

Angular Tutorial

1. Instalacja NodeJS i Angular Cli

W celach zainstalowania NodeJS niezbędnego do tworzenia aplikacji za pomocą Angulara należy wejść na stronę <https://nodejs.org/en/>. Następnie pobrać instalator (obojętnie która wersja, ja korzystam z LTS) i go odpalić. Po zakończonej instalacji otworzyć konsolę należy wpisać następującą komendę:

```
npm install -g @angular/cli
```

Komenda ta zainstaluje przydatne narzędzie jakim jest właśnie Angular Cli. Pozwala ono m.in. na tworzenie nowych aplikacji, wstawiania nowych komponentów do istniejących aplikacji oraz uruchamiania ich. Najważniejsze komendy których będziemy używać:

`ng serve` – uruchamianie aplikacji

`ng new {{nazwa}}` – utworzenie nowej aplikacji o zadanej nazwie

`ng generate` (lub w skrócie `ng g`) `{{typ}}` `{{nazwa}}` – utworzenie nowego komponentu zadanego typu w aplikacji oraz wstawienie odpowiednich odwołań

Na chwilę obecną potrzebna nam tylko pierwsza komenda, jednakże w ramach dalszej nauki poznamy także tajniki pozostałych.

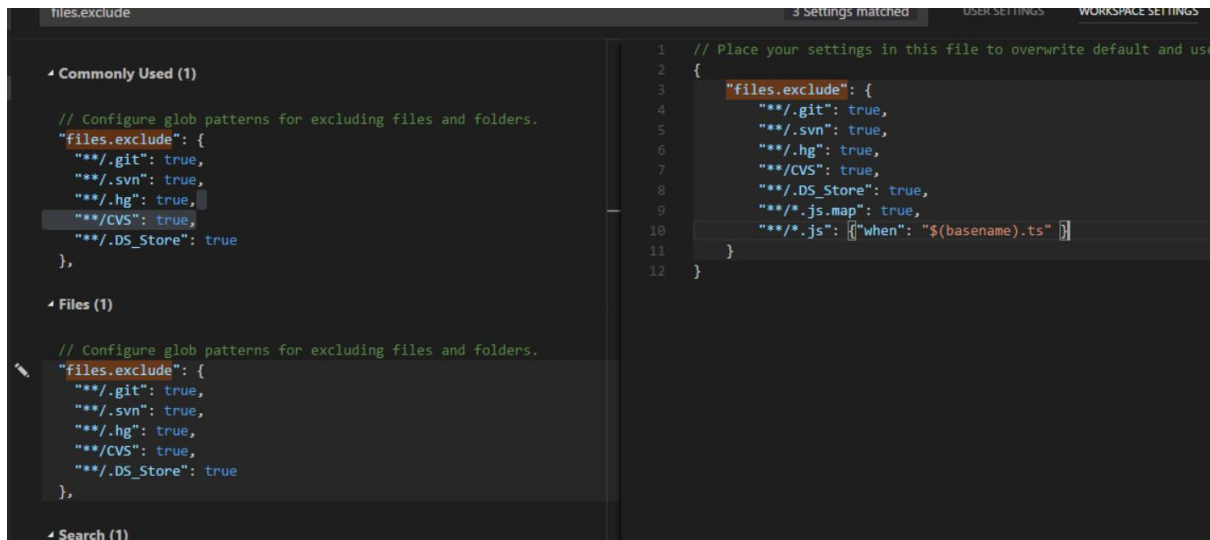
Dodatkowo przydatne komendy:

`npm install` – pobranie zależności zdefiniowanych w aplikacji

2. IDE do pisania kodu

Polecam tutaj dwie opcje: Webstorm lub Microsoft Visual Code. Code jest lżejszym narzędziem i nie tak rozbudowanym jak Webstorm ale osobiście polecam ten edytor. Posiada kilka bardzo fajnych rozwiązań, jednakże wymaga wstępnie lekkiego ulepszenia. Przede wszystkim polecam zainstalować skróty klawiszowe z ulubionego IDE, usprawnia to pracę. Osobiście korzystam ze skrótów od Intellij, wtedy importowanie, formatowanie oraz wyszukiwanie plików staje się przyjemniejsze.

Następnym krokiem jest ukrycie plików generowanych przez kompilator typescripta. Wchodzimy: file -> preferences -> settings. W wyszukiwarce wpisujemy files.exclude, klikamy na piękny ołówceczek (pojawi się po najechaniu na opcję), kopiujemy do schowka odpowiednie linie. Całość wyglądać będzie następująco:

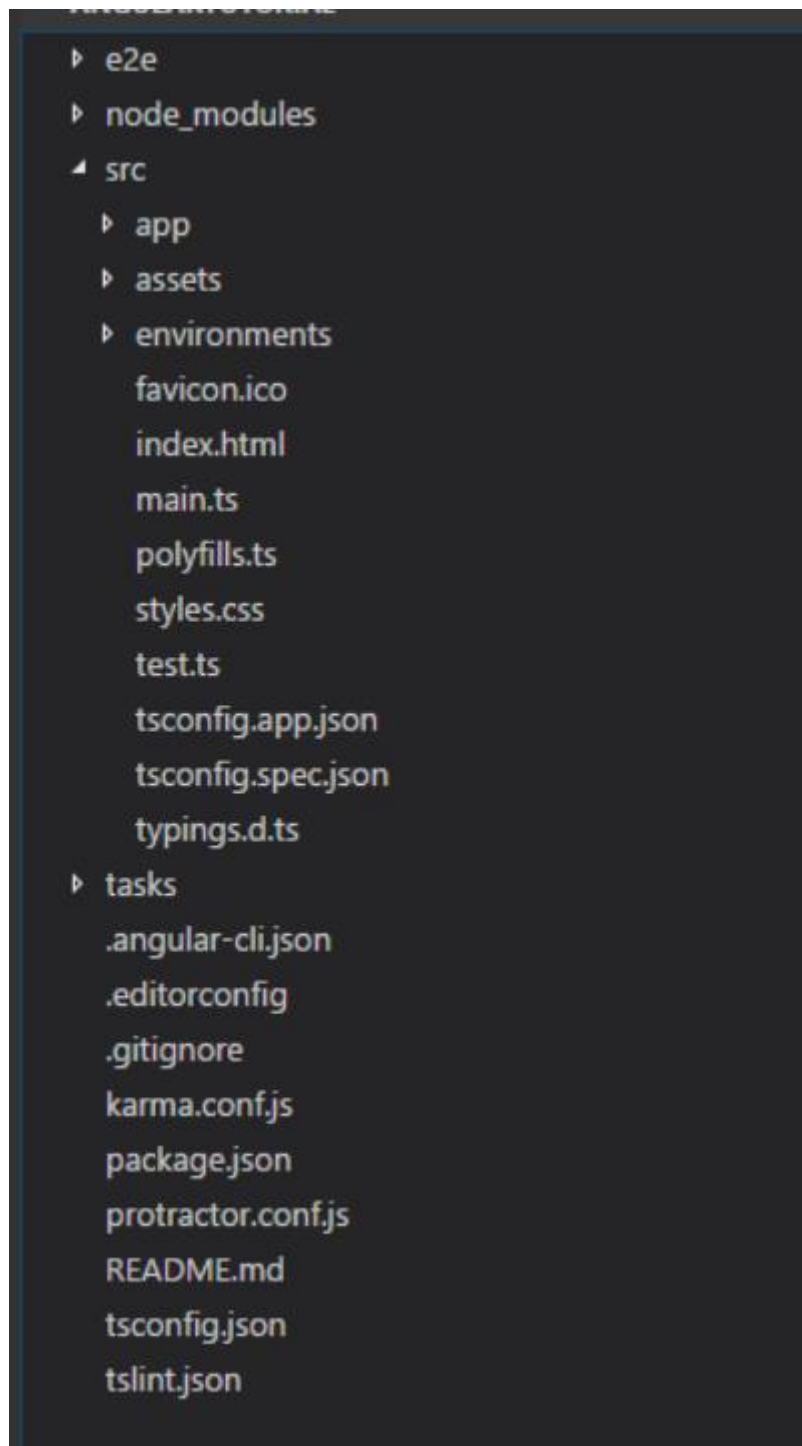
The image shows a screenshot of the Visual Studio Code interface. On the left, the 'Settings' sidebar is open, showing the 'files.exclude' setting under 'Commonly Used'. The right pane shows the 'workspaceSettings' tab with a JSON configuration for 'files.exclude'. The configuration includes patterns for .git, .svn, .hg, .CVS, .DS_Store, and *.js.map, along with a custom pattern for *.js files when the extension is .ts.

```
1 // Place your settings in this file to overwrite default and user settings
2 {
3   "files.exclude": {
4     "**/.git": true,
5     "**/.svn": true,
6     "**/.hg": true,
7     "**/CVS": true,
8     "**/.DS_Store": true,
9     "**/*.js.map": true,
10    "**/*.js": {
11      "when": "$(basename).ts"
12    }
13  }
14 }
```

Następnie jesteśmy gotowi do nauki angulara!

3. Struktura projektu

Wygenerowany przez Cli projekt ma następującą strukturę:

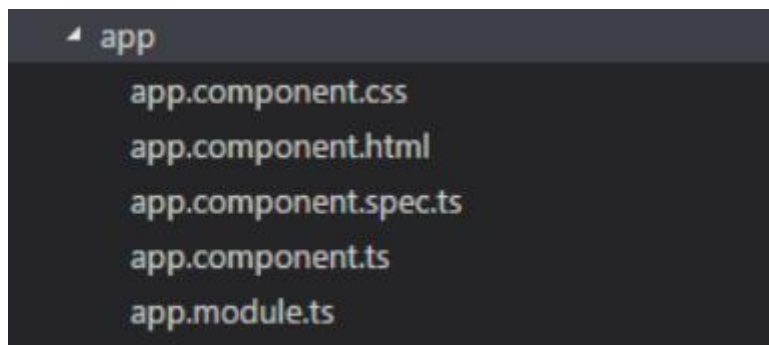


Najważniejsze rzeczy:

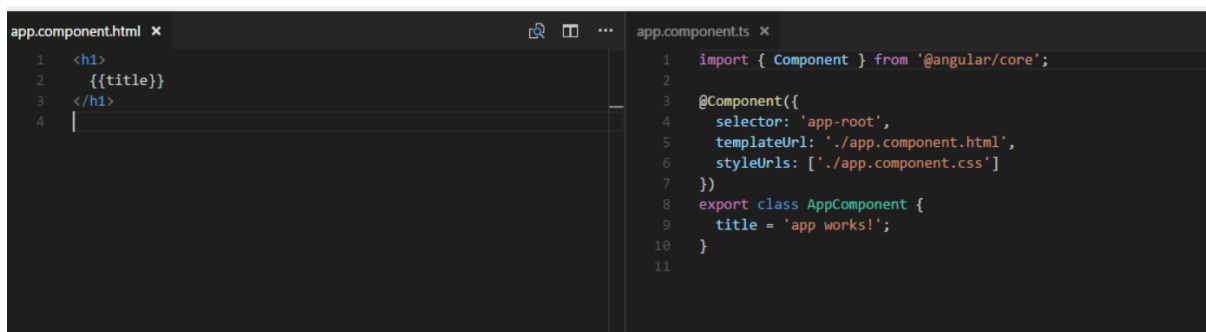
- Node_modules – zależności aplikacji pobrane po instalacji
- Src – kod źródłowy, najważniejsze elementy:
 - App – tutaj znajduje się kod angulara
 - Assets – różne przydatne do tworzenia aplikacji elementy (obrazki itp.) tutaj będą ładować
 - Environments – konfiguracja środowisk

- Index.html – aplikacje angulara są aplikacjami pojedynczej strony, index.html jest ich punktem wyjściowym
- Karma.conf.js – konfiguracja testów
- Package.json – zależności aplikacji
- Tsconfig.json – konfiguracja angulara

Szczegółowe informacje o tych plikach poznamy niebawem, tymczasem należy jeszcze spojrzeć na kod aplikacji w folderze src/app.



W angularze stosowana jest konwencja według której każdy widok (lub jego fragment wyeksportowany do innego pliku w celu ponownego użycia w innym miejscu) ma do siebie przypisany komponent odpowiedzialny za obsługiwane operacji w nim obsługiwanych. Przyjrzyjmy się temu na powyższym przykładzie



```
<button (click)="sendForm()">Send</h1>
```

o samej zasadzie działania tego wszystkiego za chwilę. Zwróćmy jednak uwagę na najważniejsze elementy naszego komponentu:

- `@Component({ ... })` – deklaracja komponentu (podobnie jak to ma miejsce w Springu) (inne adnotacje także poznamy 😊)
 - Selector – identyfikator, za jego pomocą można dany fragment kodu html wraz z logiką wkleić w innym pliku html. W tym przypadku wystarczy: `<app-root></app-root>`
 - templateUrl – ścieżka do wyświetlanej treści
 - template – można wykorzystać zamiast templateUrl – wtedy zamiast ścieżki bezpośrednio piszemy kod html, zaleca się nie używać chyba że bardzo krótkie fragmenty
 - styleUrls – ścieżka do stylów css.

Klasy w Angularze zazwyczaj poprzedzamy słowem kluczowym `export`.

Pozostało nam tylko zajrzeć do pliku `app.module.ts`. Wygląda on następująco:

```

app.module.ts x
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { FormsModule } from '@angular/forms';
4  import { HttpClientModule } from '@angular/http';
5
6  import { AppComponent } from './app.component';
7
8  @NgModule({
9    declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule,
14     FormsModule,
15     HttpClientModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21

```

Plik ten łączy całą logikę zawartą w aplikacji, znajdują się tu deklaracje komponentów (`declarations`), informacje jakie moduły należy dowiązać (`imports`), informacje o serwisach odpowiedzialnych za

komunikację w i poza aplikacją (providers) oraz informacja co należy bootstrapować. Na chwilę obecną tyle, jak należy o to wszystko dbać dowiemy się niebawem.

Mając obecną wiedzę możemy poznać już jakieś tajniki angulara!

4. Zmienne oraz wiązanie danych

Przyjrzyjmy się następującemu fragmentowi kodu:

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

  title = 'app works!';

  num : number;
  str : string;
  str2 : string = "Initiated";
  arr : string[];
  model : any;

  ngOnInit(): void {
    this.num = 3;
    this.str = "In constructor";
    this.arr = [];
    this.arr.push("Element");
    this.model = "sd";
  }
}
```

Mamy tu kilka ważnych informacji. Podobnie jak w Javie klasy mogą implementować odpowiednie interfejsy. W tym przypadku implementowany jest interfejs OnInit, jego implementacja polega na

wykonaniu metody `ngOnInit` – zostanie ona automatycznie wykonana tuż po załadowaniu komponentu przez przeglądarkę.

Zmienne w klasie można inicjować w sposób pokazany powyżej, warto zwrócić uwagę na możliwość nadania im wartości przy deklaracji, oraz na specjalny typ `any` – oznaczający że wszystko może zostać przez niego przechowane.

Do zmiennych w ciele metod odwoływać będziemy się z wykorzystaniem znanej z javy zmiennej `this`. Przyjrzyjmy się kodowi html oraz efektowi w przeglądarce.

```
app.component.ts  app.component.html x
1  <h1>
2    {{title}}, {{num}}, {{str}}, {{str2}}, {{arr[0]}}, {{model}}
3  </h1>
4
```

app works!, 3, In constructor, Initiated, Element, sd

Mamy tu przykład pierwszego wiązania danych - interpolacji. Interpolacja polega na otoczeniu zmiennej zawartej w kontrolerze w podwójne nawiasy `{{ }}` – pozwala to na wyświetlenie zawartości tych zmiennych w przeglądarce.

Jeśli chodzi o pozostałe wiązania danych to wyróżniamy:

- Wiązanie jednostronne (One Way Binding) – np. `<h1 [innerText]='title'></h1>` - brak nawiasów `{{ }}`, w przypadku zmiany zmiennej `text` zmieniona zostanie także zawartość `h1`
- Wiązanie dwustronne (Two Way Binding) – najczęściej wykorzystywany rodzaj wiązania. Wykorzystuje się je poprzez podanie modelu do elementu html (np. `input`, `checkbox` itp.) – np: `<input [(ngModel)='title']` – zmiana wartości w `input`'cie spowoduje zmianę wartości zmiennej do niego przypisanej. Żartobliwie opisuje się ten rodzaj wiązania jako *Banan w Pudełku* (Banana in the Box) – ze względu na wykorzystane nawiasy `[()]` ☞
- Przypisanie do zdarzenia (Event Binding) – przypisanie funkcji angularowej do zdarzenia (np. kliknięcia guzika): `<button (click)='doSth()'>{{title}}</button>`. Więcej o tym wiązaniu na następnej lekcji.

5. Zadanie do wykonania:

Zapoznaj się z strukturą aplikacji, następnie zadeklaruj kilka zmiennych w `app.component.ts` i wyświetl je w przeglądarce. Następnie zadeklaruj zmienną tekstową i przypisz się za pomocą `ngModel` do inputu tekstowego, obok niego dodaj etykietę (`<p>`, `<h1-6>` itp.) która będzie wyświetlała jego zawartość i zaobserwuj czy zmienia się razem z wartością w inputcie. Dodatkowo utrudnienie: podłącz do aplikacji bootstrapa 3 (dodaj link do skryptu w pliku `index.html`) i spróbuj wszystko jakoś ładnie wyświetlić 😊