

Пособие молодого бойца

Рахимов Камол

2.0.0, апреля 30, 2020

Содержание

1 Настройка Angular	1
1.1 Необходимые компоненты	1
1.2 Необходимые для реализации компоненты	2
1.3 Среда разработки	3
2 Структура приложения Angular	4
2.1 Модули	4
2.2 Компонент	4
2.3 Сервис	5
2.4 Директива	5
3 Учебный пример	6
3.1 Конечный результат учебного примера	6
3.2 Реализованные Angular приложение и Node.js сервер	8
3.2.1 Установка зависимостей Angular приложения и Node.js сервера	8
3.2.2 Запуск приложения	8
4 Создание компонентов программы	10
4.1 Импорт необходимых компонентов и модулей	10
4.1 Создание компонента журнальной формы (таблицы)	11
4.2 Создание компонента для редактирования и добавления в таблицу	12
5 Разработка задания	13
5.1 Подключение и работа с библиотекой HttpClientModule	13
5.2 Разработка функционала компонента журнальной формы (таблицы)	14
5.2.1 Подгрузка данных для таблицы из сервера	16
5.2.2 Создание компонентов удаления и редактирования, а также разработка функционала удаления справки и разработка валидации	17
5.2.2.1 Создание компонентов удаления и редактирования	17
5.2.2.2 Разработка функционала удаления справки	18
5.2.2.3 Разработка функционала редактирования справки	19
5.2.2.4 Разработка функционала компонента с категориями прав	20
5.2.2.5 Разработка валидации полей ввода	22
6 Индивидуальное задание	24

1 Настройка Angular



Angular — это открытая и свободная платформа для разработки веб-приложений, написанная на языке TypeScript. Документация по фреймворку можно посмотреть на официальном сайте - <https://angular.io/docs>. Исходники фреймворка находятся на GitHub - <https://github.com/angular/angular>.

1.1 Необходимые компоненты

Для работы с Angular, сперва, необходимо скачать и установить сервер **Node.js** (<https://nodejs.org/ru/>), в нем находится менеджер пакетов **npm**, благодаря которому мы скачаем Angular и все необходимые к нему компоненты.

Установка Angular CLI (интерфейс командной строки, который позволяет вам быстро создавать приложения, добавлять файлы и выполнять множество определенных задач, такие как тестирование, сборка и развертывание), откроем терминал (Windows PowerShell, командная консоль) и выполним следующую команду:

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

Данная команда установит пакет `@angular/cli` в качестве глобального модуля, поэтому в последующих созданиях новых приложений Angular, его не потребуется устанавливать заново. Теперь у вас есть Ангуляр в вашем компьютере и вы можете создавать приложения на данном фреймворке.

Чтобы создать приложение, необходимо в терминале выполнить следующее:

```
ng new MyTestProject
```

Где последнее слово будет является наименованием вашего приложения. Далее будут вопросы:

```
? Would you like to add Angular routing?
```

Angular CLI предлагает нам **routing** (маршрутизация между страницами вашего

приложения). Пишем в терминале - **No** и жмем **enter**. Нет необходимости в маршрутизации.

```
? Which stylesheet format would you like to use?
```

Angular CLI предлагает нам выбрать формат для каскадных стилей. Выбираем самый простой - **CSS**.

Приложение создано и будет располагаться в каталоге **MyTestProject** (как вы и наименовали ваше приложение).

1.2 Необходимые для реализации компоненты

Необходимо установить в приложение библиотеку компонентов **UI5 Web Components**. Данная библиотеки имеет готовый набор веб-компонентов, например - диалоговые окна, стилизованные кнопки, форма с панелью выбора даты и т.д. (можете подробнее ознакомиться с каждым предлагаемыми ими компонентами - <https://sap.github.io/ui5-webcomponents/playground>). В корневом каталоге нашего приложения необходимо открыть терминал и выполнить следующую команду:

```
npm install @ui5/webcomponents@1.0.0-rc.6
```

Тем самым закачали и установили самую последнюю версию данной библиотеки.

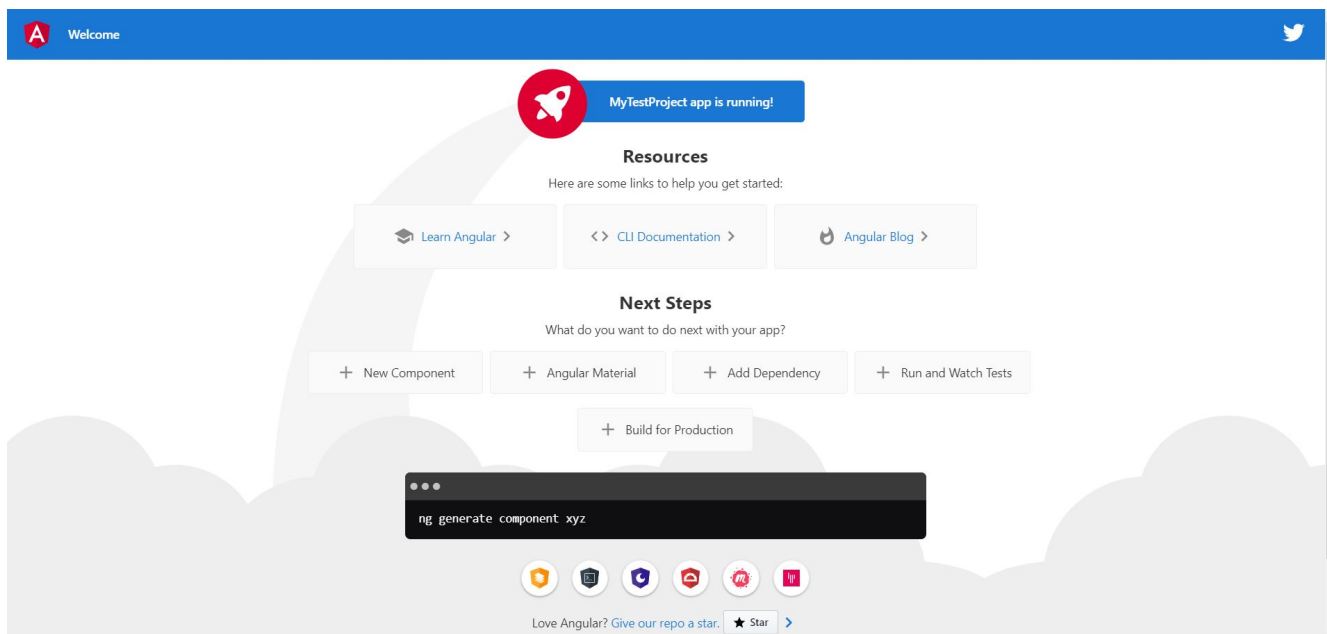
Запуск приложения:

В каталоге вашего приложения откройте терминал и пропишите следующее:

```
ng serve
```

Теперь у вас запущено Angular приложение по ссылке - <http://localhost:4200/>

Перейдя по ссылке, у вас должна открыться страница с таким же содержанием:



? Если возникла следующая ошибка:

```
An unhandled exception occurred: Port 4200 is already in use. Use '--port' to specify a different port.
```

Данная ошибка означает, что порт 4200 вами уже занят, необходимо выполнить в терминале следующее:

```
ng serve --port 4201
```

Где последнее это цифра порта (можно указать любой), на котором будет лежать Angular приложение.

1.3 Среда разработки

Три лучших **IDE** (ИСП - интегрированная среда разработки), которые удобны для web-разработки:

- **Visual Studio Code** - удобный, бесплатный, не требует незначительное количество оперативной памяти. Ссылка на официальный сайт - <https://code.visualstudio.com/>
- **Webstorm** - платный, удобный, расширенное количество функционала, проблемы с потреблением оперативной памяти. Ссылка на официальный сайт - <https://www.jetbrains.com/ru-ru/webstorm/>
- **Sublime Text** - платный, на любителя, обычный редактор, из под коробки не имеет большой функционал (необходимо докачивать определенные плагины, например синтаксический сахар). Ссылка на официальный сайт - <https://www.sublimetext.com/>

2 Структура приложения Angular

Сам фреймворк состоит из нескольких библиотек (или модулей), каждая из которых содержит в себе определенный функционал, а каждый модуль состоит из совокупности классов и их свойств и методов.

Не все библиотеки обязательны для использования в приложении, часть подключается по мере необходимости, например, **FormsModule** или **HttpClientModule**.

2.1 Модули

Разберем модули, именно с них начинается проектирование архитектуры Angular приложения. Каждый из них имеет собственный набор структурных элементов:

- **component** - отвечает за часть web-страницы и включает в себя HTML-шаблон, CSS-стили и логику поведения;
- **service** - поставщик данных для component;
- **directive** - преобразует определенную часть **DOM** (Document Object Model/Объектная Модель Документа - это программный интерфейс для HTML и XML документов, подробнее про директивы вы можете прочитать по данной [ссылке](#)) заданным образом.

Все вышеперечисленное собирается в корневой модуль, который общепринято называется **AppModule**.

Корневой модуль может быть только один, но он может использовать функционал других модулей, объявленных в объекте декоратора **@NgModule()** в свойстве **imports**.

@NgModule() - это декоратор, который принимает описывающий модуль объект.

Перечень свойств объекта:

- **declarations** - компоненты (**Component**), директивы (**Directive**) и фильтры (**Pipe**) корневого модуля;
- **exports** - компоненты, сервисы, директивы и фильтры, доступные для использования разработчикам, которые будут использовать ваш модуль в своих разработках;
- **imports** - другие модули, используемые в корневом модуле;
- **providers** - сервисы (**Service**) приложения;
- **bootstrap** - имя главного компонента приложения (как правило, называется **AppComponent**).

2.2 Компонент

Компонент - это часть интерфейса приложения с собственной логикой. Вся видимая часть Angular App реализуется с помощью компонентов, поэтому часто можно услышать, что архитектура Angular компонентная.

Ранее уже упоминалось, что за создание компонента отвечает декоратор **@Component()**. Основные свойства объекта, который принимает декоратор:

- **selector** - название компонента;
- **template** (или **templateUrl**) - HTML-разметка в виде строки (или путь к HTML-файлу);
- **providers** - список сервисов, поставляющих данные для компонента;
- **styles** - массив путей к CSS-файлам, содержащим стили для создаваемого компонента.

2.3 Сервис

Сервисы нужны для предоставления данных компонентам. Это могут быть не только запросы к серверу, но и функции, преобразующие исходные данные по заданному алгоритму. Они позволяют архитектуре Angular приложения быть более гибкой и масштабируемой.

Задача сервиса должна быть узкой и строго определенной.

Не будет считаться ошибкой, если вы реализуете функционал в компонентах, но считается хорошей практикой все обращения к серверу и функции, возвращающие данные, выносить в сервисы.

2.4 Директива

Предназначение директив - преобразование **DOM** заданным образом, наделение элемента поведением.

По своей реализации директивы практически идентичны компонентам, компонент - это директива с HTML-шаблоном, но с концептуальной точки зрения они различны.

Есть два вида директив:

- **структурные** - добавляют, удаляют или заменяют элементы в **DOM**;
- **атрибуты** - задают элементу другое поведение. Они создаются с помощью декоратора **@Directive()** с конфигурационным объектом.

В Angular имеется множество встроенных директив (***ngFor**, ***ngIf**), но зачастую их недостаточно для больших приложений, поэтому приходится реализовывать свои.

Каждому из блоков архитектуры Angular посвящена отдельная глава, где все подробно расписано и показано на примерах.

Подробнее про файловую структуру и их предназначение вы можете прочитать в официальном сайте Angular - <https://angular.io/guide/file-structure>

3 Учебный пример

Необходимо реализовать учебный Angular проект, в котором должна быть таблица с шоферскими справками, а так-же реализация определенных функций над таблицей, такие как:





- добавление элементов в таблицу;
- редактирование элементов в таблице;
- удаление элементов из таблицы;
- связь и работа с сервером посредством API;

3.1 Конечный результат учебного примера

Конечным результатом должен быть учет справок. Таблица с подгруженными данными из сервера (и кнопками, которые несут за собой определенный функционал).

Таблица шоферских справок:

+ Добавить

№	Серия	Номер	Категории	Дата	
1	720567	9966	A,B	27.03.2020	 
2	456756	1001	B	28.03.2020	 
3	902341	4422	B,C	29.03.2020	 

Диалоговое окно - добавление новой справки:

<

Диалоговое окно - редактирование справки:

+ Добавить

№	Серия
1	720567
2	456756
3	902341

Редактирование справки

Серия:

902341

Номер:

4422

Категория:

В X

С X

Введите категорию ..

▼

Дата:

29.03.2020

Отмена

Сохранить

3.2 Реализованные Angular приложение и Node.js сервер

Angular приложение приведенное выше на изображениях и Node.js сервер, находятся по данной [ссылке](#)

3.2.1 Установка зависимостей Angular приложения и Node.js сервера

- в каталоге "MedCertDrivers", откройте терминал и выполните следующее:

```
npm install
```

- в каталоге "NodeServer", выполните тоже самое.

Команда **npm install** необходима для загрузки и установки всех зависимостей (компонентов/моудлей) приложения. Зависимости указываются в файле **package.json**

3.2.2 Запуск приложения

- в каталоге "MedCertDrivers", запустите Angular приложение (как запускать приложение - см. главу 1.2)
- в каталоге "NodeServer" (находится Серверное приложение, написанное на Node.js,

данное приложение предоставит нам определённые API и данные для нашей таблицы), откройте терминал и выполните следующее:

```
node server
```

Далее должна появиться надпись - "**server started!**", означает, что сервер запущен и никаких проблем нет.

Учебное приложение запущено, можете протестировать его по ссылке - <http://localhost:4200/>

4 Создание компонентов программы

Прежде чем приступить к созданию, необходимо импортировать модули и компоненты.

4.1 Импорт необходимых компонентов и модулей

В вашем приложении, откройте файл `"/MyTestProject/src/app/app.module.ts"`, где расположен корневой модуль, далее импортируйте следующие компоненты:

```
import "@ui5/webcomponents/dist/Input.js";
import "@ui5/webcomponents/dist/features/InputSuggestions.js";
import "@ui5/webcomponents/dist/Table.js";
import "@ui5/webcomponents/dist/TableColumn.js";
import "@ui5/webcomponents/dist/TableRow.js";
import "@ui5/webcomponents/dist/TableCell.js";
import "@ui5/webcomponents/dist/Dialog";
import '@ui5/webcomponents/dist/Label';
import "@ui5/webcomponents-icons/dist/Assets.js";
import '@ui5/webcomponents/dist/DatePicker';
import "@ui5/webcomponents/dist/MultiComboBox";
import "@ui5/webcomponents/dist/MultiComboBoxItem";
```

Таким образом мы импортировали компоненты, которые необходимы для учебного задания - формы ввода, таблицы, диалогового окна и т.д., всё это компоненты из библиотеки компонентов **UI5 Web Components**.

Далее необходимо разобраться с данными компонентами на сайте - <https://sap.github.io/ui5-webcomponents/playground>, выбираем, например, компонент формы ввода - <https://sap.github.io/ui5-webcomponents/playground/components/Input/>. К каждому компоненту имеется его описание и функции. Компонент Input (форма ввода), имеет HTML тег:

```
<ui5-input></ui5-input>
```

Описание компонента расписано в пункте **"Overview"**, подробное описание функций, свойств и т.д., находится ниже пункта **"Overview"**.

Компоненты **UI5 Web Components**, не являются родными для Angular приложения, поэтому необходимо Angular указать, что мы используем чужеродные компоненты. В этом же файле (`"/MyTestProject/src/app/app.module.ts"`) импортируйте модуль **CUSTOM_ELEMENTS_SCHEMA** и поместите его в объект **@NgModule()**, а именно в свойство **schemas**:

```
@NgModule({
  ...
  schemas: [CUSTOM_ELEMENTS_SCHEMA],
  ...
})
```

4.1 Создание компонента журнальной формы (таблицы)

В файле `./MyTestProject/src/app/app.component.html`, создадим шаблон таблицы. Переходим к ссылке компонента таблицы <https://sap.github.io/ui5-webcomponents/playground/components/Table/>, разбираемся в нем (Важно! Многие компоненты описаны не подробно, и примеров использования компонентов бывает тоже недостаточно), и копируем к себе:

```
<ui5-table no-data-text="No Data" show-no-data>

  <ui5-table-column slot="columns" style="width: 3rem" demand-popin>
    <span style="line-height: 1.4rem"> </span>
  </ui5-table-column>
  <ui5-table-column slot="columns" style="width: 6rem">
    <span style="line-height: 1.4rem">Серия</span>
  </ui5-table-column>
  <ui5-table-column slot="columns" style="width: 6rem">
    <span style="line-height: 1.4rem">Номер</span>
  </ui5-table-column>
  <ui5-table-column slot="columns" style="width: 6rem">
    <span style="line-height: 1.4rem">Дата</span>
  </ui5-table-column>

  <ui5-table-row slot="default-1" *ngFor="let data of DataTable[0]?.Contracts">
    <ui5-table-cell slot="default">1</ui5-table-cell>
    <ui5-table-cell slot="default">39234</ui5-table-cell>
    <ui5-table-cell last-in-row slot="default">4444</ui5-table-cell>
    <ui5-table-cell last-in-row slot="default">14.03.2020</ui5-table-cell>
  </ui5-table-row>

</ui5-table>
```

Запустите приложение и у вас должна отобразиться таблица с четырьмя столбцами (, Серия, Номер, Дата) и одной строкой с данными (1, 39234, 4444, 14.03.2020). Протестируйте данный шаблон и попробуйте добавить больше столбцов и строк, для того чтобы понять, как устроен данный компонент.

4.2 Создание компонента для редактирования и добавления в таблицу

Так же в файле `./MyTestProject/src/app/app.component.html`, создаем шаблон диалогового окна (разместите данный компонент ниже таблицы). Переходим к ссылке компонента диалогового окна <https://sap.github.io/ui5-webcomponents/playground/components/Dialog/>, копируем к себе и добавляем еще несколько компонентов, которые необходимы для работы с данными по таблице (ссылка на компонент формы ввода - <https://sap.github.io/ui5-webcomponents/playground/components/Input/>, ссылка на компонент с панелью выбора даты - <https://sap.github.io/ui5-webcomponents/playground/components/DatePicker/>):

```
<ui5-dialog>
  <ui5-label>Серия:</ui5-label><br>
  <ui5-input placeholder="Введите серию .." style="width: 100%"></ui5-
input><br><br>

  <ui5-label>Номер:</ui5-label><br>
  <ui5-input placeholder="Введите номер .." style="width: 100%"></ui5-
input><br><br>

  <ui5-label>Категория:</ui5-label><br>
  <ui5-multi-combobox placeholder="Введите категорию .." style="width: 100%">
    <ui5-mcb-item text="A"></ui5-mcb-item>
    <ui5-mcb-item text="B"></ui5-mcb-item>
  </ui5-multi-combobox><br><br>

  <ui5-label>Дата:</ui5-label><br>
  <ui5-datepicker style="width: 100%" format-pattern='short'></ui5-
datepicker><br><br>

  <ui5-button design="Transparent">Отмена</ui5-button>
  <br><br>
  <ui5-button design="Emphasized">Сохранить</ui5-button>
</ui5-button>
</ui5-dialog>
```

В конечном результате у вас не отобразится диалоговое окно, так как мы не добавили кнопку, для того чтобы открыть данное окно. В следующих главах будет более подробно расписано про функционал компонентов и работу с данными.

В итоге - разобрались с базовыми компонентами, которые необходимы нам для нашего учебного задания.

5 Разработка задания

Установили в свое приложение компоненты, теперь необходимо придать логику и функционал компонентам, а также принимать и отправлять данные серверу.

5.1 Подключение и работа с библиотекой HttpClientModule

В файле `"./MyTestProject/src/app/app.module.ts"`, необходимо импортировать библиотеку **HttpClientModule**, благодаря которой возможна связь между сервисами front-end и back-end частей (подробнее про данную библиотеку можете прочитать [здесь](#)). Подобные сервисы так же называются - API (аббревиатура от Application Programming Interface). Импорт библиотеки:

```
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  ...
  imports: [
    ...
    HttpClientModule
    ...
  ],
  ...
})
```

Далее создаем файл `"http.service.ts"` в каталоге `"./MyTestProject/src/app/"`. В нем мы должны создать класс **HttpService**, методы которого мы будем использовать в нашем Angular компоненте:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class HttpService{

  constructor(private http: HttpClient){}

}
```

Чтобы указать, что сервис может использоваться в других сервисах, к классу сервиса применяется декоратор **@Injectable()**. Данный декоратор гарантирует, что встроенный механизм внедрения зависимостей сможет создать объект этого класса и передать его в качестве зависимости в другой объект (в другой сервис или компонент).

Необходимо реализовать методы:

- Подгрузка данных для таблицы, которые лежат на сервере.
- Подгрузка данных для компонента с категориями прав, которые лежат на сервере.
- Отправка новой справки на сервер, которую создал пользователь.
- Отправка справки на сервер, которая была отредактирована пользователем.
- Отправка идентификационного номера справки на сервер, которая была удалена пользователем.

Пример метода (POST), который отправляет идентификационный номер справки:

```
deleteData(id){
    return this.http.post('/api/deleteTableData/', {id});
}
```

Параметр метода **deleteData**, принимает от Angular компонента **id** (данный метод мы дальше вызовем в Angular компоненте) и дальше отправляется в качестве второго параметра для модуля **HttpClient** (который назначен в нашем классе, как - **http**). Первый параметр является путь к API серверу.

Пример метода (GET), который принимает все справки из сервера для таблицы:

```
getData(){
    return this.http.get('/api/getTableData/');
}
```

весь список API сервера:

- '/api/getTableData/' - получение всех справок (GET)
- '/api/getCategories/' - получение всех категорий прав (GET)
- '/api/addTableData/' - отправка новой справки на сервер (POST)
- '/api/deleteTableData/' - удаление определенной справки по идентификационному номеру (POST)
- '/api/editTableData/' - отправка отредактированной справки на сервер (POST)

? Почему путь не абсолютный - так как у нас Angular приложение и Node.js сервер запущены на разных хостах (**http://localhost:4200** и **http://localhost:3000**), то им обоим тяжело настроиться и разрешать общение с друг другом. Для этого нужно настроить в Angular приложении прокси конфигурацию (подробнее про проксирование расписано [здесь](#))

5.2 Разработка функционала компонента журнальной формы (таблицы)

Прежде чем выполнять удаление, редактирование и добавление, нужно получить данные таблицы из сервера.

Данные при вызове API метода - get('/api/getTableData'):

```
{
  "Contracts": [
    {
      "id": 1,
      "ser": "720567",
      "no": "9966",
      "issueDate": "27.03.2020"
    },
    {
      "id": 2,
      "ser": "456756",
      "no": "1001",
      "issueDate": "28.03.2020"
    },
    {
      "id": 3,
      "ser": "902341",
      "no": "4422",
      "issueDate": "29.03.2020"
    }
  ]
}
```

Данные при вызове API метода - get('/api/getCategories'):

```
{
  "Categories": [
    {
      "category": "A"
    },
    {
      "category": "B"
    },
    {
      "category": "C"
    },
    {
      "category": "D"
    },
    {
      "category": "M"
    }
  ]
}
```

5.2.1 Подгрузка данных для таблицы из сервера

В файле `./MyTestProject/src/app/app.component.ts` необходимо сперва импортировать наш сервис, который мы создали в главе 5.1 и указать Angular компоненту, что мы будем использовать сервис:

```
import { Component } from '@angular/core';
import { HttpService } from './http.service';
@Component({
  ....
  providers: [HttpService],
  ....
})
export class AppComponent {
  constructor(private httpService: HttpService){}
}
```

Далее импортируем модуль жизненного цикла Angular компонента (подробнее про жизненный цикл компонентов можете узнать по данной [ссылке](#)):

```
import { Component, OnInit } from '@angular/core';
import { HttpService } from './http.service';
@Component({
  ....
  providers: [HttpService],
  ....
})
export class AppComponent implements OnInit{
  constructor(private httpService: HttpService){}
  ngOnInit(){

  }
}
```

Создадим список, в котором будут лежать наши данные, по таблице, из сервера. Внутри **ngOnInit()** мы пропишем метод нашего **http** сервиса:

```
DataTable = [];
ngOnInit() {
  this.httpService.getData().subscribe(
    (data) => {
      this.DataTable.push(data);
    }
  );
}
```

Мы подписались на метод сервиса, при помощи функции **subscribe** и вставили новые

данные в список **DataTable**. Теперь при открытии страницы нашего Angular приложения, переменная **DataTable** будет заполнена данными из сервера.

Важно! Если не подписаться на метод сервиса, то метод не будет проинициализирован.

Так как таблица может иметь большое количество строк с элементами, то необходимо установить цикл, считывающий каждый элемент списка **DataTable**. Используем **html** тега **<ui5-table-row>** директиву **ngFor** (подробнее про директиву **ngFor** можете прочитать по данной [ссылке](#)), внутри которого необходимо создать новую переменную, которая будет пробегать по всему списку **DataTable**:

```
<ui5-table-row slot="default" *ngFor="let data of DataTable[0]?.Contracts">
```

Далее необходимо данные списка **DataTable** отобразить в таблице. В файле **"/MyTestProject/src/app/app.component.html"** необходимо интерполировать данные из Angular компонента (из списка **DataTable**). Для это используем интерполяцию **{{...}}** (подробнее про интерполяцию вы можете прочитать по данной [ссылке](#)).

```
.....
    <ui5-table-cell slot="default">
        {{data?.id}}
    </ui5-table-cell>

    <ui5-table-cell slot="default">
        {{data?.ser}}
    </ui5-table-cell>

    .....
```

Таким образом мы загрузили все данные из сервера в таблицу.

5.2.2 Создание компонентов удаления и редактирования, а также разработка функционала удаления справки и разработка валидации

5.2.2.1 Создание компонентов удаления и редактирования

Добавим в таблицу еще одну пустую колонну (добавьте новую колонну после тех, которые уже имеются):

```
<ui5-table-column slot="columns" style="width: 5rem"></ui5-table-column>
```

Далее в ячейках данной колонны будут лежать кнопки - удаления и редактирования справки. Расположим кнопки в строку:

```
<ui5-table-cell last-in-row slot="default">
  <span style="margin-right: 10px;">
    <ui5-button icon="delete"></ui5-button>
  </span>
  <span>
    <ui5-button icon="edit"></ui5-button>
  </span>
</ui5-table-cell>
```

Кнопки имеют свойство **icon**, в нем можно расположить только одно наименование определённой иконки (список всех иконок находятся по данной [ссылке](#))

5.2.2.2 Разработка функционала удаления справки

Чтобы удалить определённую справку в таблице, нам необходимо кликнуть на кнопку удаления и произвести удаление, как в таблице у клиента, так и на серверной части.

В компонент удаления можно повесить обработчик события клика на элемент - **click** (подробнее про обработчик события клика вы можете прочитать [здесь](#)).

```
<ui5-button icon="delete" (click)="editItem(data.id)"></ui5-button>
```

Функция должна иметь в качестве параметра **id** справки, которую необходимо удалить.

При нажатии на кнопку мы сможем при помощи обработчика **click**, вызывать функцию, которую мы инициализируем в Angular компоненте:

```
export class AppComponent implements OnInit {

  constructor(private httpService: HttpService){}

  ....
  removeItem(id) {
    this.httpService.deleteData(id).subscribe();
    this.DataTable[0].Contracts = this.DataTable[0].Contracts.filter(item =>
item.id !== id);
  }
  ....
}
```

Приняли с шаблона **html** идентификатор определённого элемента, отправили его в сервер, по средствам сервиса **http**, сервер удаляет эту справку у себя, далее мы тоже должны произвести удаление на клиентской стороне. Пробегаемся по всем элементам в списке **DataTable**, при помощи функции **filter** находим элемент, который имеет идентичный идентификатор и метод **filter** сам удалит его. Обновляем список и удаление было успешно проинициализированно!

В следующей главе будет расписано про редактирование определённой справки.

5.2.2.3 Разработка функционала редактирования справки

Чтобы отредактировать определённую справку в таблице, нам необходимо кликнуть на кнопку редактирования и произвести удаление, как в таблице у клиента, так и на серверной части.

При нажатии на кнопку мы должны открыть диалоговое окно. К диалоговому окну присваиваем идентификатор (`<ui5-dialog #dialog></ui5-dialog>`), благодаря которому, мы сможем использовать объект данного компонента в Angular компоненте:

```
@ViewChild('dialog', { static: false }) Dialog: ElementRef;
```

Подробнее про связку шаблонных переменных находится по данной [ссылке](#).

Имея переменную **Dialog**, мы сможем использовать определенные функции диалогового компонента (открытие, закрытие и т.д.).

Так же в диалоговом окне находятся формы ввода и форма с панелью выбора даты. Свяжем их с Angular компонентом:

- html шаблон:

```
<ui5-input #serInput placeholder="Введите серию .." style="width: 100%"></ui5-input>

<ui5-input #noInput id="no" placeholder="Введите номер .." style="width: 100%"
required></ui5-input>

<ui5-datepicker #dateInput style="width: 100%" format-pattern='short'></ui5-
datepicker>
```

- Angular компонент:

```
@ViewChild('serInput', { static: false }) serInput: ElementRef;
@ViewChild('noInput', { static: false }) noInput: ElementRef;
@ViewChild('dateInput', { static: false }) dateInput: ElementRef;
```

Теперь необходимо кнопке редактирования добавить обработчик события и вызывать функцию, в которую мы передадим идентификатор определенного элемента:

```
<ui5-button icon="edit" (click)="editItem(data.id)"></ui5-button>
```

Далее в Angular компоненте пропишем функцию `editItem()`:

```

export class AppComponent implements OnInit {
  oItemToEdit: any

  constructor(private httpService: HttpService){}

  ....

  editItem(id) {
    let dataEditTable = this.DataTable[0].Contracts;

    this.ItemToEdit = dataEditTable.find((Item) => {
      return Item.id === id;
    });

    this.serInput.nativeElement.value = this.ItemToEdit.ser;
    this.noInput.nativeElement.value = this.ItemToEdit.no;
    this.dateInput.nativeElement.value = this.ItemToEdit.issueDate;

    this.Dialog.nativeElement.open();
  }

  ....
}

```

Приняли с шаблона **html** идентификатор определённого элемента. Пробегаемся по всем элементам в списке **DataTable**, при помощи функции **find** находим элемент, который имеет идентичный идентификатор, дальше метод **find** вернет нам нужный элемент. Заносим данные в формы ввода, которые находятся в объекте **ItemToEdit** и теперь в диалоговом окне, формы ввода будут заполнены теми данными, которые были выбраны определенной справкой.

5.2.2.4 Разработка функционала компонента с категориями прав

Создадим список, в котором будут лежать наши данные из сервера. Внутри **ngOnInit()** мы пропишем метод нашего **http** сервиса:

```

CategoriesList = [];
ngOnInit() {
  ....
  this.httpService.getDataCategories().subscribe(
    (data) => {
      this.CategoriesList.push(data);
    }
  );
  ....
}

```

Необходимо в таблице создать еще одну колонну, для того чтобы отображать в ней категории прав.

При помощи интерполяции выведем все категории прав в форме ввода "**категории прав**" (был выбран компонент - Multi Combo Box, подробнее можете прочитать [здесь](#)):

```
....
<ui5-multi-combobox #categInput placeholder="Введите категорию .." style="width:
100%">
  <ui5-mcb-item *ngFor="let dataCateg of CategoriesList[0]?.Categories"
text="{{dataCateg?.category}}"></ui5-mcb-item>
</ui5-multi-combobox>
....
```

Теперь в нашей форме ввода, в диалоговом окне, при нажатии на стрелку вниз, будут отображаться все категории прав, полученные с сервера.

Далее необходимо удалять и отображать данные, в форме ввода **категорий прав**, если человек закрыл диалоговое окно, то очистить форму ввода, если человек открыл диалоговое окно при редактировании справки, то в форме ввода, должны быть данные выбранной справки:

- Удаление из формы ввода **категорий прав** всех данных, при закрытии диалогового окна:

```
@ViewChild('categInput', { static: false }) categInput: ElementRef;
closeDialog() {
  ....
  for (let i = 0; i < this.categInput.nativeElement._state.items.length; i++) {
    this.categInput.nativeElement._state.items[i].removeAttribute("selected");
  }
  ....
}
```

Произвели шаблонную связку с компонентом "**ui5-multi-combobox**", далее при помощи цикла пробегаемся по каждому элементу нашего компонента и удаляем у них свойство **selected**.

- Добавление в форму ввода **категорий прав**, тех категорий, которые имеет справка:

```

.....
editItem(id) {
    .....
    let ItemCateg = this.ItemToEdit.category.split(',')
    let InputItemsCateg = this.categInput.nativeElement._state.items

    for (let i = 0; i < InputItemsCateg.length; i++) {
        for (let k = 0; k < ItemCateg.length; k++) {
            if (InputItemsCateg[i]._state.text == ItemCateg[k]) {
                InputItemsCateg[i].setAttribute("selected", "true");
            }
        }
    }
    .....
}
.....

```

Пробегаемся при помощи цикла, сначала, по каждому элементу нашего компонента, далее производим еще один цикл, в котором будем пробегаться по тем элементам, которые имеет справка. Сверяем совпадает ли элемент из компонента с элементом из справки, если да, то присваиваем элементу нашего компонента свойство **selected**.

Подробнее про работу с атрибутами (**removeAttribute**, **setAttribute** и т.д.), можете прочитать по данной [ссылке](#)

5.2.2.5 Разработка валидации полей ввода

Разберем валидацию для поля **Серия**. Создадим текстовый компонент **ui5-label**, под полем и обозначим в нем директиву **ngIf**:

```

<ui5-label>Серия:</ui5-label><br>
<ui5-input #serInput placeholder="Введите серию .." style="width: 100%"></ui5-input>
<ui5-label *ngIf="checkSerInput" style="color:red">*необходимо ввести серию!</ui5-label><br><br>

```

Далее при нажатии на кнопку сохранения или добавления в диалоговых окнах, мы должны проверить данное поле на то - заполнен он или нет. Если не заполнен, то отобразить сообщение **необходимо ввести серию!** (а также не производить инициализацию отправки данных на сервер):

```

checkSerInput = false;
saveDialog($event) {
    .....
    if (this.serInput.nativeElement.value == '') this.checkSerInput = true
    .....
}

```


Тем самым, если поле будет пустым при сохранении или добавлении, то отобразиться данное сообщение. Соответственно нам нужно скрыть сообщение, когда закрываем диалоговое окно:

```
closeDialog() {  
    ....  
    this.serInput.nativeElement.value = '';  
    ....  
}
```

6 Индивидуальное задание

Необходимо доработать функционал с редактированием справки:

- при нажатии на кнопку **"сохранить"**, отправить отредактированные данные на сервер;
- при нажатии на кнопку **"отмена"**, закрыть диалоговое окно и удалить все данные с полей;
- произвести валидацию со всеми полями.

Так же добавить кнопку поверх **"Добавить"** поверх таблицы, которая имеет следующий функционал:

- при нажатии на кнопку **"добавить"**, открывается диалоговое окно, в котором лежат пустые поля ввода (Серия, Номер и т.д.). Также кнопки **"отмена"** и **"добавить"**;
- при нажатии на кнопку **"добавить"** в диалоговом окне, отправить введенные пользователем данные на сервер;
- при нажатии на кнопку **"отмена"**, закрыть диалоговое окно и удалить все данные с полей;
- произвести валидацию со всеми полями.