

THD(C) - Frontend (UI)

Warstwa prezentacji aplikacji THD(C)

Spis treści

1. Przygotowania
2. Uruchomienie
3. Wytyczne
4. Komendy Angular CLI
5. Dodawanie nowego tłumaczenia
6. Dostępne komendy aplikacji

Przygotowania

Przed rozpoczęciem prac należy zainstalować narzędzie nvm - menedżer do zarządzania wersjami **Node.js** lub nvm-windows jeśli używasz systemu operacyjnego Windows. Przy pomocy ww. narzędzia pobieramy Node.js w wersji **20.11.1 (LTS)**, który domyślnie instaluje również npm w wersji **10.2.4**:

```
nvm install 20.11.1
```

Następnie aktualizujemy npm do wersji **10.9.0**:

```
npm install -g npm@10.9.0
```

Możemy teraz zainstalować wszystkie dependencje w projekcie. Otwórz terminal i przejdź do katalogu, w którym znajduje się aktualnie przeglądany plik README.md i uruchom komendę:

```
npm install
```

Uruchomienie

Otwórz terminal, przejdź do głównego katalogu projektu np. `C:/Projects/THD-C/Frontend/` lub `/home/projects/THD-C/Frontend` a następnie uruchom polecenie:

```
npm start
```

⚠ Na systemie operacyjnym Windows możesz napotkać poniższy błąd:

```
File C:\path\to\project\ cannot be loaded because running scripts is d
about_Execution_Policies at https://go.microsoft.com/fwlink/?LinkID=135
At line:1 char:1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityExceptio
+ FullyQualifiedErrorId : UnauthorizedAccess
```

W takim przypadku uruchom PowerShell jako administrator i wykonaj polecenie:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

zmiana `ExecutionPolicy` na `RemoteSigned` pozwala na pobranie i uruchomienie skryptów, które mają podpis cyfrowy. Warto mieć to na uwadze.

Nierekomendowane jest ustawianie wartości na `Unrestricted` !

Wytyczne

Co do strukturyzowania katalogów zaleca się, aby elementy Angular'owe przechowywać w grupach tj.:

- Moduły w katalogu -> `/src/app/modules/` ,
- Serwisy w katalogu -> `/src/app/services/` ,
- Strażników w katalogu -> `/src/app/guards/` ,
- Dyrektywy w katalogu -> `/src/app/directives/` ,
- "Rury" w katalogu -> `/src/app/pipes/` ,
- z kolei Komponenty:
 - jeśli są to kontrolki współdzielone czyli używane w wielu różnych miejscach to w katalogu -> `/src/app/shared/components` ,
 - jeśli są to elementy widoku w ramach danego modułu np. panel administratora to odpowiednio w `/src/app/modules/admin/components` ,
`/src/app/modules/home/components` itp.


W przypadku plików związanych z modelami danych w komponencie (czyli pliki `<nazwa-modelu>.model.ts`) przechowywać w katalogu komponentu.


Jeśli w komponencie pojawią się metody lub pola, które odpowiadają za konfigurację kontrolki, to należy je wydelegować do osobnego pliku w katalogu komponentu o nazwie `<nazwa-komponentu>.config.ts` np.

```
export const showMore: boolean = true;

export const displayFormat = (data: any) => {
  return `${data} PLN`;
};
```

Komendy Angular CLI

 Upewnij się, że Terminal wskazuje na główny katalog projektu (czyli tam gdzie znajduje się aktualnie czytany plik)!

 Domyślnie generator wskazuje na ścieżkę `./src/app` . Warto mieć to na uwadze!

Moduł

Skupia, przechowuje, wyodrębnia pewną część logiki aplikacji. Najczęściej będzie tworzone w momencie dodawania kompletnie nowego widoku np. `/home` , `/user` , `/admin` etc.

Tworzenie:

```
ng generate module modules/<module-name>
```

skrótowy zapis:

```
ng g m modules/<module-name>
```

Komponent

Jest to część widoku, który wykonaliśmy od 0 samodzielnie i zawiera logikę biznesową lub custom'owy element/kontrolka, np. `<tn-time-picker></tn-time-picker>` . Komendy CLI oparte są na wytycznych. Dobrze, aby się z nimi zapoznać.

Tworzenie:

```
ng generate component --standalone=false modules/<module-name>/components/
```

skrótowy zapis:

```
ng g c --standalone=false modules/<module-name>/components/<component-name>
```

Parametr `--standalone=false` generuje komponent, który jest zależny od modułu. Komponent będzie można użyć tylko gdy zostanie zadeklarowany w module np.

```
@NgModule({  
  declarations: [YourComponent],  
})  
export class YourModule {}
```

Serwis

Zapewnia modularyzację, wstrzykiwanie zależności (dependency injection). W nim powinno być zawarte wszystko co **nie jest** związane z logiką biznesową czyli obsługa żądań HTTP do API, manipulacja danymi w session/local storage'u.

Tworzenie:

```
ng generate service <service-name>
```

skrótowy zapis:

```
ng g s <service-name>
```

"Rura"

Formatuje dane do pewnej postaci. Przykład:

w komponencie mamy pole `amount = 100`, które potrzebujemy sformatować do postaci `100.00`. Możemy to uzyskać przy użyciu wbudowanego pipe'a `number`: `{{ amount | number: '1-2.2' }}`. Wartość pola `amount` zostaje przesłana do formatora `number`, który na widoku pokazuje `100.00`.

Tworzenie:

```
ng generate pipe <pipe-name>
```

skrótowy zapis:

```
ng g p <pipe-name>
```

Dyrektywa

Pozwala na nałożenie na elementy w DOM'ie pewnych dodatkowych funkcjonalności, stylów itp.

Tworzenie:

```
ng generate directive <directive-name>
```

skrótowy zapis:

```
ng g d <directive-name>
```

Strażnik

Pośrednik, który wykonuje się w momencie wejścia na dany widok. Dzięki temu można zabezpieczyć pewne widoki administracyjne przed zwykłymi użytkownikami.

Tworzenie:

```
ng generate guard <guard-name>
```

skrótowy zapis:

```
ng g g <guard-name>
```

Inne

Angular CLI ma jeszcze inne opcje jak tworzenie klasy, interfejsu, enum'a. `ng generate class|interface|enum`

Tłumaczenia

Gdy w aplikacji będziemy chcieli dorzucić kolejne tłumaczenia aplikacji do poszczególnych należy dodać:

- w `xliffmerge.json` w tablicy `xliffmergeOptions:languages` dorzucić własny identyfikator języka:

```
{
  "xliffmergeOptions": {
    (...)
    "defaultLanguage": "en",
    "languages": ["en", "pl", "<twoj_jezyk>"],
    (...)
  }
}
```

- w `angular.json` w sekcji `projects:Frontend:i18n` dorzucić ścieżkę do kolejnej wersji językowej:

```
"projects": {
  "Frontend": {
    "projectType": "application",
    "schematics": {
      "@schematics/angular:component": {
        "style": "scss"
      }
    },
    "root": "",
    "sourceRoot": "src",
    "prefix": "app",
    "i18n": {
      "sourceLocale": "en-US",
      "locales": {
        "en": "src/locales/messages.en.xlf",
        "pl": "src/locales/messages.pl.xlf",
        "<twoj_jezyk>": "src/locales/messages.<twoj_jezyk>.xlf"
      }
    }
  }
  (...)
}
```

Dostępne komendy aplikacji

Komenda	Opis
<code>npm start</code>	Uruchamia lokalny serwer deweloperski na <code>localhost:4200</code> . Jeśli chcesz uruchomić serwer lokalny na innym porcie to w <code>angular.json</code> w sekcji <code>projects.TravelNestUI.architect.serve.options</code> podmień wartość pola <code>port</code> a następnie uruchom komendę
<code>npm run build</code>	Kompiluje projekt aplikacji i wygenerowane pliki wrzuca do katalogu <code>dist/</code>

Komenda

Opis

npm run
test

Uruchamia lokalne testy jednostkowe przy wykorzystaniu Karma

npm run
watch

Uruchamia lokalny serwer na podstawie zbudowanej paczki plików
w katalogu dist/

npm run
lint

Uruchamia linter'a, który sprawdza poprawność kodu na podstawie
reguł ustalonych w .eslintrc.json

npm run
extract

Generuje tłumaczenia. Najpierw tworzy plik messages.xlf, na
podstawie którego generowane są osobne tłumaczenie m.in w
języku polskim (messages.pl.xlf) oraz angielski (messages.en.xlf -
DOMYŚLNY!) Po puszczenie komendy tagi <target
state="new"/> należy uzupełnić poprawnym tłumaczeniem w
danym języku