JS

Лабораторная работа 9.

Сервис – сущность angular, которая используется в основном для:

* Предоставления данных приложению (например из внешних источников).
* Использования в качестве канала взаимодействия между отдельными компонентами приложения.

1. **Рассмотрим приложение services-less.**

Приложение состоит из 3х компонентов:

app-root

app-product-interface, содержащий элементы интерфейса (input text и button)

app-product-element, отвечающий за вывод и логику внутри отдельного продукта.

Так же в приложении в папке services есть 2 сервиса ConsolePrintService и ProductsService.

Создание нового сервиса.

ng generate service путь\имя

например, **ng generate service services\testsrv** или **ng g s services\testsrv**

**Сервис ConsolePrintService**

представляет собой класс с одним методом:

log(str: string) {

console.log(str);

}

Который выводит в консоль строку str.

Для того, чтобы использовать сервис в компоненте нужно в конструкторе включить в компонент объект класса сервиса (inject),

*constructor*(*private* consolePrint: ConsolePrintService) { }

импортировав при этом его,

*import* { ConsolePrintService } *from* '../services/console-print.service';

После этого мы можем воспользоваться объектом consolePrint.

*this.consolePrint.log('Имя продукта: ' + this.productName);*

Данный класс иллюстрирует использование сервиса, как класса с некоторыми обобщенными методами, которые могут быть сгруппированы и использованы в приложении.

**Сервис ProductsService**

В этот сервис входит массив продуктов с полями name (название) и flagBuy (куплен или нет)

products = [

{

name: 'Помидоры',

flagBuy: true

},

{

name: 'Огурцы',

flagBuy: false

}

];

И метод

addProduct(str: string) {

*this*.products.push({

name: str,

flagBuy: false

});

}

Который добавляет в массив products элемент с именем, взятым из параметра функции (str) и flagBuy = false.

Перейдем в компонент интерфейса (product-interface.component.ts) и рассмотрим использование данного сервиса.

В компоненте сервис ProductService импортирован и объявлен в конструкторе

*import* { ProductsService } *from* '../services/products.service';

*…*

*constructor*(*private* consolePrint: ConsolePrintService, *private* productsService: ProductsService) { }

Вернемся в компонент app-product-interface. В функции

onAddProduct() {

*this*.productsService.addProduct(*this*.productName);

*this*.consolePrint.log('Имя продукта: ' + *this*.productName);

}

Происходит вызов метода сервиса ProductsService add(). В параметрах указана строка, связанная с input text.

Перейдем в компонент app-root. В конструкторе класса объявляется объект сервиса и используется массив products из него.

В шаблоне данного компонента app.component.html при помощи ngFor этот массив выводится с помощью компонента app-product-element.

<div *class*="container">

<app-product-interface>

</app-product-interface>

<hr>

<app-product-element

*\*ngFor*="let product of products"

*[productElement]* = "product"></app-product-element>

</div>

1.2 Провайдеры: с версии angular 6+ при создании сервиса в декораторе Injectable появилась возможность указать, что данный сервис предоставляется во всем приложении, записью

*@*Injectable({

    providedIn: 'root'

})

Без этой записи – требуется указать сервис в списке провайдеров модуля, внутри которого он предоставляется (с учетом иерархии). Сервис нужно указать в массиве providers

providers: []

1. HttpClient

Рассмотрим проект service-placeholder. Он содердит всего один компонент, отображающий массив объектов Posts

<div *class*="row" *\*ngFor*="let post of posts">

<div *class*="col-2">

{{post.id}}

</div>

<div *class*="col-10">

{{post.title}}

</div>

<div *class*="col-12">

{{post.body}}

</div>

</div>

Данный массив берется из тестовой api <https://jsonplaceholder.typicode.com/>

В папке services находится сервис PostsService, наследуемый от BaseApi

*export* *class* PostsService *extends* BaseApi {

Рассмотрим BaseApi

Данный класс в этом проекте родительский для сервисов, осуществляющих http запросы. В нем созданы методы для осуществления основных http запросов post, put, get, delete <https://developer.mozilla.org/ru/docs/Web/HTTP/Methods>

Адрес api

*private* baseUrl = 'https://jsonplaceholder.typicode.com/';

И метод для формирования строки щапроса

*private* getUrl(url: string = ''): string {

*return* *this*.baseUrl + url;

    }

Для осуществления http запросов используется класс HttpClient, который добавлен в конструктор

*constructor*(*public* http: HttpClient) { }

С его помощью наш клиент будет отправлять запросы на сервер.

Рассмотрим метод для создания get запроса

*public* get(url: string = '', header: HttpHeaders): Observable<any> {

*let* requestOptions = {

            headers: header

        };

*return* *this*.http.get(*this*.getUrl(url), requestOptions);

    }

Данному методу нужна строка запросы (url) и заголовок запроса (header). Метод будет возвращать observable, который мы в дальнейшем переделаем в promise для удобства использования. <https://metanit.com/web/angular2/6.2.php>

По аналогии созданы методы для остальных запросов.

Рассмотрим сервис PostsService. Он наследует BaseApi

options: HttpHeaders;

*constructor*(*public* http: HttpClient) {

        super(http);

*this*.options = new HttpHeaders();

*this*.options = *this*.options.set('Content-Type', 'application/json');

    }

После вызова конструктора родительского классы, мы создаем header, Где указываем content-type.

Далее создаем методы для получения постов, на базе основных методов взаимодействия с api.

*async* getPosts() {

*return* *this*.get('posts', *this*.options).toPromise();

    }

Строка posts указывает на адрес запроса <https://jsonplaceholder.typicode.com/posts>

This.options – это заголовок.

.toPromise() – оборачивает ответ api в обещание для удобства использования.

Теперь вернемся в компонент app.

*try* {

*let* posts = *this*.postsServise.getPosts();

*this*.posts = (isNullOrUndefined(*await* posts)) ? [] : *await* posts;

        } *catch* (err) {

            console.log(err);

        }

Создаем переменную posts и записываем в нее обещание, которое вернул метод сервиса.

В результате это обещание будет массивом постов, либо null, в случае, если постов нет. В this.posts мы запишем содержимое posts, а в случае null – пустой массив.

1. Домашнее задание

Переделать приложение из работы №13 так, чтобы оно работало с бд. API можно сделать при помощи json-server.