**Laboratory 3Angular framework - basics**

# Introduction

In this laboratory, we will create and run a Hello World application using the Angular framework and then expand it with several possibilities using its various elements.

# Environment configuration and first application

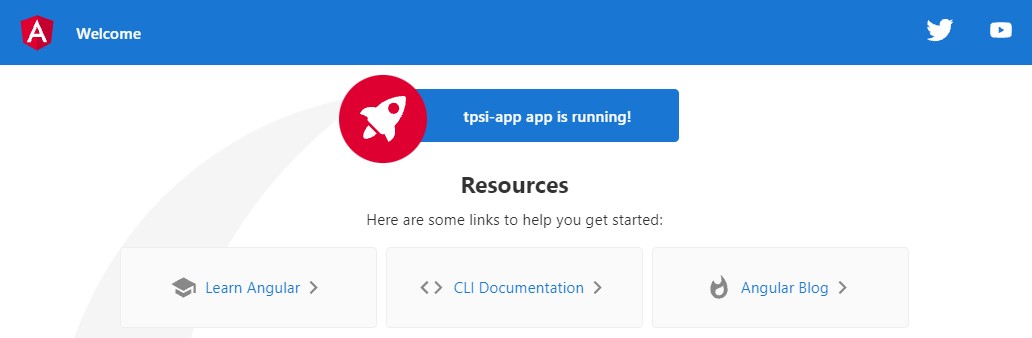
At the beginning, we have to install the Node.js runtime environment and the NPM (package manager) attached to it, if it is not already installed. We can check if it is installed by calling the following command in the console:

> **node** -v

> **npm** -v

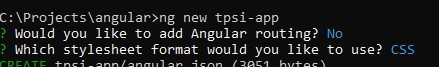
1. We have to install the Angular CLI via the NPM package manager using the following command in the console:

> **npm** install -g **@angular/cli**

Note: the -g flag installs the package globally, so you only need to install the CLI environment once. Now we can create a new application using the installed CLI.

> **ng** new <application name>

We can choose the version without routing and the style sheet format of our choice.



After installing all the necessary packages and creating the application via CLI, we can run the created framework by calling the command ng serve in the application folder:

Note that the flag --open will cause the automatic launching of the browser on the home page of our application. So we run the commands:

> **cd** <application name>

> **ng** serve --open

2. After starting the development server, we should see the Hello World page created by Angular (which will be serve by default at the address <http://localhost:4200>)

# Angular application project structure

When we open a new application project, we should see a file tree similar to the one in the picture:

node\_modules : contains all the npm packages necessary for the application to runsrc/app: the main folder of our application, here we write the source codesrc/assets: images, documents or other static filessrc/environments: application configurationsrc/index.html - the main page that will be sent when someone visits the site: there is usually no need to edit this filesrc/styles.css - global styles for the entire projectmore information about other Angular configuration files you can find: https://angular.io/guide/file-structure

# Excercise 1

Create a new component that will print a heading on the screen (e.g. Welcome to my website). Display a sample field from the component class instead of static text.

To create a new component, we can use the CLI to execute the following command:

> **ng** generate component hello

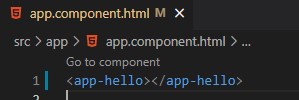
Short version:

> **ng g c <component-name>**

A new component will then be created in a separate folder, consisting of four files:CSS stylesheet (only works within a component)HTML documentunit test file (spec.ts)component/script logic (.ts)After opening the component file ( hello.component.ts ), we see the TypeScript class:

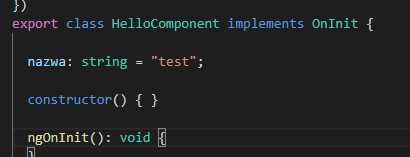
|  |
| --- |
| **import** { Component, OnInit } **from** '@angular/core' ;  @Component({ selector: 'app-hello' ,  // a tag with which we can declare our component in the file  HTML templateUrl: './hello.component.html' ,  // name of the HTML template file  styleUrls: [ './hello.component.css']  // stylesheet name  }) **export class HelloComponent implements OnInit** { **constructor** () { } ngOnInit(): **void** { }  } |

Modify the existing HTML file ( app.component.html) to add your own component by adding a tag from the above class:

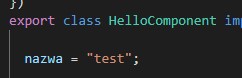


When the development server notices a change in the source files, it will automatically reload the page in the browser.

Note: To declare a field in a component class, we use the following syntax:



If the type of the field can be inferred from its contents, we do not need to specify it (the following syntax is also valid):



We can then use our variable directly in the HTML file, using double braces:

**<h1>{{name}}</h1>**

# Exercise 2: personalizing Hello World.

Add an input element that will be connected directly to the variable to personalize the user's name in the header (e.g. Welcome to my website, Marek)

To connect the input field to a variable, we use the ngModel attribute:

<**h1**>Welcome, {{name}}</**h1**>

<**input** type="text" ngModel="name">

After compiling the code, we will get an error:



**!! Module is missing. To use ngModel, we must first import the FormsModule in our application's module!!**

**app.module.ts:**

1. we import the module:

**import** { FormsModule } **from** '@angular/forms' ;

1. we must add it into the array:

|  |
| --- |
| imports: [  BrowserModule,  FormsModule ] , |

Now we can use ngModel in our application (in a module).More details: Built-in directive

# Exercise 3: We are adding a click counter

Add a simple counter, and two buttons (increment by 1 (+) and decrement by 1 (-) ).

To add a method to handle an event, we use round brackets:

< **button** ( click)= "incrementCounter()">+< / **button** >

Then we implement the method and field in our class:

incrementCounter() { **this** .counter++;

}

|  |
| --- |
| **export class HelloComponent implements OnInit** { // ...  counter: number = 0; // ...  } |

We proceed similarly for the button that decreases the counter.

|  |  |  |
| --- | --- | --- |
| **{{ }}** | {{ zmienna }} | z klasy do widoku, wypisanie wartości |
| **[ ]** | [ value] = “zmienna” | z klasy do widoku,  powiązanie własności HTML |
| **( )** | ( click) = “funkcja()” | z widoku do klasy, powiązanie zdarzenia |
| **[( )]** | [( ngModel)] = “zmienna” | powiązanie dwukierunkowe |

# exercise 4: To-do list version 1

Create a simple to-do list in a new component (only displaying data for now) and add it to the home page (use the \*ngFor directive)

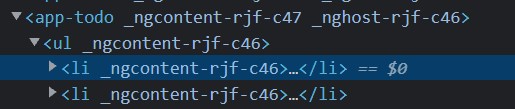
First, let's declare an array of objects in our component where we will store our stuff:

|  |
| --- |
| **export class HelloComponent implements OnInit** { // ...  items = [  { id: 1, name: "Take out the trash" } ,  { id: 2, name: "Go shoping" ,  ] ;  // ...  } |

We can then list our elements using the \*ngFor directive:

|  |
| --- |
| < ul>  < **li** \*ngFor= "let item of items" >  {{ item.id }} - {{ item.name }}  < / **li** > < /ul> |

After opening the page in the browser, we will see the generated elements in the page structure:



# Exercise 5: To-do list version 2

Extend the component with implementation of adding an item to a list.

We can add a new element by using the click event and the ngModel directive:

<**input** type= "text" [( ngModel)]="todoItem" >

< **button** ( click)= "addItem()">Add task </ **button** >

It is worth adding a field that will help us autoincrement the id field:

lastId = 2;

We can implement adding an element to the list using the .push() method:

**const item = { id: ++ this .lastId, name: this .todoItem }; this.items.push(item);**

It would also be a good idea to clear the input field after adding the item.

# exercise 6: to-do list version 3

Extend the component by implementing the removal of an item from the list.

We will add a delete button to each field, and in the event handler we provide the ID of the given element:

|  |
| --- |
| < **li** \*ngFor= "let item of items" >  {{ item.name }}  < **button** ( click)= "deleteItem(item.id)"> X</ **button**>  < / **li** > |

An example implementation of deleting a selected item from an array (we find the item with given index in the array by ID and then remove it from the list):

|  |
| --- |
| deleteItem(id: number) { **const** index = **this** .items.findIndex(obj => obj.id == id); **if**( index !== -1) { **this** .items.splice(index, 1);  }  } |

If the index is equal to -1, it means that the element is not in our array.

The above description of the anonymous function: obj => obj.id == id shortens the following function:

**function**( obj) { **return** ( obj.id == id);

}

Arrow [Functions - JavaScript | MDN](https://developer.mozilla.org/pl/docs/Web/JavaScript/Reference/Functions/Arrow_functions)

# Exercise 7: to-do list version 4.

Extend the component in such a way that when user try to add the same task to the list as one of the already added tasks, an appropriate message is displayed and the element is not added.

When adding an element, we must to check whether the element does not already exist in the given array. After slightly transforming, we can again use the method .findIndex() :

|  |
| --- |
| **const** index = **this** .items.findIndex(obj => // warunek do spełnienia ) ; **if**( index !== -1) {  // what should happen if we find an existing element?  } |

To display the message, we can use the \*ngIf directive - if the variable is true, the HTML element will be displayed on the page.

< **div** id= "error" \*ngIf= "errorBoolean" > The item already exists!!

< / **div** >

Our component after adding modification will look like this:

|  |
| --- |
| **export class TodoComponent implements OnInit** {  // ...  todoItem = "" ; errorBoolean = **false** ; // ...  addItem() { **const** index = **this** .items.findIndex(obj => obj.name == **this** .todoItem ); **if**( index !== -1) { **this** .errorBoolean = **true**;  **return**; // we exit the method, skipping the rest of the code }  **const** item = { id: ++ **this** .lastId, name: **this** .todoItem } **this** .items.push(item);  **this** .todoItem = "" ;  **this** .errorBoolean = **false** ;  } |

# Exercise 8 (to be done on your own)

Using previous tasks, create a page with a to-do list and the ability to remove and add them. As in the application from the previous task, each added task should have a symbol or button that, when clicked, will remove that task from the list. Each task on the list should consist of a name, description, priority "important", "medium important", etc. and the deadline for its completion.Tip: The ngModel directive also works for drop-down lists in a similar way:

|  |
| --- |
| < **select** [( ngModel)]= "priority" >  < **option** value= "1">1< / **option**>  < **option** value= "2">2< / **option**>  < **option** value= "3">3< / **option** > < / **select** > |

# Useful links:

Introduction to ANgular: [Introduc on to the Angular Docs](https://angular.io/docs)

JavaScript documentation: [JavaScript | MDN](https://developer.mozilla.org/pl/docs/Web/JavaScript)