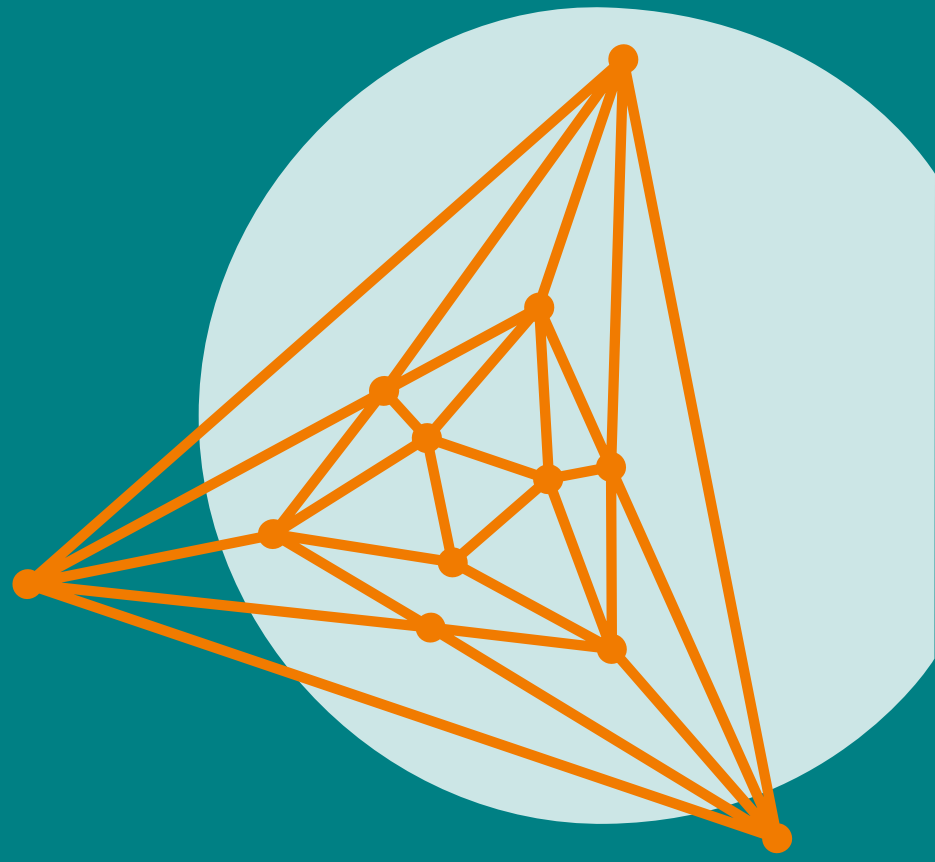

convista:

convista:

Wprowadzenie do Angulara

Komunikacja międzykomponentowa



Opole | 13.11.2023

Agenda

01 **Podstawowe pojęcia związane z Angularem**

- Elementy składowe aplikacji Angularowej
- Budowa komponentu
- Budowa modułu

02 **Składnia szablonów**

- Interpolacja tekstu
- Wiązanie właściwości
- Wiązanie zdarzeń
- Wbudowane dyrektywy strukturalne

03 **Interakcja komponentów**

- Przekazywanie danych od rodzica do dziecka
- Nasłuchiwanie zdarzeń emitowanych przez dziecko
- Komunikacja komponentów za pomocą serwisu

04 **Projekt ćwiczeniowy**

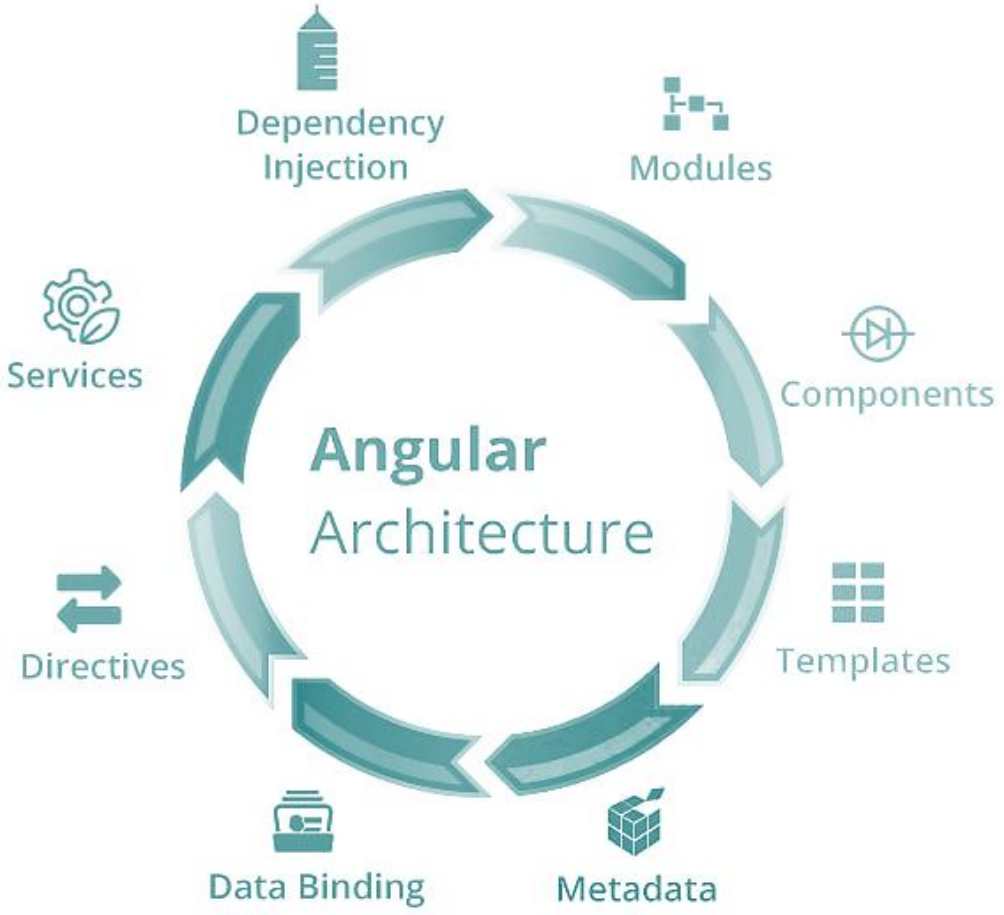
Podstawowe pojęcia związane z Angularem

01

convista:

Elementy składowe aplikacji Angularowej

https://www.besanttechnologies.com/what-is-angular



Budowa komponentu

Selektor

Selektor CSS, który jednoznacznie identyfikuje komponent w kodzie HTML

https://angular.io/api/core/Directive#selector

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

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

@Component({
    selector: 'app-root’,
    templateUrl: './app.component.html’,
    styleUrls: ['./app.component.css'] })
export class AppComponent {
    // component’s behaviour
}
```

Budowa komponentu

app.component.html

```
<!-- before Angular16 -->
<app-root></app-root>

<!-- since Angular16 -->
<app-root/>
```

Budowa modułu

app.module.ts

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

Budowa modułu

Właściwość	Szczegóły
declarations	Deklaracja komponentów, dyrektyw i potoków, które należą do tego modułu
exports	Podzbiór deklaracji, które powinny być widoczne w szablonach komponentów innych modułów
imports	Moduły, których klasy wykorzystywane są w tym module
providers	Serwisy, które mają być dostępne we wszystkich częściach aplikacji
bootstrap	Deklaracja komponentu głównego, który obsługuje wszystkie inne widoki aplikacji

Składnia szablonów

02

convista:

Interpolacja tekstu

- Interpolacja łańcuchów tekstowych (ang. interpolation) jest przykładem jednokierunkowego wiązania danych
- Używana jest do łączenia danych z kodu TypeScript z szablonem HTML
- Wiązanie polega na użyciu **podwójnych nawiasów klamrowych**

app.component.ts

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

@Component({
    selector: 'app-root’,
    templateUrl: './app.component.html’,
    styleUrls: ['./app.component.css'] })
export class AppComponent {
    welcomeMessage = 'Witaj w Angular!’;
    imageUrl = 'https://example.com/image.jpg';
}
```

Interpolacja tekstu

- Interpolacja łańcuchów tekstowych (ang. interpolation) jest przykładem jednokierunkowego wiązania danych
- Używana jest do łączenia danych z kodu TypeScript z szablonem HTML
- Wiązanie polega na użyciu **podwójnych nawiasów klamrowych**

app.component.html

```
<div>
    <h1>{{ welcomeMessage }}</h1>
    <img src=„{{imageUrl}}" alt=„Image">
</div>
```

Wiązanie właściwości

- Wiązanie właściwości (ang. property binding) jest przykładem jednokierunkowego wiązania danych
- Podczas takiego wiązania łączona jest właściwość elementu DOM z właściwością zdefiniowaną w kodzie TypeScript komponentu
- Aby powiązać właściwość elementu DOM, należy ująć ją w **nawiasy kwadratowe**

app.component.ts

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

@Component({
    selector: 'app-root’,
    templateUrl: './app.component.html’,
    styleUrls: ['./app.component.css'] })
export class AppComponent {
    imageUrl = 'https://example.com/image.jpg';
}
```

Wiązanie właściwości

- Wiązanie właściwości (ang. property binding) jest przykładem jednokierunkowego wiązania danych
- Podczas takiego wiązania łączona jest właściwość elementu DOM z właściwością zdefiniowaną w kodzie TypeScript komponentu
- Aby powiązać właściwość elementu DOM, należy ująć ją w **nawiasy kwadratowe**

app.component.html

```
<div>
    <img [src]="imageUrl" alt=„Image">
</div>
```

Wiązanie właściwości

- Nawiasy kwadratowe powodują, że Angular traktuje prawą stronę przypisania jako **wyrażenie dynamiczne**
- Bez nawiasów Angular traktuje prawą stronę jako literał łańcuchowy i ustawia właściwość na tę statyczną wartość

app.component.html

```
<div>
    <img [src]="imageUrl" alt=„Image">
</div>

<!-- Rezultat -->
<div>
    <img src="https://example.com/image.jpg"
         alt=„Image">
</div>
```

Wiązanie właściwości

- Nawiasy kwadratowe powodują, że Angular traktuje prawą stronę przypisania jako **wyrażenie dynamiczne**
- Bez nawiasów Angular traktuje prawą stronę jako literał łańcuchowy i ustawia właściwość na tę statyczną wartość

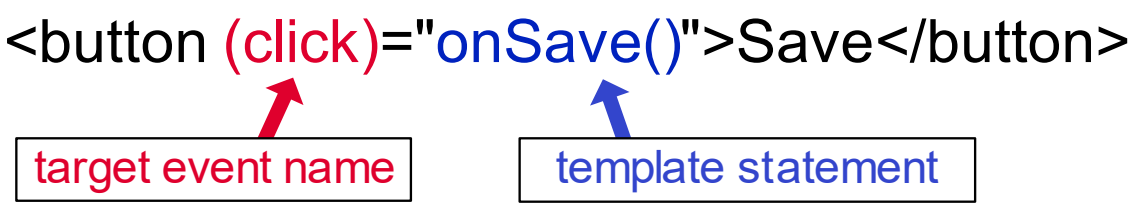
app.component.html

```
<div>
    <img src="imageUrl" alt=„Image">
</div>

<!-- Rezultat -->
<div>
    <img src=„imageUrl"
         alt=„Image">
</div>
```

Wiązanie zdarzeń

- Wiązanie zdarzeń wykorzystywane jest do obsługi zdarzeń wywoływanych bezpośrednio z DOM
- Interakcja taka powoduje wywołanie określonej metody w komponencie
- Należy powiązać nazwę zdarzenia docelowego w nawiasach po lewej stronie znaku równości z wywołaniem metody w cudzysłowie



Wiązanie zdarzeń

app.component.ts

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

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css'] })
export class AppComponent {
    welcomeMessage = 'Witaj w Angular!';

    changeMessage() {
        this.welcomeMessage = 'Zmieniony tekst';
    }
}
```

Wiązanie zdarzeń

app.component.html

```
<div>
    <h1>{{ welcomeMessage }}</h1>

    <button (click)="changeMessage()">
        Zmień tekst
    </button>
</div>
```

Wbudowane dyrektywy strukturalne

- Dyrektywy strukturalne (ang. Structural Directives) zaczynają się od **znaku specjalnego ***
- Pozwalają one na manipulowanie oraz zmianę struktury elementów drzewa DOM.

Dyrektywa strukturalna	Szczegóły
NgIf	Warunkowo tworzy lub usuwa podwidoki z szablonu
NgFor	Powtarza węzeł dla każdego elementu na liście
NgSwitch	Zestaw dyrektyw, które przełączają między alternatywnymi widokami

Wbudowane dyrektywy strukturalne

- Dyrektywy strukturalne (ang. Structural Directives) zaczynają się od **znaku specjalnego ***
- Pozwalają one na manipulowanie oraz zmianę struktury elementów drzewa DOM.

app.component.ts

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

@Component({
    selector: 'app-root’,
    templateUrl: './app.component.html’,
    styleUrls: ['./app.component.css'] })
export class AppComponent {
    showText = true;
    colors = ['Red', 'Green', 'Blue'];
}
```

Wbudowane dyrektywy strukturalne

- Dyrektywy strukturalne (ang. Structural Directives) zaczynają się od **znaku specjalnego ***
- Pozwalają one na manipulowanie oraz zmianę struktury elementów drzewa DOM.

app.component.html

```
<div>
    <p *ngIf="showText">To jest wyświetlane,
        ponieważ showText jest true.</p>

    <ul>
        <li *ngFor="let color of colors">
            {{ color }}
        </li>
    </ul>
</div>
```

Interakcja komponentów

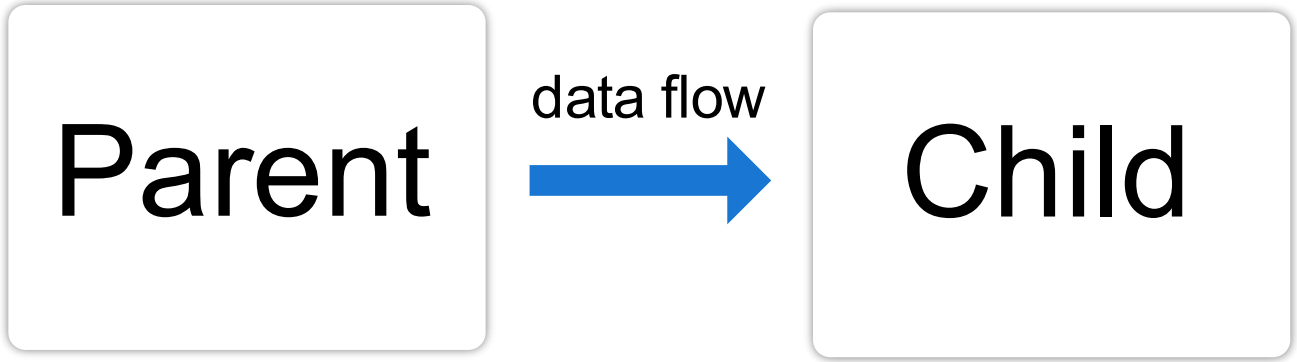
03

convista:

Przekazywanie danych od rodzica do dziecka

- Dekorator @**Input()** w komponencie podrzędnym oznacza, że właściwość może otrzymać swoją wartość od komponentu nadrzędnego
- Aby przekazać dane od rodzica do dziecka, należy skonfigurować element nadrzędny i podrzędny.

@Input



Przekazywanie danych od rodzica do dziecka

1. Aby użyć dekoratora @Input() w klasie komponentu podrzędnego, najpierw zaimportuj Input, a następnie udekoruj właściwość za pomocą @Input()

child.component.ts

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

@Component({
    selector: 'app-child’,
    template: `
        <h3>Dziecko Component</h3>
        <p>{{ message }}</p>
    `,
})
export class ChildComponent {
    @Input() message: string;
}
```

Przekazywanie danych od rodzica do dziecka

2. Następnym krokiem jest **powiązanie właściwości** w szablonie komponentu nadrzędnego

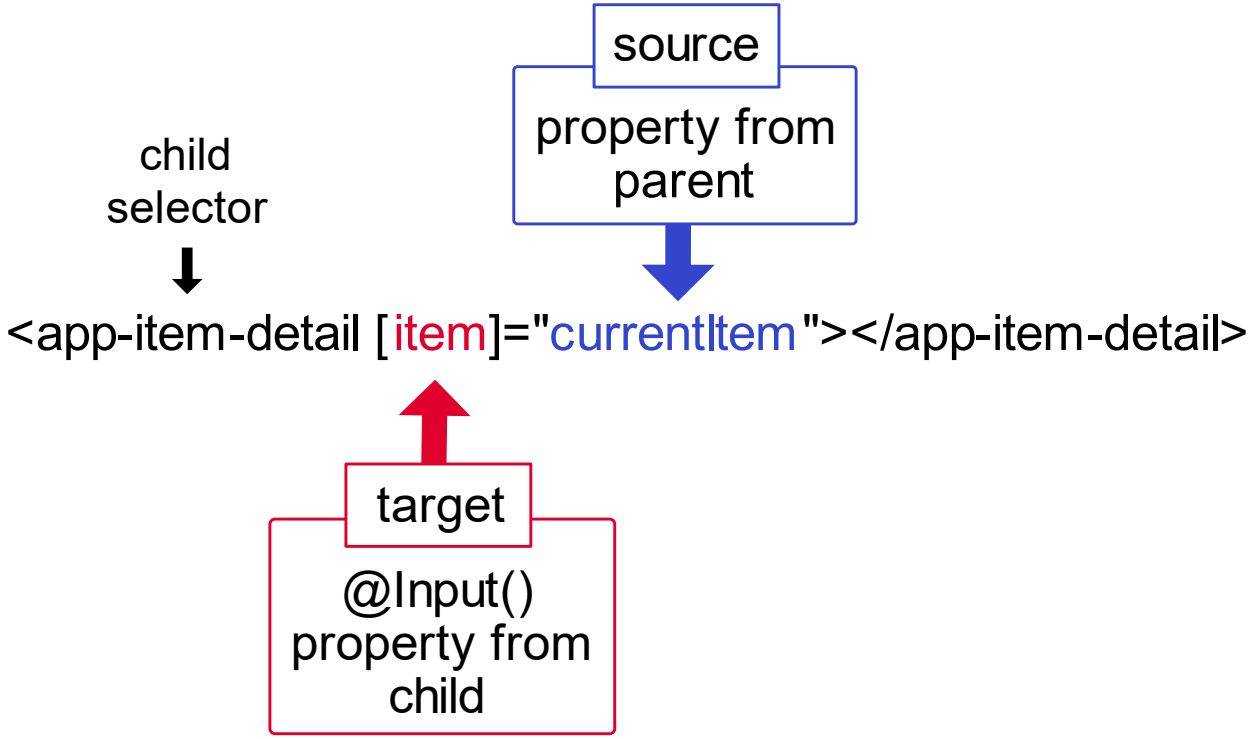
parent.component.ts

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

@Component({
    selector: 'app-parent’,
    template: `
        <h2>Rodzic Component</h2>
        <app-child [message]="parentMessage"></app-child>
    `,
})
export class ParentComponent {
    parentMessage = 'To jest wiadomość od rodzica!’;
}
```

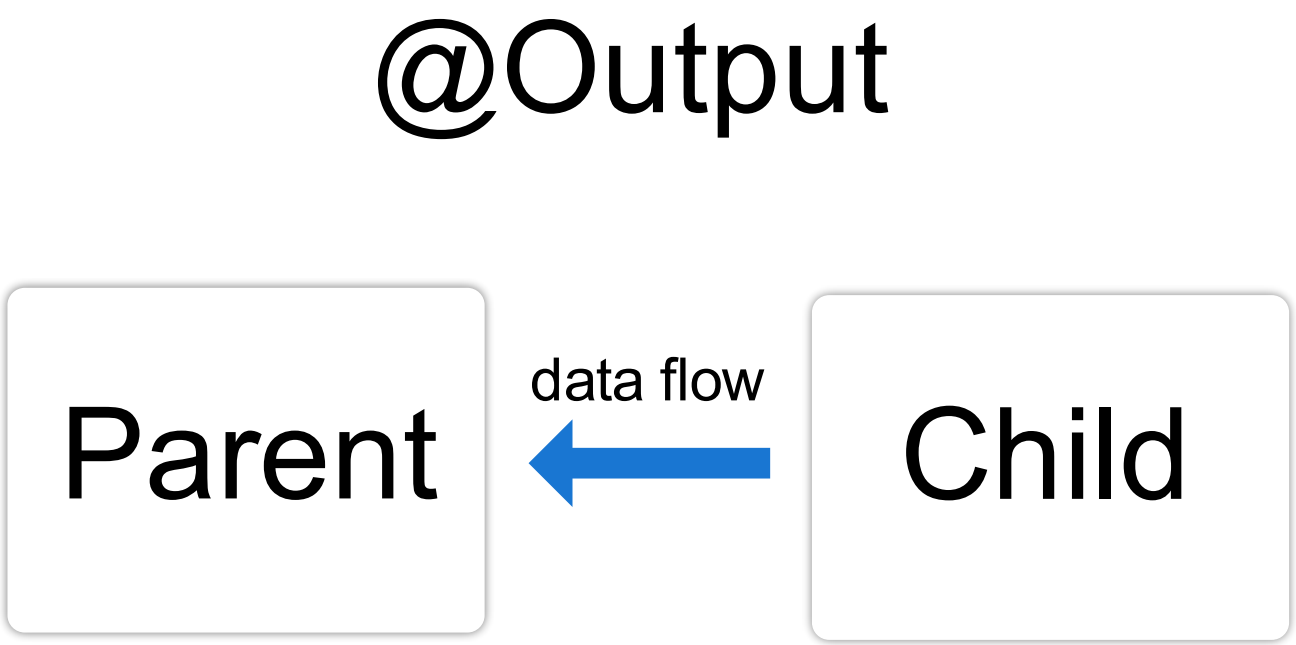
Przekazywanie danych od rodzica do dziecka

- Właściwość w nawiasach kwadratowych, to właściwość, która dekorowana jest za pomocą @Input() w komponencie podrzędnym
- Źródło wiązania, część po prawej stronie znaku równości, to dane, które komponent nadrzędny przekazuje do komponentu zagnieżdżonego



Nasłuchiwanie zdarzeń emitowanych przez dziecko

- Dekorator @**Output()** w komponencie podrzędnym pozwala na przepływ danych z komponentu podrzędnego do nadrzędnego
- Komponent podrzędny używa właściwości @Output() do **wyemitowania zdarzenia** w celu powiadomienia rodzica o zmianie



Nasłuchiwanie zdarzeń emitowanych przez dziecko

child.component.ts

```
import { Component, Output, EventEmitter } from '@angular/core’;

@Component({
    selector: 'app-child’,
    template: `
        <button (click)="sendMessageToParent()">Wyślij wiadomość
            do rodzica</button>
    `,
})
export class ChildComponent {
    @Output() messageEvent = new EventEmitter<string>();

    sendMessageToParent() {
        const message = 'To jest wiadomość od dziecka!’;
        this.messageEvent.emit(message);
    }
}
```

Nasłuchiwanie zdarzeń emitowanych przez dziecko

parent.component.ts

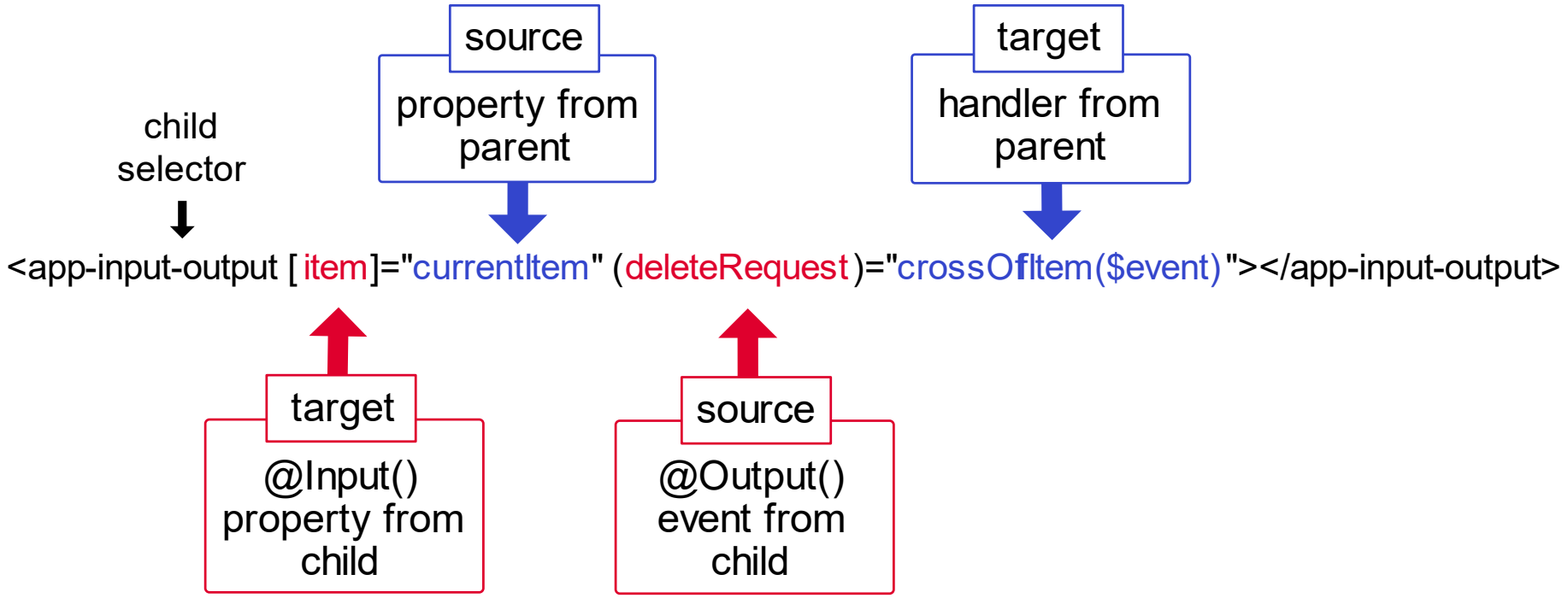
```
import { Component } from '@angular/core’;

@Component({
selector: 'app-parent’,
template: `
    <app-child (messageEvent)="receiveMessageFromChild($event)>
    </app-child>
    <p>Otrzymana wiadomość od dziecka: {{ receivedMessage }}</p>
  `,
})
export class ParentComponent {
    receivedMessage: string;

    receiveMessageFromChild(message: string) {
        this.receivedMessage = message;
    }
}
```

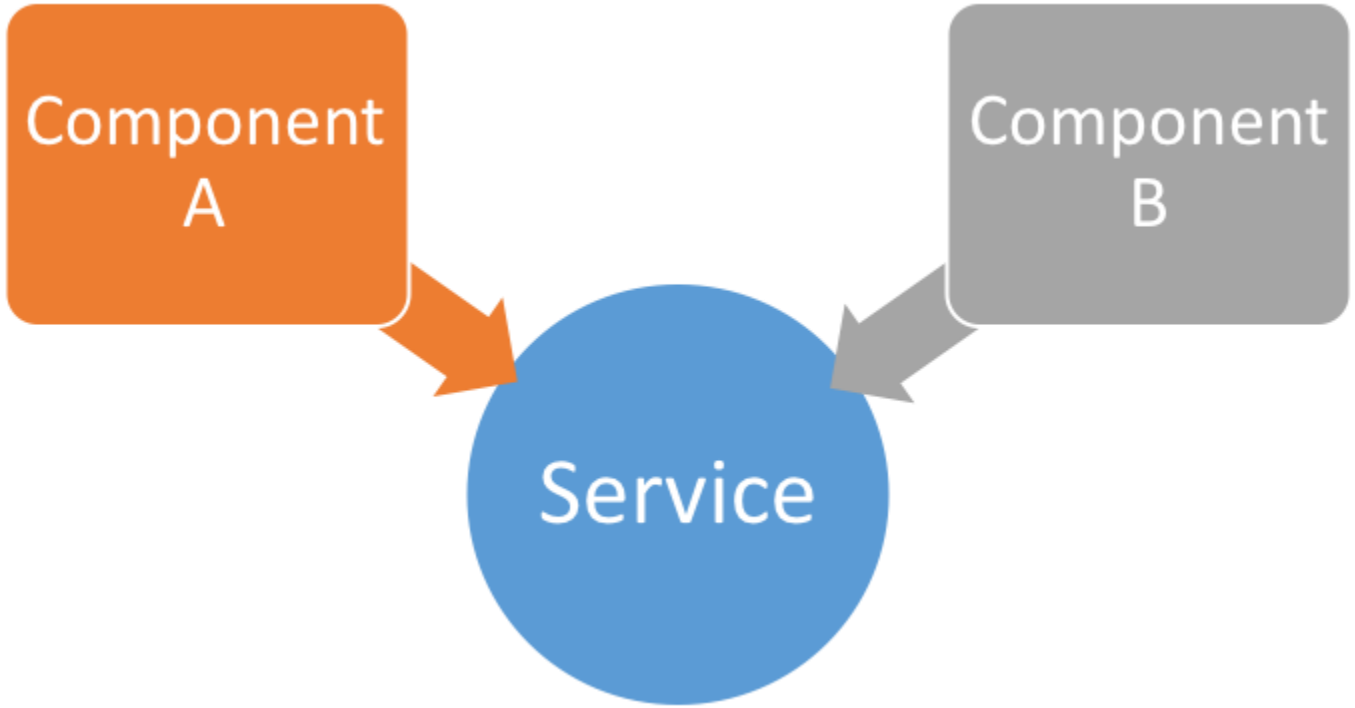
Nasłuchiwanie zdarzeń emitowanych przez dziecko

- Obiekt docelowy, który jest właściwością @Input() w klasie komponentu podrzędnego, otrzymuje swoją wartość z właściwości rodzica
- Komponent podrzędny zgłasza zdarzenie, które jest argumentem dla metody komponentu nadrzędnego



Komunikacja komponentów za pomocą serwisu

- Podczas tworzenia mniejszej części systemu, takiej jak moduł lub klasa, może być konieczne użycie funkcji z innych klas.
- **Wstrzykiwanie zależności** (ang. Dependency Injection, DI) to wzorzec projektowy i mechanizm tworzenia i dostarczania niektórych części aplikacji do innych części aplikacji, które ich wymagają.



Komunikacja komponentów za pomocą serwisu

- Dekorator **@Injectable()** oznacza klasę jako dostępną do dostarczenia i wstrzyknięcia jako zależność
- Dodatkowe użycie **providedIn: 'root'** oznacza udostępnienie serwisu na poziomie głównym aplikacji, co pozwala na wstrzykiwanie go do innych klas w aplikacji

TS data.service.ts

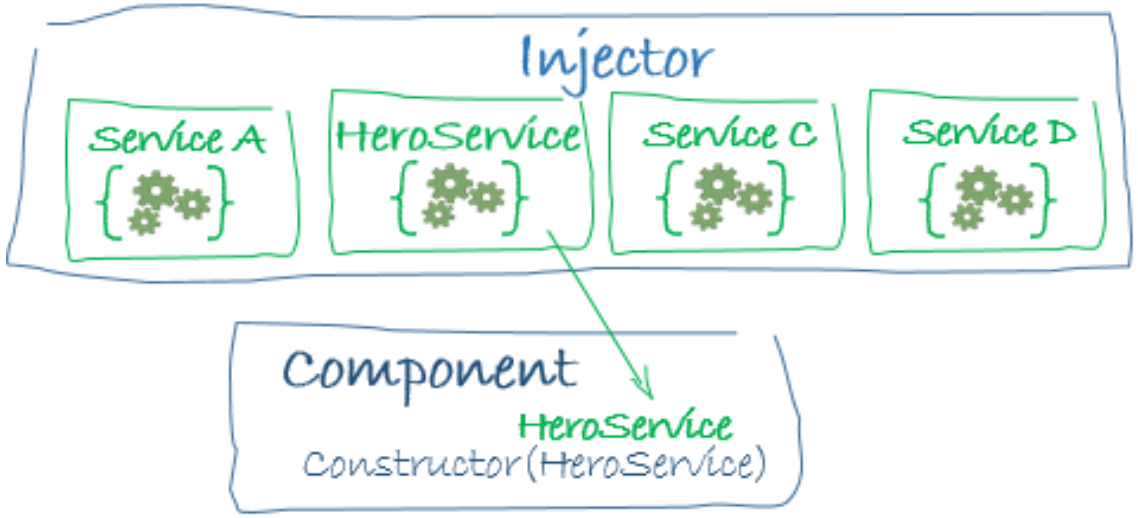
```
@Injectable({
    providedIn: ‚root’
})
export class DataService {
    private message = ’’ ;

    sendMessage(message: string) {
        this.message = message;
    }

    getMessage() {
        return this.message;
    }
}
```

Komunikacja komponentów za pomocą serwisu

- Najczęstszym sposobem wstrzykiwania zależności jest zadeklarowanie jej w konstruktorze klasy
- Gdy Angular tworzy nową instancję komponentu, określa, jakich usług lub innych zależności potrzebuje ta klasa, patrząc na typy parametrów konstruktora



Komunikacja komponentów za pomocą serwisu

first.component.ts

```
import { Component } from '@angular/core’;
import { DataService } from './data.service’;

@Component({
    selector: 'app-first’,
    template: `
    <button (click)="sendMessage()">
        Wyślij wiadomość do komponentu II
    </button> `,
})
export class FirstComponent {
    constructor(private dataService: DataService) {}

    sendMessage () {
        const message = 'To jest wiadomość od komponentu I!’;
        this.dataService.sendMessage(message);
    }
}
```

Komunikacja komponentów za pomocą serwisu

second.component.ts

```
import { Component, OnInit } from '@angular/core’;
import { DataService } from './data.service’;

@Component({
    selector: 'app-second’,
    template: `<p>
        Wiadomość od komponentu I: {{ getMessage() }}</p>`,
})

export class SecondComponent {

    constructor(private dataService: DataService) {}

    getMessage() {
        return this.dataService.getMessage();
    }
}
```

Projekt ćwiczeniowy

04

convista:

Dziękujemy
za uwagę

www.convista.pl/kariera

Michał Rusek
mrusek@convista.com

Paulina Sikacka
psikacka@convista.com

convista: