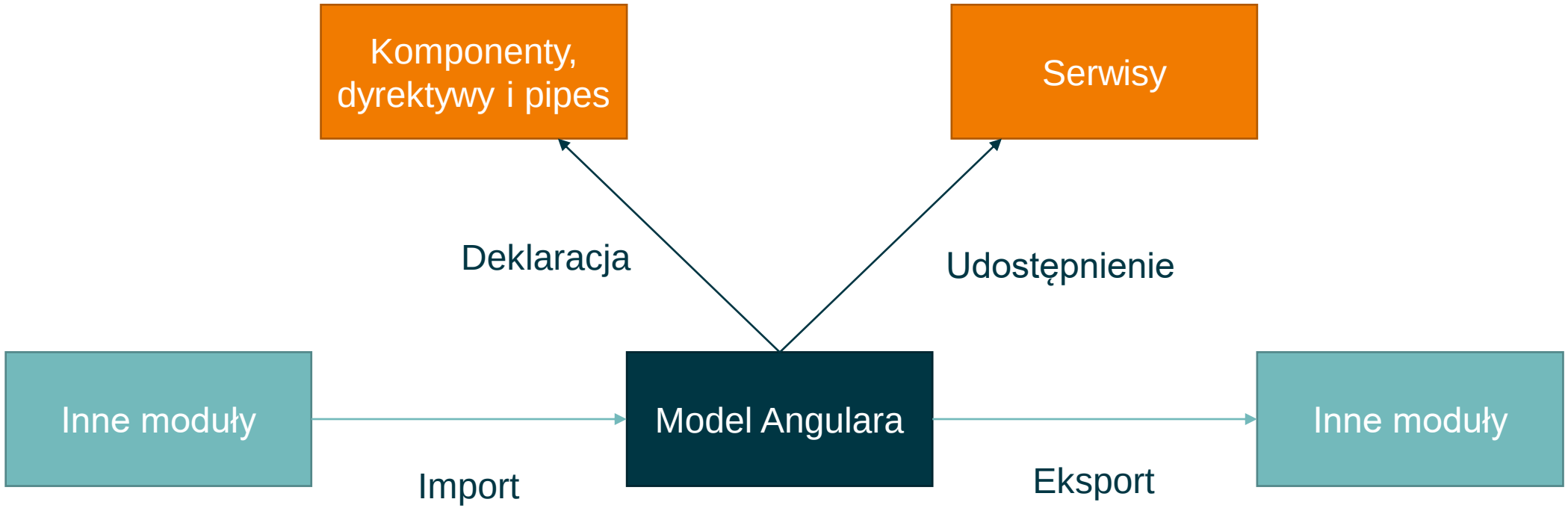
Sposób 1:

- Wygenerować nowy moduł za pomocą CLI (Command Line Interface) poleceniem
- np. `ng generate module my-module`

Sposób 2:

- Ręcznie utworzyć klasę modułu i zarejestrować ją w AppModule lub innym module

Moduły Angulara



```ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MyComponent } from './my-component/my-
component.component';

@NgModule({
 declarations: [MyComponent],
 imports: [CommonModule],
 exports:[MyComponent],
})
export class MyModule { }
```

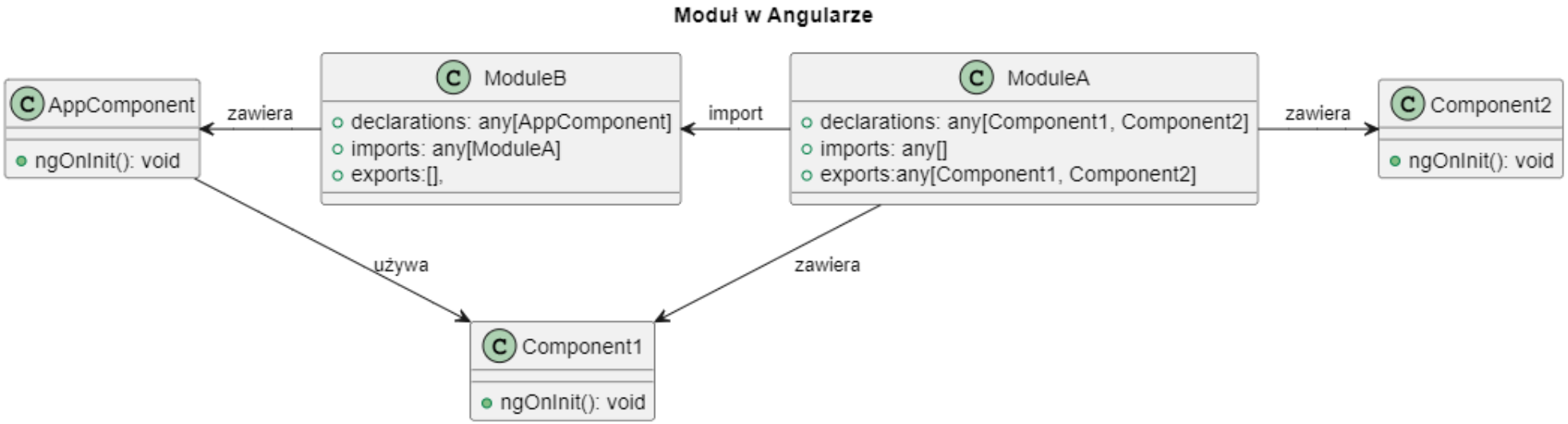
```ts
import { Component, OnInit } from '@angular/core';
import { MyModalComponent } from '../my-modal/my-modal.component';
import { NgbModal } from '@ng-bootstrap/ng-bootstrap';

@Component({
  selector: 'app-mycomponent',
  templateUrl: './mycomponent.component.html',
  styleUrls: ['./mycomponent.component.scss']
})
export class MyComponent implements OnInit {

  constructor(private modalService: NgbModal) { }
  open() {
    this.modalService.open(MyModalComponent);

  }


  ngOnInit(): void {
  }

}
```

convista:

```ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MyComponent } from './my-component/my-
component.component';

@NgModule({
 declarations: [MyComponent],
 imports: [CommonModule],
 exports:[MyComponent],
})
export class MyModule { }
```

```ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { MyModule } from './mycomponent/my-module';

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

Moduły opis


Moduł w Angularze
AppComponent
ngOnInit(): void
zawiera
ModuleB
declarations: any[AppComponent]
imports: any[ModuleA]
exports:[],
import
ModuleA
declarations: any[Component1, Component2]
imports: any[]
exports:any[Component1, Component2]
zawiera
Component2
ngOnInit(): void
używa
zawiera
Component1
ngOnInit(): void

Biblioteki

Biblioteki

Co to są biblioteki?

Zewnętrzne pakiety, które można zainstalować w aplikacji Angularowej. Udostępniają gotowe rozwiązania, takie jak komponenty, dyrektywy, usługi i wiele więcej

Jak korzystać z bibliotek w Angularze?

Krok 1: Zainstaluj bibliotekę za pomocą CLI (Command Line Interface) poleceniem

np. `npm install --save my-library`

Krok 2: Zaimportuj potrzebne elementy z biblioteki w pliku komponentu lub innym pliku

np. `import { MyComponent } from 'my-library'`

Popularne biblioteki Angulara

Bootstrap

Biblioteka CSS, która zapewnia gotowe style i komponenty interfejsu użytkownika

Material

Biblioteka CSS, która zapewnia gotowe style i komponenty interfejsu użytkownika w stylu Material Design.

RxJS

Biblioteka, która zapewnia narzędzia do programowania reaktywnego

NgRx

Biblioteka, która zapewnia narzędzia do zarządzania stanem aplikacji

Component interaction

02

String Intelpolaration

Sposoby interakcji między komponentami

String interpolation:

- Służy do wyświetlania danych w szablonie
- Umożliwia wyświetlanie danych z komponentów w szablonie za pomocą klamerek {{ }}

```
<!-- Szablon komponentu -->
<div class="container-fluid bg-dark text-light py-4">
    <div class="card" style="width: 18rem;">
    <img src="{{itemImageUrl}}" class="card-img-top" alt="card">
    <div class="card-body">
      <h5 class="card-title">{{title}}</h5>
      <p class="card-text">RICK</p>
    </div>
  </div>
</div>

<!-- Komponent TS -->
export class HelloComponent {
  title = 'Rick roll';
  itemImageUrl=
'https://media.tenor.com/x8v1oNUOmg4AAAAM/rickroll-roll.gif';
}
```

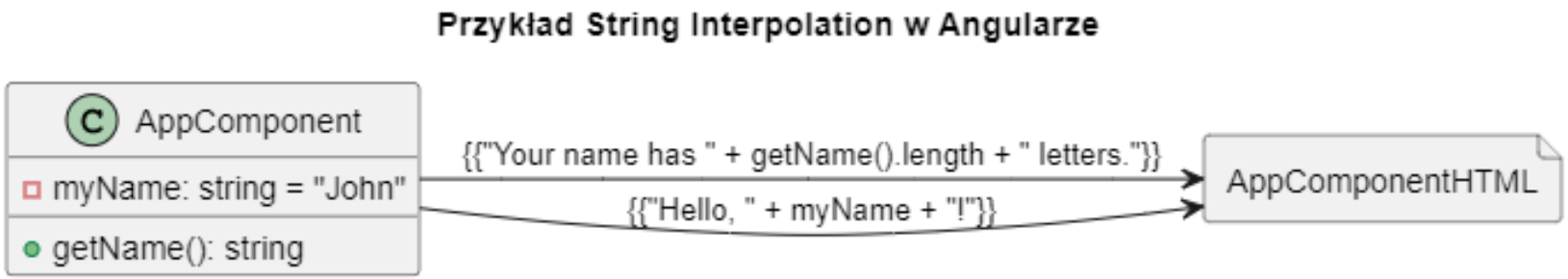
Rick roll

RICK

Elementy | Konsola | Źródła | Sieć | Wydajność | Pamięć | Aplikacja | Zabezpieczenia | Lighthouse

```
<!DOCTYPE html>
<html lang="en" class=" bcxijuqhu idc0_347">
▶ <head> … </head>
▼ <body>
  ▼ <app-root _nghost-kho-c24 ng-version="15.2.9">
    ▼ <app-mycomponent _ngcontent-kho-c24 _nghost-kho-c23>
      ▼ <div _ngcontent-kho-c23 class="container-fluid bg-dark text-light py-4">
        ▼ <div _ngcontent-kho-c23 class="card" style="width: 18rem;"> flex
            <img _ngcontent-kho-c23 alt="card" class="card-img-top" src="https://media.tenor.com/x8v1oNUOmg4AAAAM/rickroll-roll.gif">
          ▼ <div _ngcontent-kho-c23 class="card-body"> == $0
              <h5 _ngcontent-kho-c23 class="card-title">Rick roll</h5>
              <p _ngcontent-kho-c23 class="card-text">RICK</p>
            </div>
          </div>
        </div>
      </app-mycomponent>
    </app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```

Sposoby interakcji między komponentami



Property **binding**

Sposoby interakcji między komponentami

Property binding:

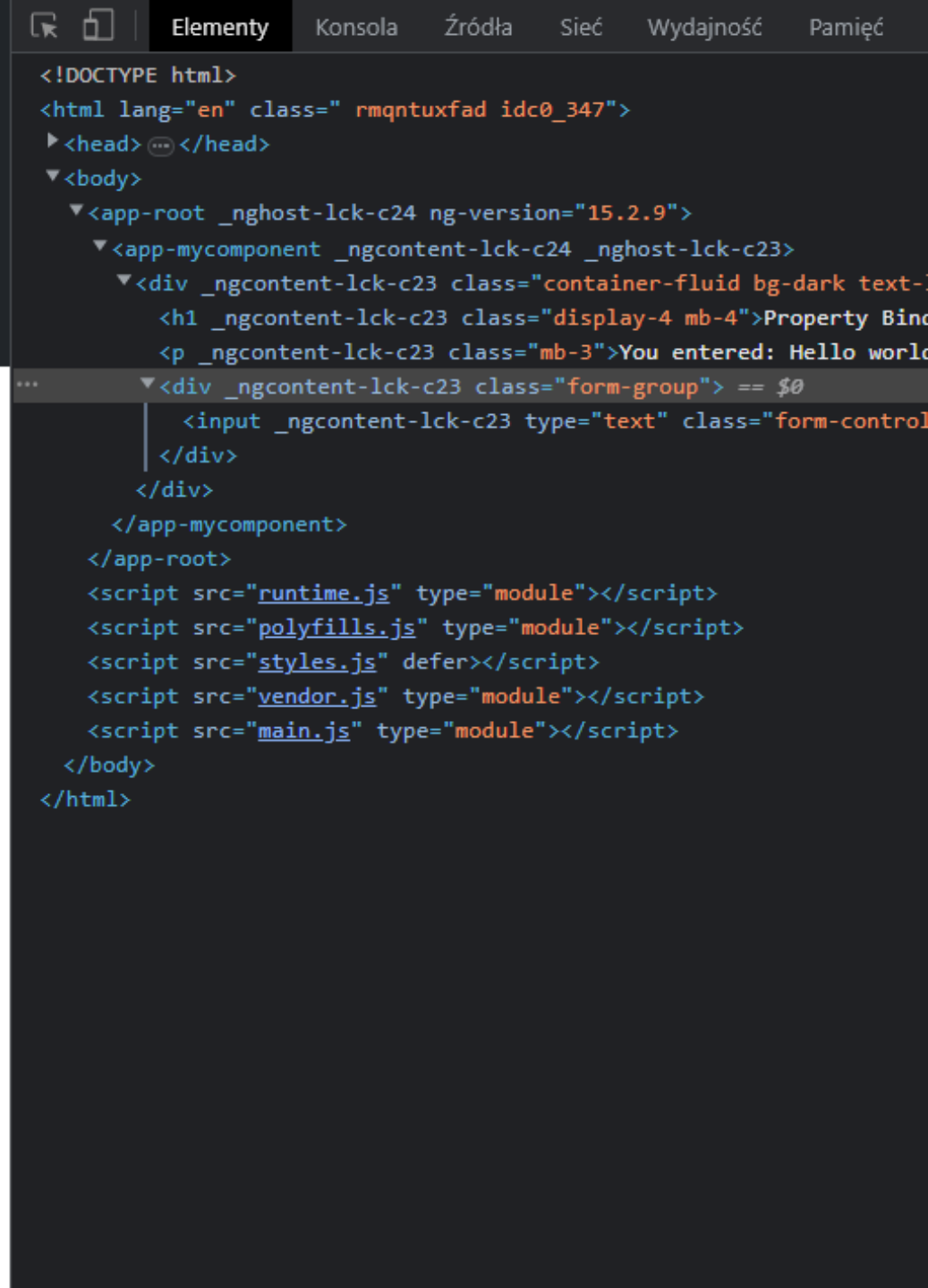
- Służy do przekazywania danych z jednego komponentu do drugiego
- Umożliwia przypisanie wartości do atrybutu HTML z komponentu

```
<!-- Szablon komponentu, ustawienie wartości-->
<div class="container-fluid bg-dark text-light py-4">
    <h1 class="display-4 mb-4">Property Binding Example</h1>
    <p class="mb-3">{{message}}</p>
    <div class="form-group">
      <input type="text" class="form-control" [value]="inputValue"
(input)="onInputChange($event)">
    </div>
</div>

<!-- Komponent TS, ustawienie wartości -->
export class HelloComponent {
  message: string = 'Hello, Angular!';
  inputValue: string = '';

  onInputChange(event: any) {
    this.inputValue = event.target.value;
    this.message = 'You entered: ' + event.target.value;
  }
}
```

Property Binding Example

You entered: Hello world

Hello world



Sposoby interakcji między komponentami

Event **binding**

Sposoby interakcji między komponentami

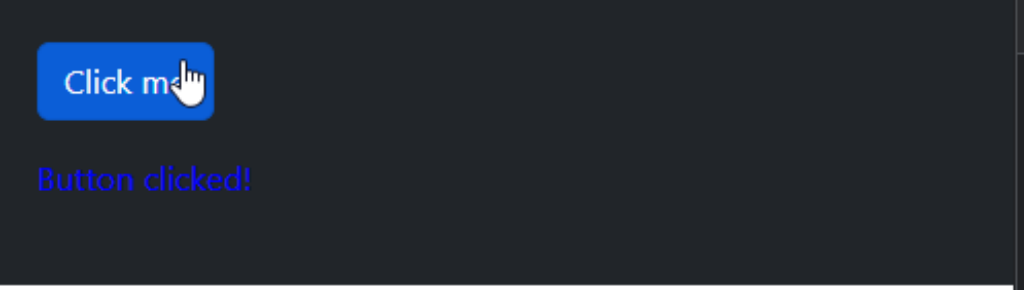
Event binding:

- Służy do obsługi zdarzeń w Angularze
- Umożliwia reagowanie na akcje użytkownika, takie jak kliknięcie przycisku

```
<!-- Szablon komponentu, przypisanie funkcji do kliknięcia przycisku -->
<div class="container-fluid bg-dark text-light py-4">
    <div class="container">
        <button class="btn btn-primary" (click)="onButtonClick()">Click me</button>
        <p class="mt-3" [ngStyle]="{ 'color': textColor }">{{message}}</p>
      </div>
  </div>


<!-- Komponent TS, zdefiniowanie funkcji onButtonClick() -->
export class ButtonComponent {
  message: string = 'Hello, Angular!';
  textColor: string = 'white';

  onButtonClick() {
    this.message = 'Button clicked!';
    this.textColor = 'blue';
  }
}
```

Elementy Konsola Źródła Sieć Wydajność Pamięć Aplikacja Zabezpieczenia Lighthou

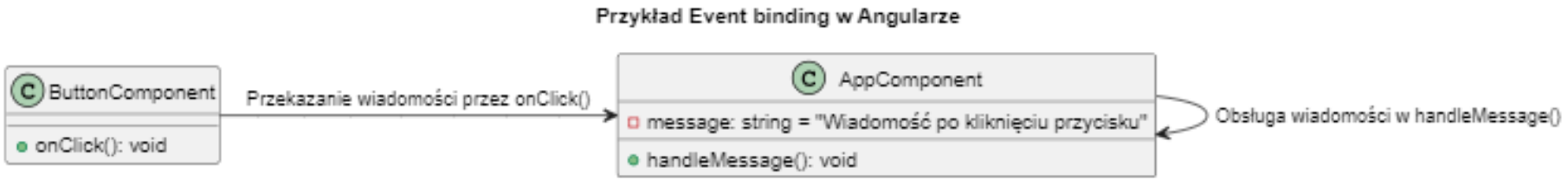
```
<!DOCTYPE html>
<html lang="en" class=" tanll idc0_347">
▶ <head> … </head>
▼ <body>
  ▼ <app-root _nghost-pbk-c24 ng-version="15.2.9">
    ▼ <app-mycomponent _ngcontent-pbk-c24 _nghost-pbk-c23>
      ▼ <div _ngcontent-pbk-c23 class="container-fluid bg-dark text-light py-4">
        ▼ <div _ngcontent-pbk-c23 class="container"> == $0
            <button _ngcontent-pbk-c23 class="btn btn-primary">Click me</button>
            <p _ngcontent-pbk-c23 class="mt-3" ng-reflect-ng-style="[object Object]" style="color: white
          </div>
        </div>
      </app-mycomponent>
    </app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```



Elementy Konsola Źródła Sieć Wydajność Pamięć Aplikacja Zabezpieczenia Lighth

```
<!DOCTYPE html>
<html lang="en" class=" klxuxrgpf idc0_347">
▶<head> … </head>
▼<body>
  ▼<app-root _nghost-era-c24 ng-version="15.2.9">
    ▼<app-mycomponent _ngcontent-era-c24 _nghost-era-c23>
      ▼<div _ngcontent-era-c23 class="container-fluid bg-dark text-light py-4">
        ▼<div _ngcontent-era-c23 class="container"> == $0
            <button _ngcontent-era-c23 class="btn btn-primary">Click me</button>
            <p _ngcontent-era-c23 class="mt-3" ng-reflect-ng-style="[object Object]" style="color: blu
          </div>
        </div>
      </app-mycomponent>
    </app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```

Sposoby interakcji między komponentami

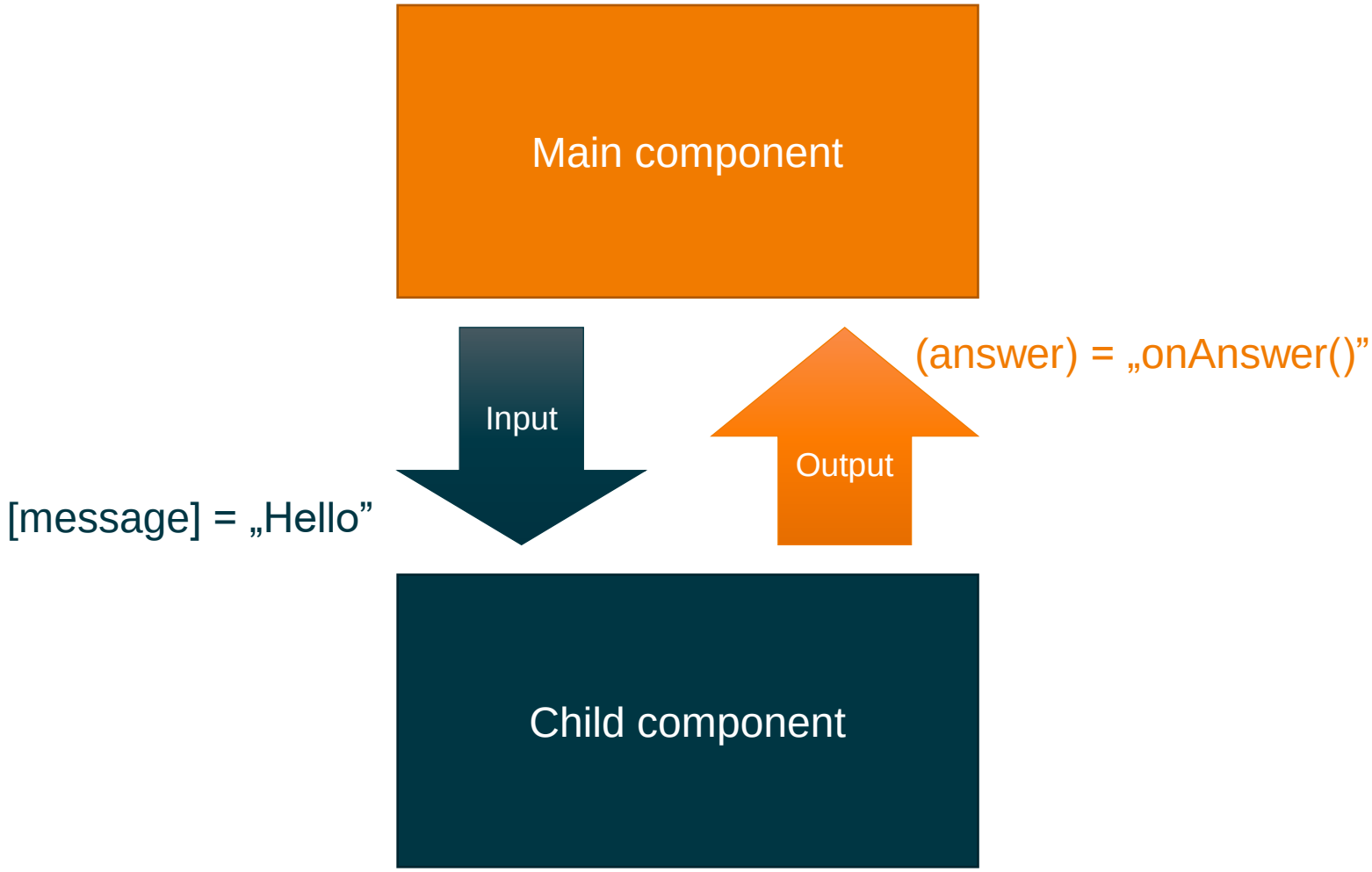


Inputs and Outputs

Sposoby interakcji między komponentami

Inputs and Outputs:

- Służą do przekazywania danych między komponentami
- Inputs umożliwiają przekazywanie danych od rodzica do dziecka
- Outputs umożliwiają przekazywanie danych z dziecka do rodzica



Main component

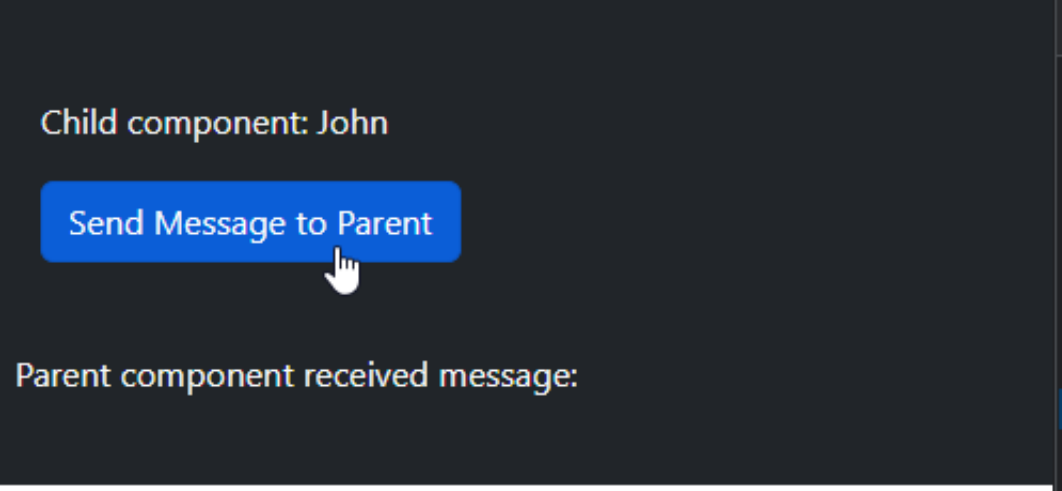
```
<!-- Szablon komponentu-->
<div class="container-fluid bg-dark text-light py-4">
    <app-my-modal [name]="'John'" (messageEvent)="receiveMessage($event)">
    </app-my-modal>
    <p class="mt-3">Parent component received message:{{messageFromChild}}</p>
</div>
```

```
<!-- Komponent TS-->
export class MainComponent {
  messageFromChild: string = '';

  receiveMessage(message: string) {
    this.messageFromChild = message;
  }
}
```

Child component

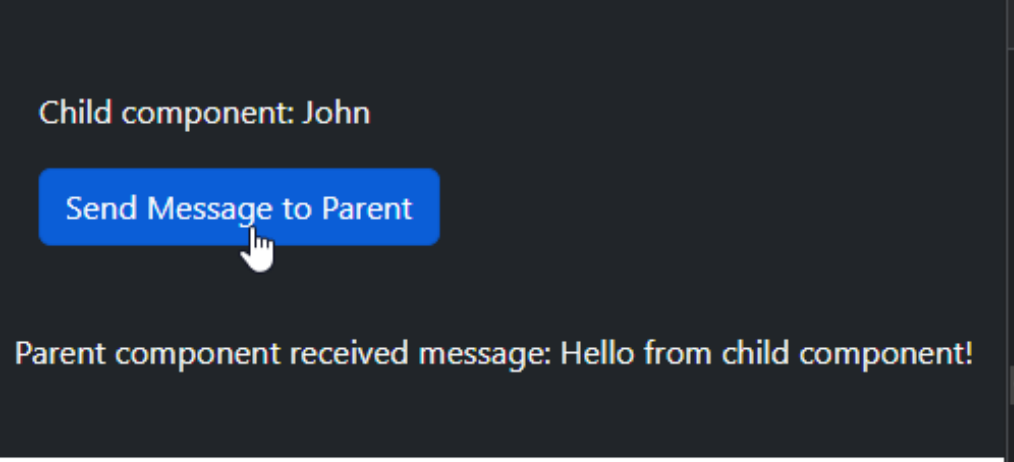
```
<!-- Szablon komponentu-->
<div class="container-fluid bg-dark text-light py-4">
    <p class="mb-3">Child component: {{name}}</p>
    <button class="btn btn-primary" (click)="sendMessage()">
    Send Message to Parent</button>
</div>
```

```
<!-- Komponent TS-->
export class ChildComponent {
  @Input() name: string | undefined;
  @Output() messageEvent = new EventEmitter<string>();

  sendMessage(){
    this.messageEvent.emit("Hello from child");
  }

}
```



Elementy Konsola Źródła Sieć Wydajność Pamięć Aplikacja Zabezpieczenia

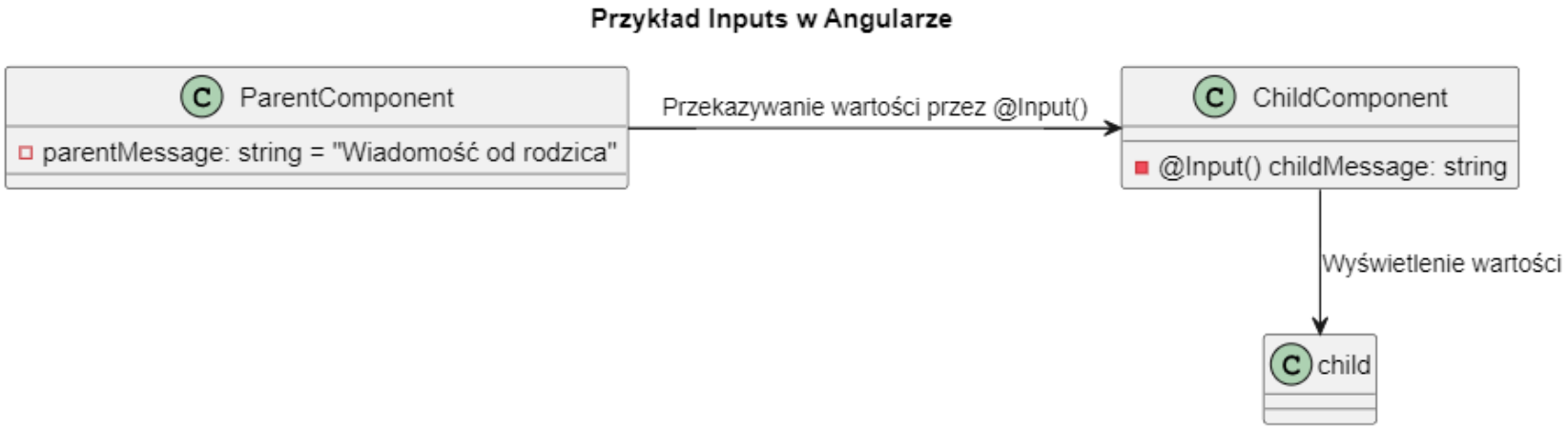
```
<!DOCTYPE html>
<html lang="en" class=" jztcawmx idc0_347">
▶<head>…</head>
▼<body>
  ▼<app-root _nghost-kna-c25 ng-version="15.2.9">
    ▼<app-mycomponent _ngcontent-kna-c25 _nghost-kna-c24>
      ▼<div _ngcontent-kna-c24 class="container-fluid bg-dark text-light py-4">
        ▼<app-my-modal _ngcontent-kna-c24 _nghost-kna-c23 ng-reflect-name="John">
          ▼<div _ngcontent-kna-c23 class="container-fluid bg-dark text-light py-4"> == $0
              <p _ngcontent-kna-c23 class="mb-3">Child component: John</p>
              <button _ngcontent-kna-c23 class="btn btn-primary">Send Message to Parent</button>
            </div>
          </app-my-modal>
          <p _ngcontent-kna-c24 class="mt-3">Parent component received message: </p>
        </div>
      </app-mycomponent>
    </app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```



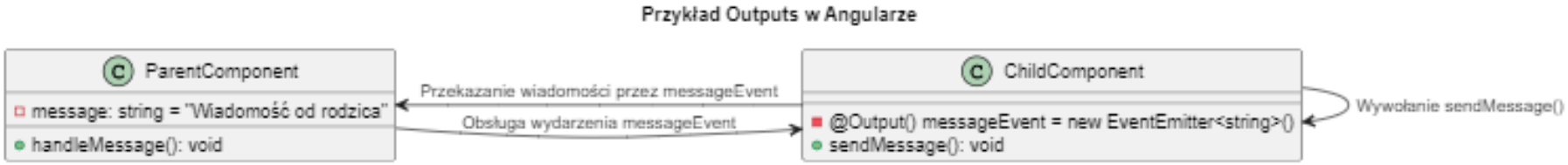
Elementy Konsola Źródła Sieć Wydajność Pamięć Aplikacja Zabezpieczenia Lighthou

```
<!DOCTYPE html>
<html lang="en" class=" jztcawmx idc0_347">
▶ <head> … </head>
▼ <body>
  ▼ <app-root _nghost-kna-c25 ng-version="15.2.9">
    ▼ <app-mycomponent _ngcontent-kna-c25 _nghost-kna-c24>
      ▼ <div _ngcontent-kna-c24 class="container-fluid bg-dark text-light py-4">
        ▼ <app-my-modal _ngcontent-kna-c24 _nghost-kna-c23 ng-reflect-name="John">
          ▼ <div _ngcontent-kna-c23 class="container-fluid bg-dark text-light py-4"> == $0
              <p _ngcontent-kna-c23 class="mb-3">Child component: John</p>
              <button _ngcontent-kna-c23 class="btn btn-primary">Send Message to Parent</button>
            </div>
          </app-my-modal>
          <p _ngcontent-kna-c24 class="mt-3">Parent component received message: Hello from child compone
        </div>
      </app-mycomponent>
    </app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```

Sposoby interakcji między komponentami



Sposoby interakcji między komponentami



Structural **directives**

Sposoby interakcji między komponentami

Structural directives:

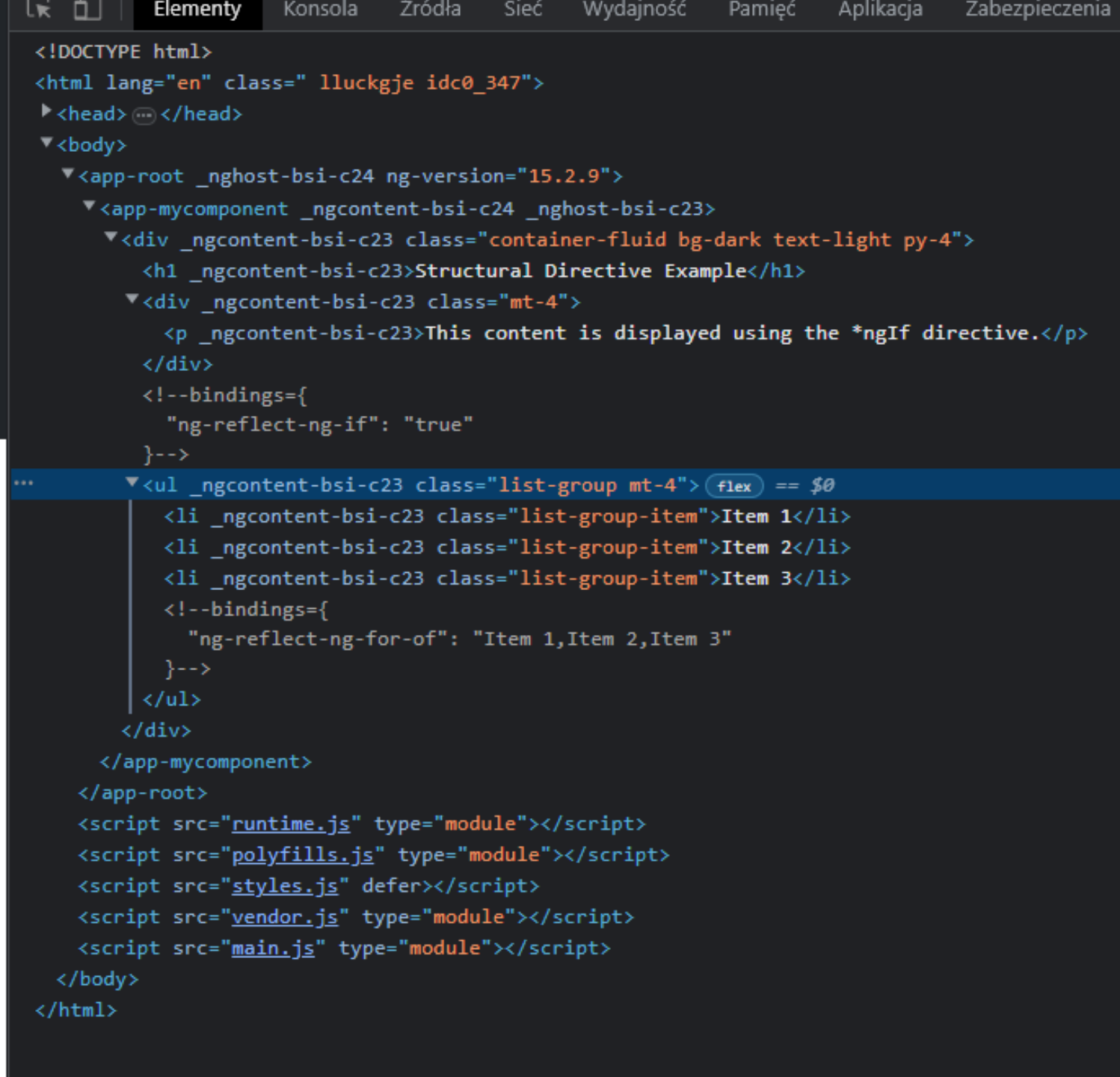
- Służą do manipulowania strukturą szablonu
- Umożliwiają dodawanie i usuwanie elementów HTML w zależności od wartości wyrażenia logicznego

```
<!-- Szablon komponentu, użycie dyrektywy *ngfor do wypisywania wartości tablicy -->

<div class="container-fluid bg-dark text-light py-4">
    <h1>Structural Directive Example</h1>

    <div *ngIf="showContent" class="mt-4">
      <p>This content is displayed using the *ngIf directive.</p>
    </div>

    <ul class="list-group mt-4">
      <li *ngFor="let item of items" class="list-group-item">{{item}}</li>
    </ul>
</div>


<!-- Komponent TS, ustawienie z tablicą items-->

export class DirectiveComponent {
  showContent: boolean = true;
  items: string[] = ['Item 1', 'Item 2', 'Item 3'];

  toggleContent() {
    this.showContent = !this.showContent;
  }
}
```

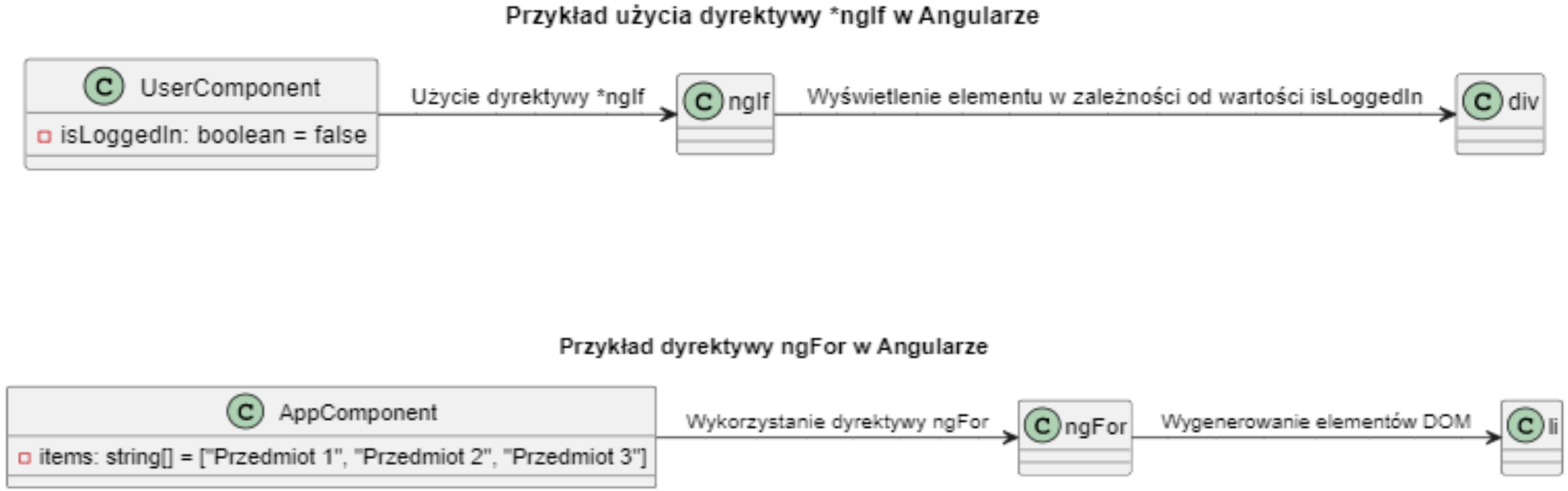
Structural Directive Example

This content is displayed using the *ngIf directive.

- Item 1
- Item 2
- Item 3



Sposoby interakcji między komponentami

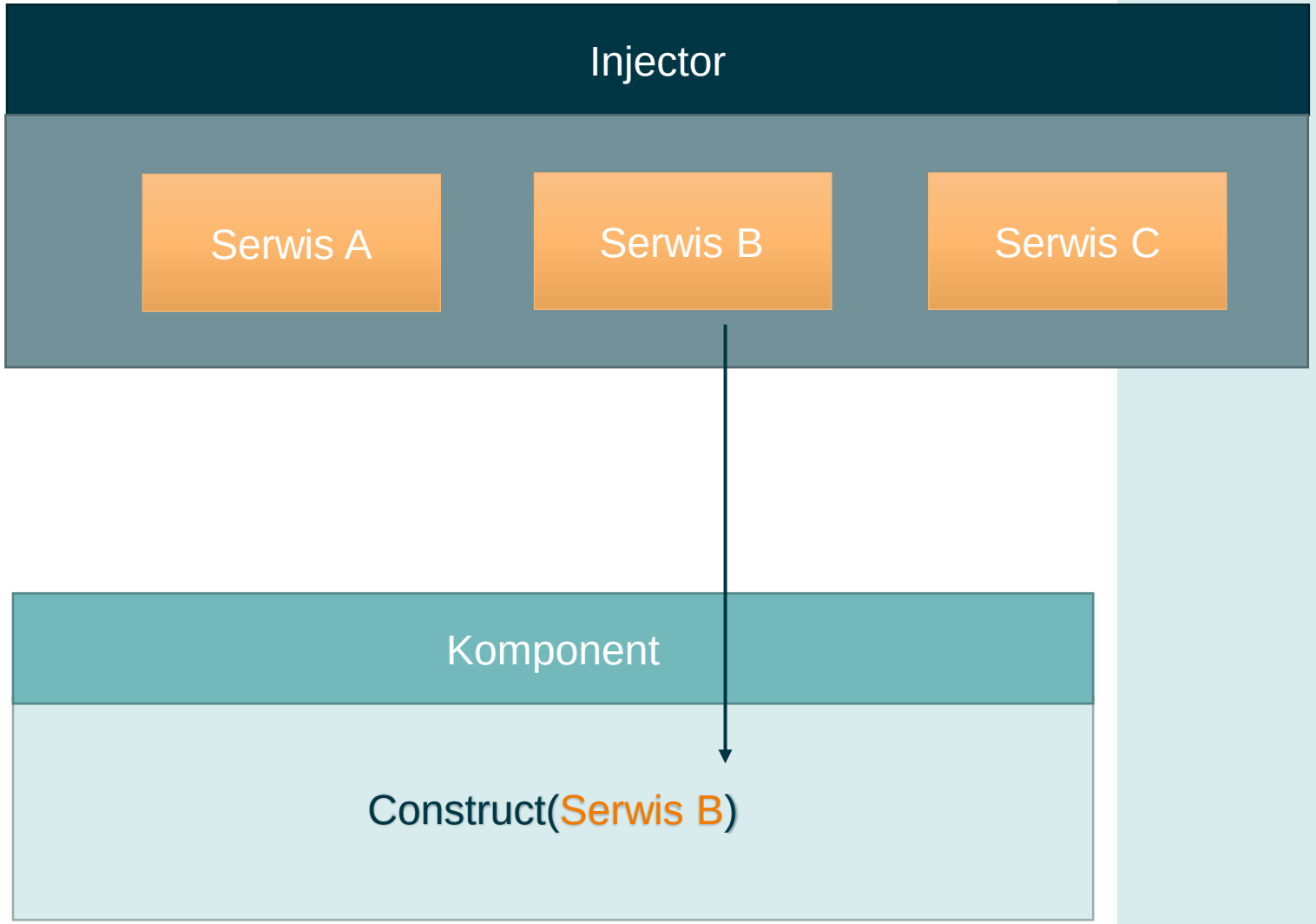


Dependency Injection i routing

03

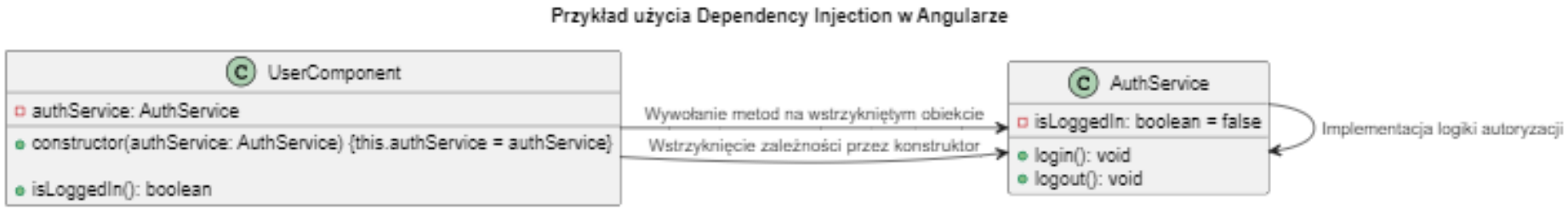
convista:

Dependency Injection w Angular



- Dependency Injection to wzorzec projektowy, który pozwala na przeniesienie odpowiedzialności za tworzenie i dostarczanie obiektów do innej klasy
- W Angularze Dependency Injection jest zaimplementowany w taki sposób, że komponenty dostają swoje zależności od zewnętrznego dostawcy
- W Angularze zależności definiuje się za pomocą konstruktorów
- Angular dostarcza kontener wstrzykiwania zależności (Dependency Injection Container), który umożliwia definiowanie i dostarczanie zależności
- Kontener jest dostępny w całej aplikacji, co umożliwia łatwe i skuteczne zarządzanie zależnościami

Dependency Injection w Angular



Dependency Injection w Angular

```ts
import { Component, Inject } from '@angular/core';
import { SomeService } from './some.service';

@Component({
  selector: 'app-example',
  template: `
    <h1>Example Component</h1>
    <p>{{ message }}</p>
  `,
})
export class ExampleComponent {
  constructor(@Inject(SomeService) private service: SomeService)
{}

  message = this.service.getMessage();
}
```

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

@Injectable({
  providedIn: 'root'
})

export class SomeService {
  getMessage(): string {
    return 'Hello from SomeService';
  }
}
```

Routing

Routing w Angular

Routing to mechanizm dzięki której możemy nawigować między komponentami w aplikacji jednostronicowej.

Pozwala na dynamiczną zmianę zawartości strony bez przeładowywania całej aplikacji.

W Angularze, routing jest obsługiwany przez moduł RouterModule, który musi być zaimportowany i zainicjalizowany w aplikacji.

Moduł ten definiuje zestaw tras (routes), które są mapowane na konkretne komponenty

1. Definiowanie trasy w odpowiednim module

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

2. Dodanie dyrektywę <router-outlet> do szablonu głównego komponentu..

```
<router-outlet></router-outlet>
```

3. Wstrzyknięcie RouterModule do tablicy imports w głównym module aplikacji.

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

4. Definiowanie trasy w odpowiednim module

```
<a routerLink="/">Strona główna</a>
<a routerLink="/about">O nas</a>
<a routerLink="/contact">Kontakt</a>
```

```
import { Router } from '@angular/router';

constructor(private router: Router) { }

navigateToAbout() {
  this.router.navigate(['/about']);
}
```

Project Introduction

04

convista:

Stackblitz:

https://stackblitz.com/github/pssyvende/angular_crash_course_competition

Email:

marek.cichon@convista.com

convista: