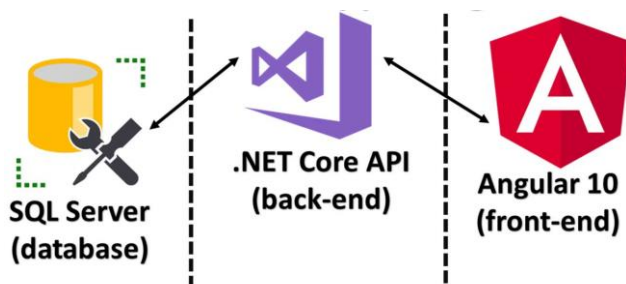


## Кратко о главном

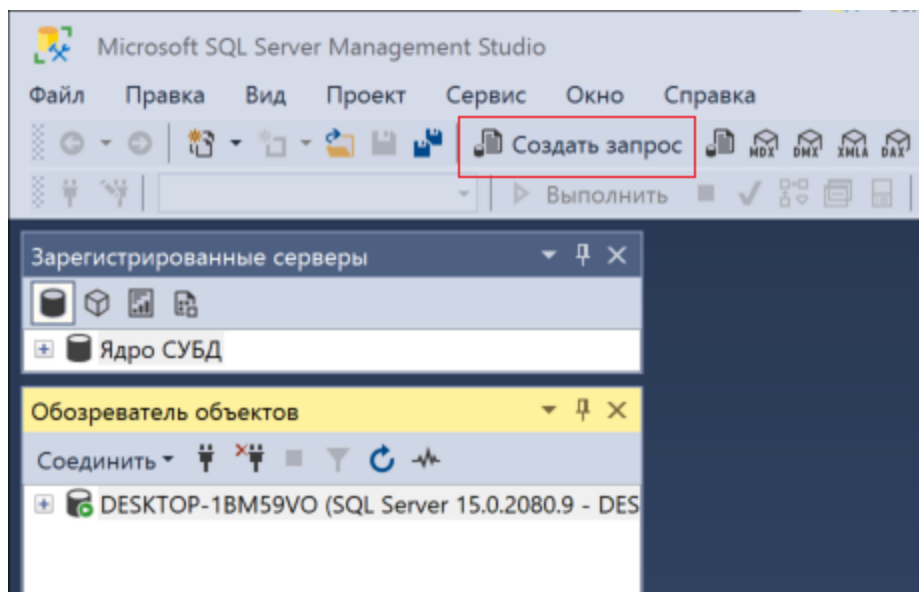
Для закрепления знаний создадим комплексный проект используя знания со всего курса. В ходе данной работы будет разработано WEB-приложение с нуля используя популярные технологии SQL Server для баз данных (Часть 1), .NET Core Web Api для создания backend (Часть 2) и Angular 10 для front-end (Часть 3).



### Часть 1. Создание базы данных ( Модели )

Начнём нашу работу из создания базы данных, которой пользоваться в ходе дальнейшей работы. Мы будем использовать MSSql и СУБД Microsoft SQL Server Management Studio (SSMS). Понятное дело, эти компоненты должны быть установлены на вашем ПК.

После запуска SSMS и подключения к серверу мы будем иметь подобную картину:



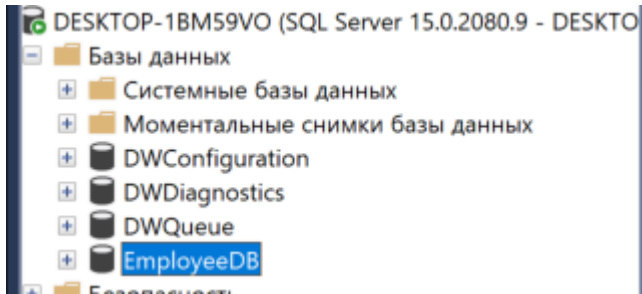
Создаём новый запрос, в котором пропишем:

```
create database EmployeeDB
```

Выполняем с мастера:



После чего увидим созданную базу в списке (если не видите, обновите список):



Создадим таблицы отделов и сотрудников:

```
create table dbo.Department(  
    DepartmentId int identity(1,1),  
    DepartmentName varchar(500)  
)
```

```
insert into dbo.Department values  
( 'IT' ),  
( 'Support' )
```

```
create table dbo.Employee(  
    EmployeeId int identity(1,1),  
    EmployeeName varchar(500),  
    Department varchar (500),  
    DateOfJoining date,  
    PhotoFileName varchar(500)  
)
```

```
insert into dbo.Employee values  
( 'Sam', 'IT', '2021-05-15', 'cat1.png' ),  
( 'Tom', 'Support', '2019-06-11', 'Tom.png' ),  
( 'Jerry', 'Support', '2019-06-11', 'Jerry.png' )
```

```
select * from dbo.Department;  
select * from dbo.Employee;
```

Выполняем запрос с созданной базы данных:



В результате имеем две таблицы Department и Employee, с которыми и будем работать в дальнейшем:

	DepartmentId	DepartmentName
1	1	IT
2	2	Support

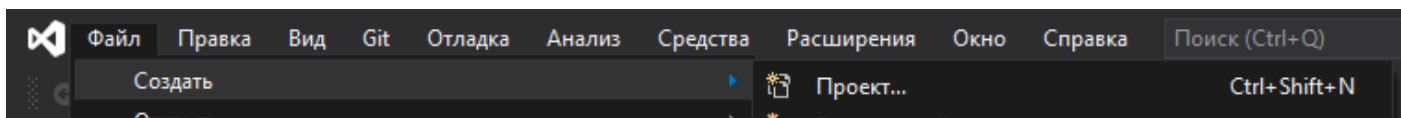
  

	EmployeeId	EmployeeName	Department	DateOfJoining	PhotoFileName
1	1	Sam	IT	2021-05-15	cat1.png
2	2	Tom	Support	2019-06-11	Tom.png
3	3	Jerry	Support	2019-06-11	Jerry.png

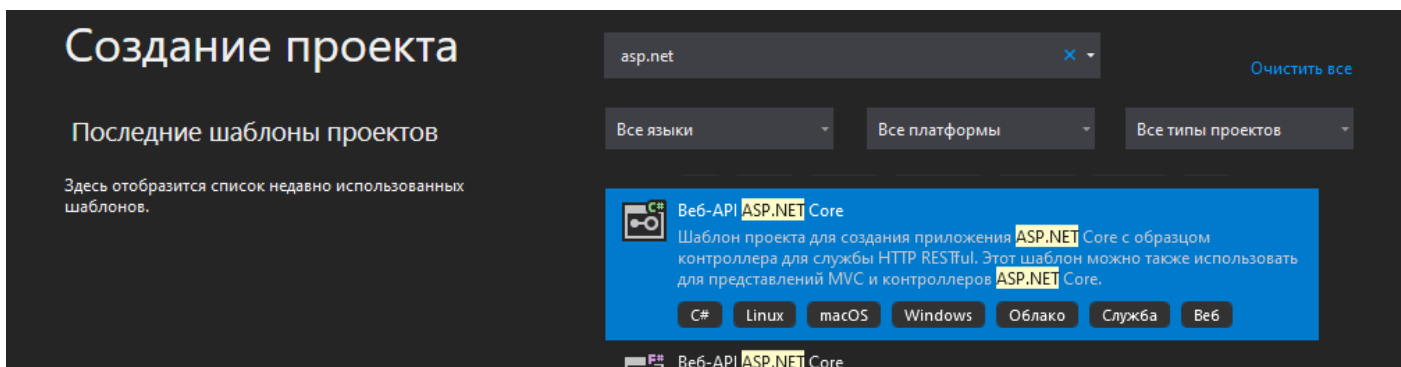
## Часть 2. Создание API ( Контроллер )

Создавать .NET Core Web Api будем в Microsoft Visual Studio 2019 (не менее, так как мы будем работать с .NET Core 3)(в других версиях процесс будет аналогичным, может немного отличаться интерфейс ):

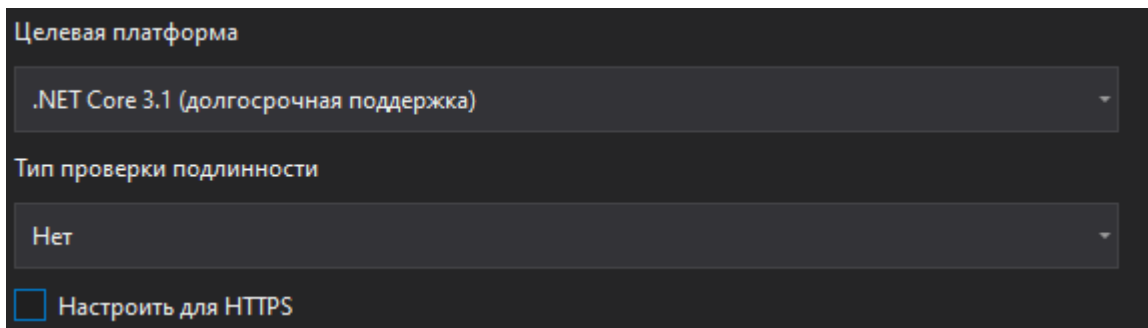
Создадим новый проект:



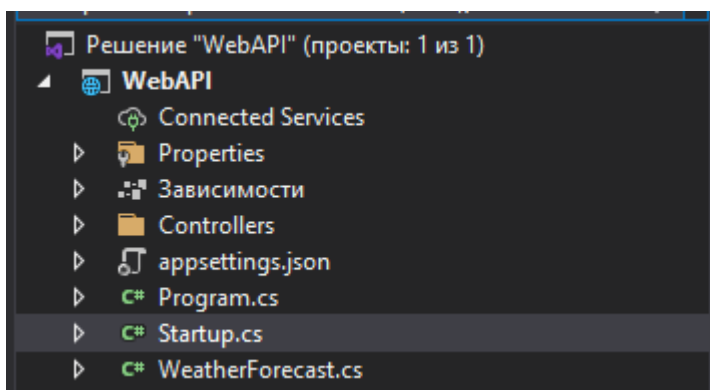
Выбираем веб-приложение ASP.Net Core (если такого выбора нету, то у Вас установлены не все нужные компоненты Microsoft Visual Studio):



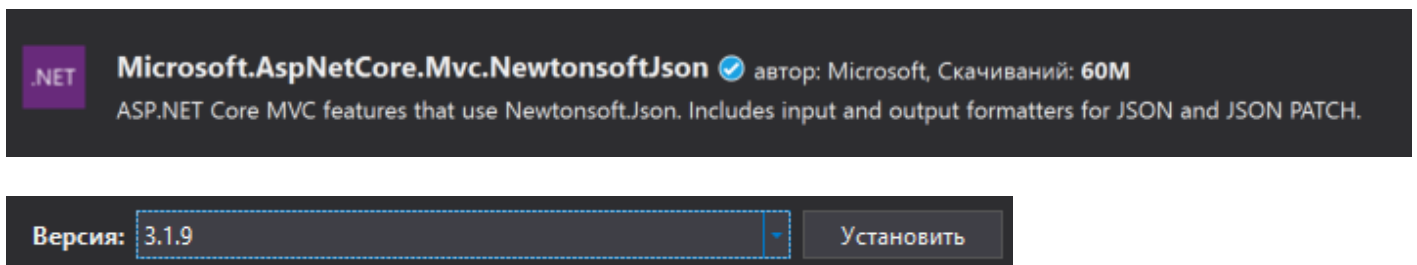
Дадим проекту имя «WebAPI», выбираем путь. В следующем диалоговом окне убираем HTTPS так как он просто не нужен в данной работе (топим за безопасность):



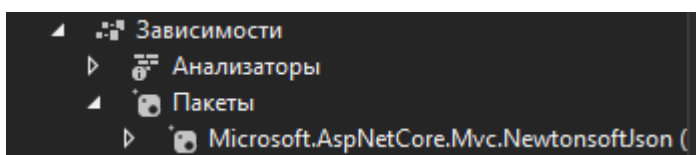
После создания проекта его структура будет выглядеть следующим образом:



Для начала создадим Json сериализацию по умолчанию. Для этого сначала установим новый NuGet пакет. Переходим в Проект → Управления пакетами NuGet. В открывшемся окне во вкладке «Обзор» вводим в поиске «mvc.newton» и с найденных вариантов выбираем и устанавливаем:



После установки пакет появится в зависимостях проекта:



Изменим класс Startup.cs, в котором производится конфигурация всех сервисов проекта. Подключим пространство имён:

```
using Newtonsoft.Json.Serialization;
```

Метод ConfigureServices изменим на:

```
public void ConfigureServices(IServiceCollection services)
{
    //enable CORS
    services.AddCors(c =>
    {
        c.AddPolicy("AllowOrigin", options =>
            options.AllowAnyOrigin()
                .AllowAnyMethod()
                .AllowAnyHeader());
    });
    //JSON Serializer
    services.AddControllersWithViews().AddNewtonsoftJson(options =>
        options.SerializerSettings.ReferenceLoopHandling =
        Newtonsoft.Json.ReferenceLoopHandling.Ignore)
        .AddNewtonsoftJson(options => options.SerializerSettings.ContractResolver
        = new DefaultContractResolver());

    services.AddControllers();
}
```

А метод Configure изменим на:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseCors(options =>
        options.AllowAnyOrigin()
            .AllowAnyMethod()
            .AllowAnyHeader()
        );
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();
```

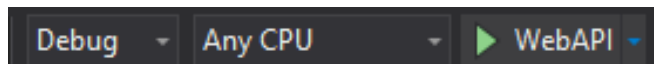
```

app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});
}

```

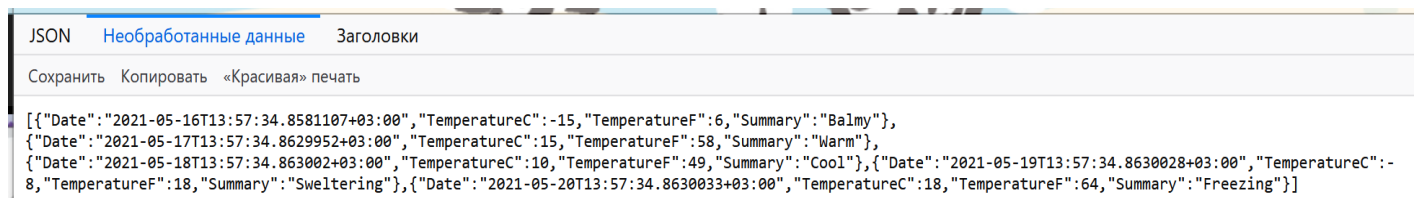
На данный момент проект должен успешно собраться и запуститься. По запуску приложения откроется консоль и страница в браузере:



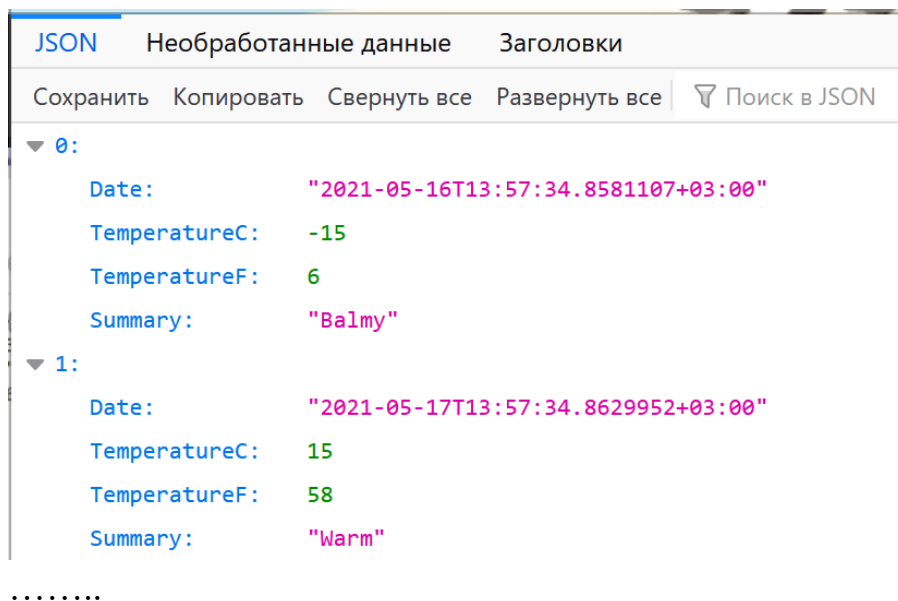
```

C:\> D:\For_university\TRPZ_4\final_project\WebAPI\WebAPI\bin\Debug\netcoreapp3.1\WebAPI.exe
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: D:\For_university\TRPZ_4\final_project\WebAPI\WebAPI

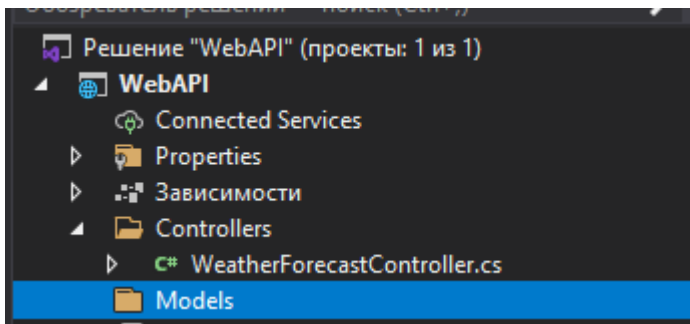
```



ИЛИ



Создадим модель необходимую для нашего приложения. Создадим папку для моделей Models (ПКМ по названию проекта → Добавить → Создать папку):



В неё добавим новый класс Department:

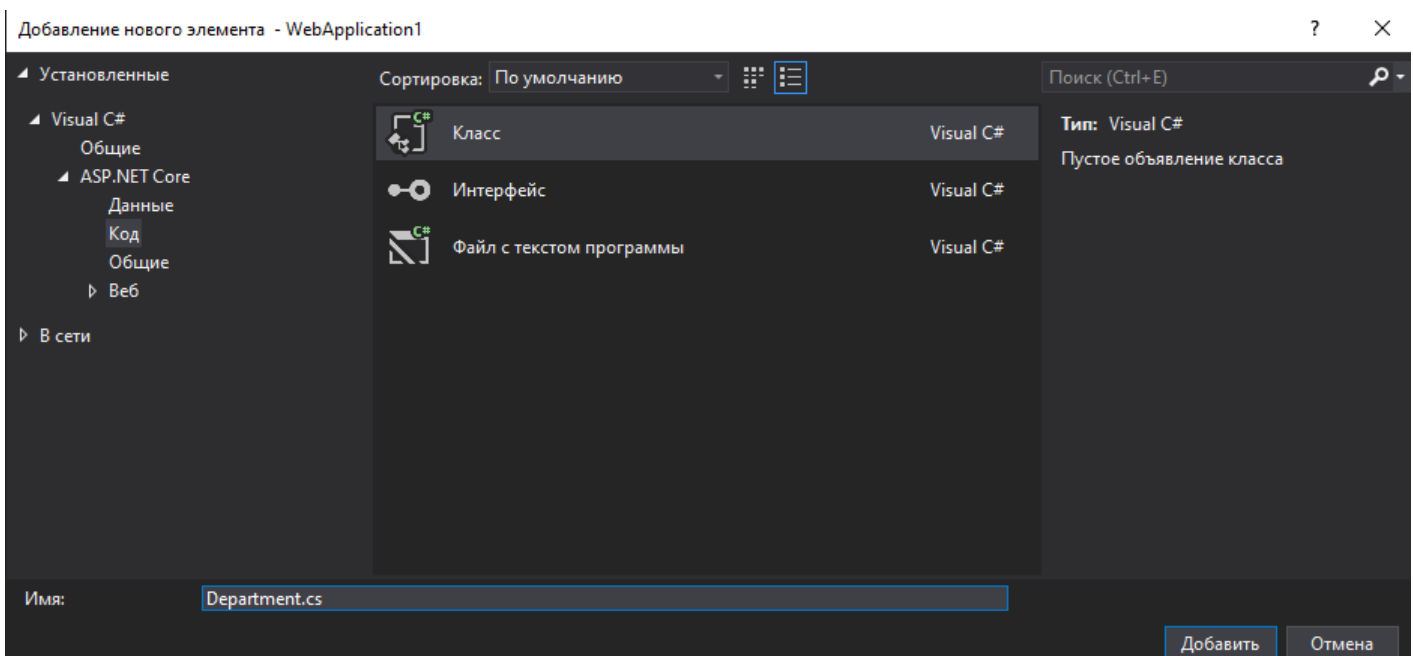


Таблица Department имеет две колонки DepartmentId и DepartmentName поэтому добавляем для них в класс свойства. Исходный код нового класса будет выглядеть:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication1.Models
{
    public class Department
    {
        public int DepartmentId { get; set; }
    }
}
```

```

        public string DepartmentName { get; set; }
    }
}

```

Аналогично создаём класс Employee. Исходный код нового класса будет выглядеть:

```

namespace WebApplication1.Models
{
    public class Employee
    {
        public int EmployeeId { get; set; }
        public string EmployeeName { get; set; }
        public string Department { get; set; }
        public string DateOfJoining { get; set; }
        public string PhotoFileName { get; set; }
    }
}

```

Теперь в файл appsettings.json добавим параметры подключения к базе данных:


```

{
  "ConnectionStrings": {
    "EmployeeAppCon": "Data Source=DESKTOP-1BM59VO;Initial Catalog=EmployeeDB;
Integrated Security=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}

```

В этом файле DESKTOP-1BM59VO имя SQL сервера. Название можно посмотреть и скопировать в SSMS.

Для работы с базой данных нам необходимо установить ещё один NuGet пакет:


**System.Data.SqlClient**

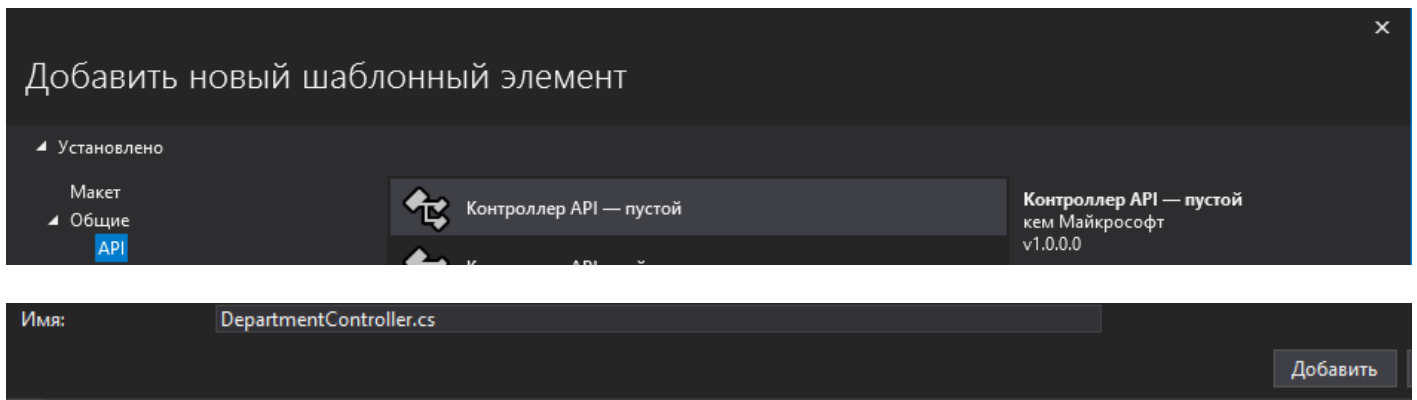
автор: Microsoft, Скачиваний: 207M

Provides the data provider for SQL Server. These classes provide access to versions of SQL Server and encapsulate database-specific protocols, including tabular data stream (TDS)

v4.8.2



Теперь добавим контроллер для таблицы Department (ПКМ по Controllers -> Добавить -> Контроллер):



В созданный файл DepartmentController .cs подключим пространства:

```
using Microsoft.Extensions.Configuration;
using System.Data.SqlClient;
using System.Data;
```

И в класс DepartmentController добавим метод для выборки всех строк из таблицы:

```
[HttpGet]
public JsonResult Get()
{
    string query = @"
        select DepartmentId, DepartmentName from dbo.Department";
    DataTable table = new DataTable();
    string sqlDataSource = 
_configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader); ;

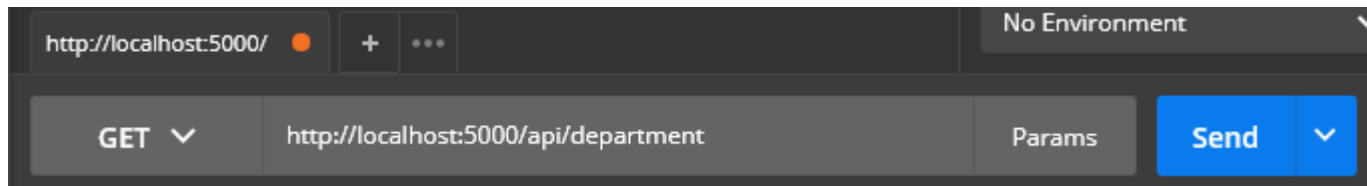
            myReader.Close();
            myCon.Close();
        }
    }
}
```

```

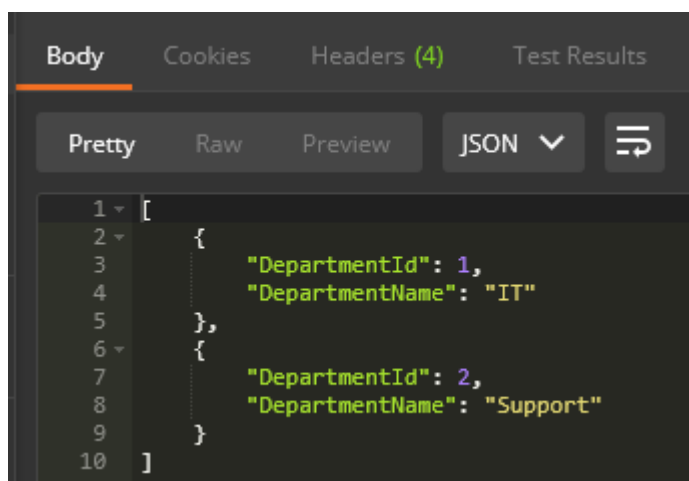
    return new JsonResult(table);
}

```

Соберём приложение и протестируем его работу в Postman:



Адрес можно скопировать в открытом окне консоли или браузера. В результате запроса получаем от API ответ:



Как видим запрос на API и в базу данных отработал правильно и мы получили данные с таблицы Department.

Если метод не работает, первым делом проверьте на каком порту запустилось API приложение.

Добавим метод для добавления данных в таблицу (подключив `using WebAPI.Models;`):

```

[HttpPost]
public JsonResult Post(Department dep)
{
    string query = @"
        insert into dbo.Department values
        ('"+dep.DepartmentName+"")
    ";
    DataTable table = new DataTable();
}

```

```

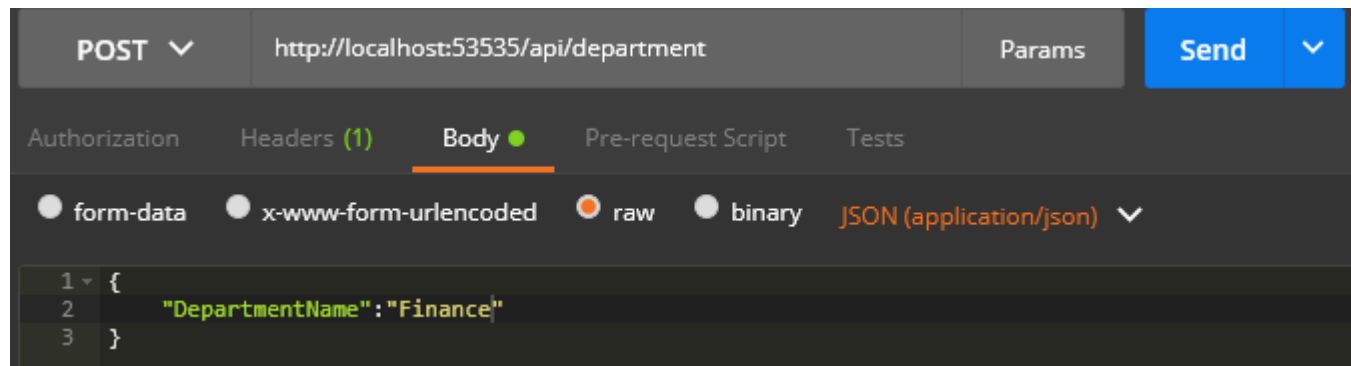
        string sqlDataSource =
_configuration.GetConnectionString("EmployeeAppCon");
        SqlDataReader myReader;
        using (SqlConnection myCon = new SqlConnection(sqlDataSource))
        {
            myCon.Open();
            using (SqlCommand myCommand = new SqlCommand(query, myCon))
            {
                myReader = myCommand.ExecuteReader();
                table.Load(myReader); ;

                myReader.Close();
                myCon.Close();
            }
        }

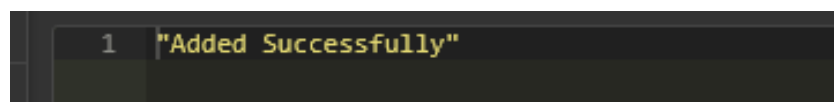
        return new JsonResult("Added Successfully");
    }

```

Тестируем работу метода в Postman:



Получаем ответ от API:



Как видим метод отработал и сообщил о успешном добавлении.

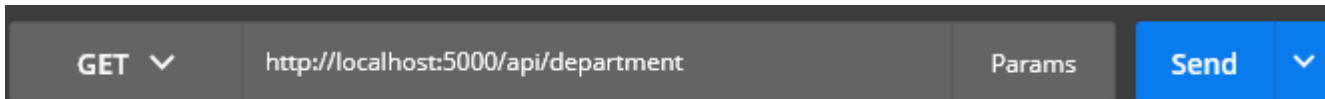
Выполним в SSMS:

```
select * from dbo.Department;
```

Получаем:

	DepartmentId	DepartmentName
1	1	IT
2	2	Support
3	3	Finance

Или прошлый запрос в Postman:



Ответ от API:

```

1  [
2    {
3      "DepartmentId": 1,
4      "DepartmentName": "IT"
5    },
6    {
7      "DepartmentId": 2,
8      "DepartmentName": "Support"
9    },
10   {
11     "DepartmentId": 3,
12     "DepartmentName": "Finance"
13   }
14 ]

```

Как видим метод полностью рабочий и добавляет новые записи в таблицу.

Добавим метод для добавления изменения данных в таблице:

```

[HttpPut]
public JsonResult Put(Department dep)
{
    string query = @"
        update dbo.Department set
        DepartmentName = '' + dep.DepartmentName + @''
        where DepartmentId = '' + dep.DepartmentId + @''
    ";
    DataTable table = new DataTable();
    string sqlDataSource =
_configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {

```

```

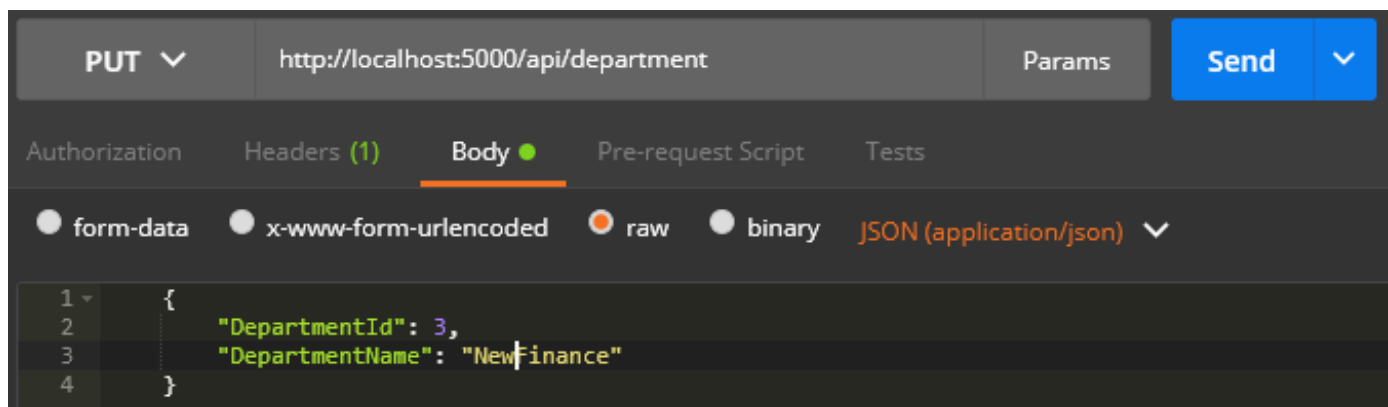
myCon.Open();
using (SqlCommand myCommand = new SqlCommand(query, myCon))
{
    myReader = myCommand.ExecuteReader();
    table.Load(myReader); ;

    myReader.Close();
    myCon.Close();
}

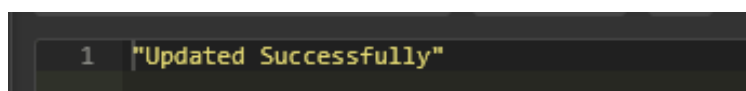
return new JsonResult("Updated Successfully");
}

```

Тестируем работу метода в Postman:



Ответ от API:



Как видим метод отработал и сообщил о успешном обновлении данных в таблице. Убедиться, что данные изменились, можно отправив запрос на выборку данных с Postman или сделать выборку прямо в SSMS.

И последний метод для удаления данных с таблицы:

```

[HttpDelete("{id}")]
public JsonResult Delete(int id)
{
    string query = @"
        delete from dbo.Department
        where DepartmentId = " + id + @"
    
```

```

        ";
        DataTable table = new DataTable();
        string sqlDataSource =
_configuration.GetConnectionString("EmployeeAppCon");
        SqlDataReader myReader;
        using (SqlConnection myCon = new SqlConnection(sqlDataSource))
        {
            myCon.Open();
            using (SqlCommand myCommand = new SqlCommand(query, myCon))
            {
                myReader = myCommand.ExecuteReader();
                table.Load(myReader); ;

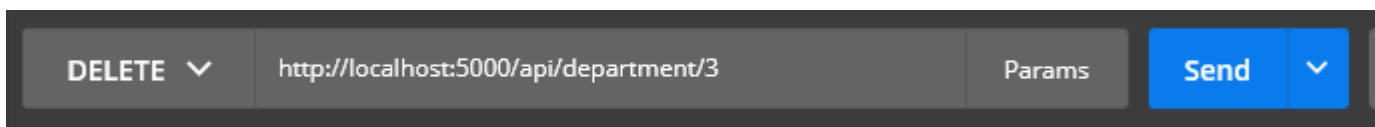
                myReader.Close();
                myCon.Close();
            }
        }

        return new JsonResult("Deleted Successfully");
    }

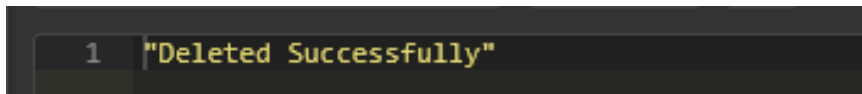
```

Тестируем работу метода в Postman. Данные в таблице до удаления:

	DepartmentId	DepartmentName
1	1	IT
2	2	Support
3	3	NewFinance



Ответ от API:



Как видим метод отработал и сообщил о успешном удалении строки из таблицы. Убедиться в это можно отправив запрос на выборку данных с Postman или сделать выборку прямо в SSMS.

Повторим все эти действия и создадим контролер с такими же 4-ма методами GET POST PUT DELETE для таблицы Employee. Для для этого создадим контроллер EmployeeController.cs и добавим в него следующий код:

```
private readonly IConfiguration _configuration;

public EmployeeController(IConfiguration configuration)
{
    _configuration = configuration;
}

[HttpGet]
public JsonResult Get()
{
    string query = @"
        select EmployeeId, EmployeeName, Department,
        convert(varchar(10),DateOfJoining,120) as DateOfJoining
        ,PhotoFileName
        from dbo.Employee
        ";
    DataTable table = new DataTable();
    string sqlDataSource =
_configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader); ;

            myReader.Close();
            myCon.Close();
        }
    }

    return new JsonResult(table);
}

[HttpPost]
public JsonResult Post(Employee emp)
```

```

{
    string query = @"
        insert into dbo.Employee
        (EmployeeName,Department,DateOfJoining,PhotoFileName)
        values
        (
            '" + emp.EmployeeName + @"'
            ,'" + emp.Department + @"'
            ,'" + emp.DateOfJoining + @"'
            ,'" + emp.PhotoFileName + @"'
        )
    ";

    DataTable table = new DataTable();
    string sqlDataSource =
_configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader); ;

            myReader.Close();
            myCon.Close();
        }
    }

    return new JsonResult("Added Successfully");
}

```

```

[HttpPut]
public JsonResult Put(Employee emp)
{
    string query = @"
        update dbo.Employee set
        EmployeeName = '" + emp.EmployeeName + @"'
        ,Department = '" + emp.Department + @"'
        ,DateOfJoining = '" + emp.DateOfJoining + @"'
        where EmployeeId = " + emp.EmployeeId + @"
    ";

    DataTable table = new DataTable();

```



```

        string sqlDataSource =
_configuration.GetConnectionString("EmployeeAppCon");
        SqlDataReader myReader;
        using (SqlConnection myCon = new SqlConnection(sqlDataSource))
        {
            myCon.Open();
            using (SqlCommand myCommand = new SqlCommand(query, myCon))
            {
                myReader = myCommand.ExecuteReader();
                table.Load(myReader); ;

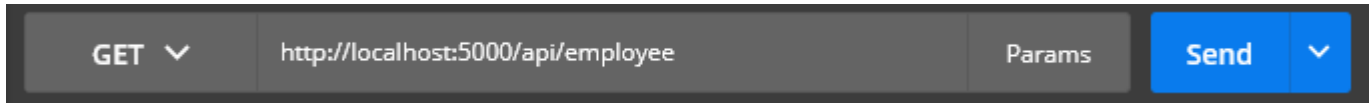
                myReader.Close();
                myCon.Close();
            }
        }

        return new JsonResult("Updated Successfully");
    }
    [HttpDelete("{id}")]
    public JsonResult Delete(int id)
    {
        string query = @"
            delete from dbo.Employee
            where EmployeeId = " + id + @"
        ";
        DataTable table = new DataTable();
        string sqlDataSource =
_configuration.GetConnectionString("EmployeeAppCon");
        SqlDataReader myReader;
        using (SqlConnection myCon = new SqlConnection(sqlDataSource))
        {
            myCon.Open();
            using (SqlCommand myCommand = new SqlCommand(query, myCon))
            {
                myReader = myCommand.ExecuteReader();
                table.Load(myReader); ;

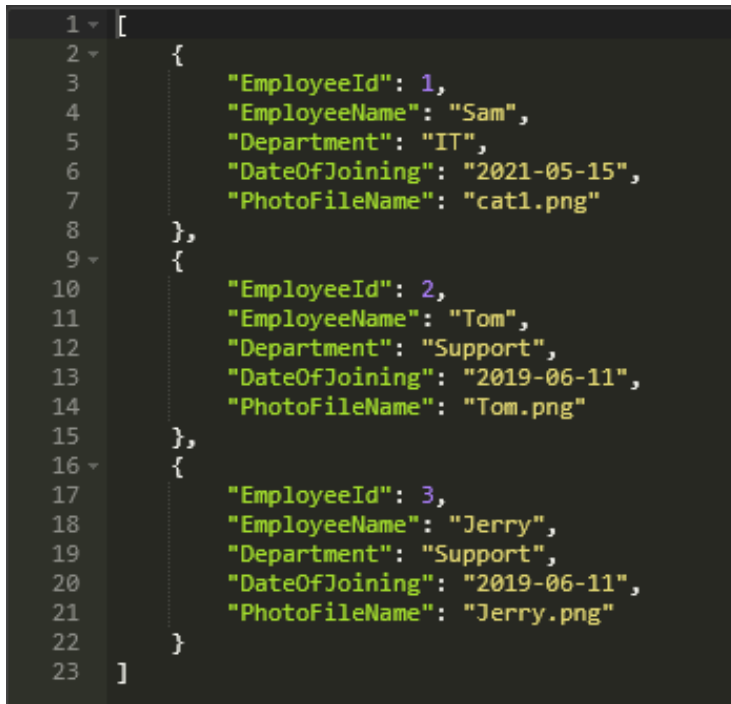
                myReader.Close();
                myCon.Close();
            }
        }
        return new JsonResult("Deleted Successfully");
    }
}

```

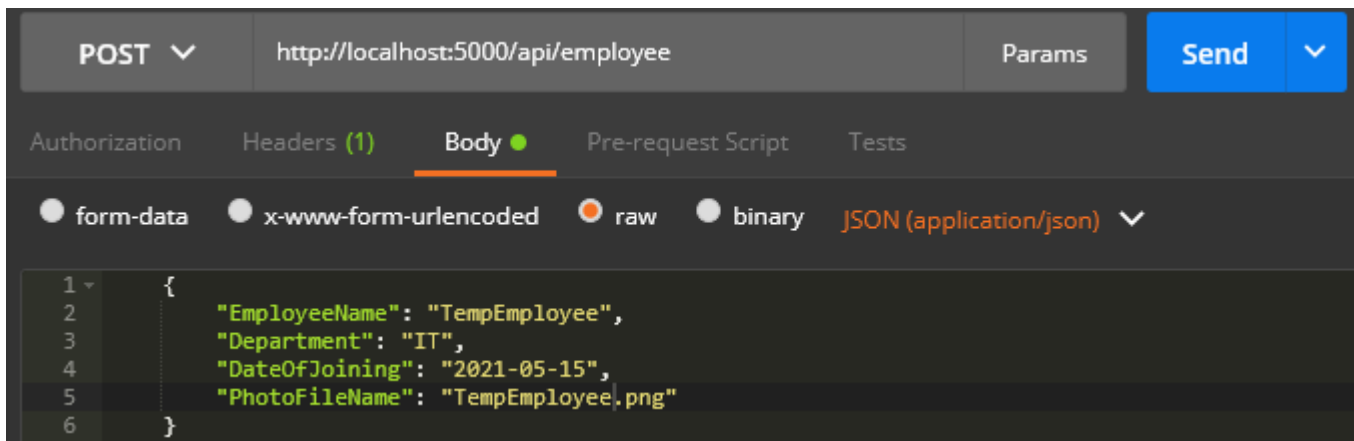
Проверим работу всех методов. Выборка данных GET-запрос:



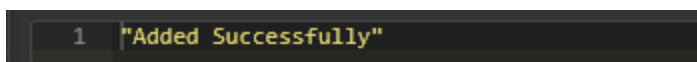
Ответ от API:



Добавление данных POST-запрос:



Ответ от API:



Обновлённые данные:

	EmployeeId	EmployeeName	Department	DateOfJoining	PhotoFileName
1	1	Sam	IT	2021-05-15	cat1.png
2	2	Tom	Support	2019-06-11	Tom.png
3	3	Jerry	Support	2019-06-11	Jerry.png
4	4	TempEmployee	IT	2021-05-15	TempEmployee.png

Обновление данных PUT-запрос:

PUT

http://localhost:5000/api/employee

Params

Send

Authorization

Headers (1)

Body

Pre-request Script

Tests

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

JSON (application/json)

```
1 {
2   "EmployeeId": 4,
3   "EmployeeName": "Vasya",
4   "Department": "IT",
5   "DateOfJoining": "2021-05-15",
6   "PhotoFileName": "Vasya.png"
7 }
```

Ответ от API:

```
1 "Updated Successfully"
```

Обновлённые данные:

	EmployeeId	EmployeeName	Department	DateOfJoining	PhotoFileName
1	1	Sam	IT	2021-05-15	cat1.png
2	2	Tom	Support	2019-06-11	Tom.png
3	3	Jerry	Support	2019-06-11	Jerry.png
4	4	Vasya	IT	2021-05-15	TempEmployee.png

Удаление данных DELETE-запрос:

DELETE

http://localhost:5000/api/employee/4

Params

Send

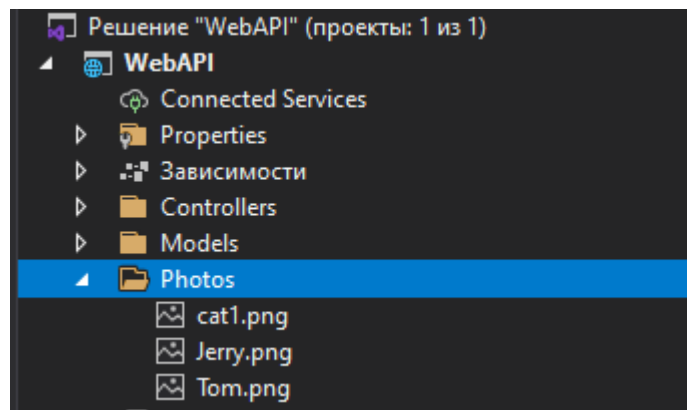
Ответ от API:

```
1 "Deleted Successfully"
```

Обновлённые данные:

	EmployeeId	EmployeeName	Department	DateOfJoining	PhotoFileName
1	1	Sam	IT	2021-05-15	cat1.png
2	2	Tom	Support	2019-06-11	Tom.png
3	3	Jerry	Support	2019-06-11	Jerry.png

Теперь позаботимся про возможность загружать файлы фото. Для начала в проекте создадим папку Photos. В таблице Employee уже есть три записи с именами картинок. поэтому в созданную папку добавим картинки cat1.png Tom.png Jerry.png.



Для того, чтобы пользоваться новой папкой в приложения, мы должны добавить инструкции в Startup.cs. Подключаем:

```
using System.IO;
using Microsoft.Extensions.FileProviders;
В конец метода Configure добавим:
app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new PhysicalFileProvider(
        Path.Combine(Directory.GetCurrentDirectory(), "Photos")),
    RequestPath = "/Photos"
});
```

В контролер EmployeeController подключаем:

```
using System.IO;
using Microsoft.AspNetCore.Hosting;
```

В начало класса добавим зависимость, которая даст возможность использовать пути папок и файлов в проекта:

```
private readonly IWebHostEnvironment _env;
```

Метод EmployeeController изменим на :

```
public EmployeeController(IConfiguration configuration, IWebHostEnvironment env)
{
    _configuration = configuration;
    _env = env;
}
```

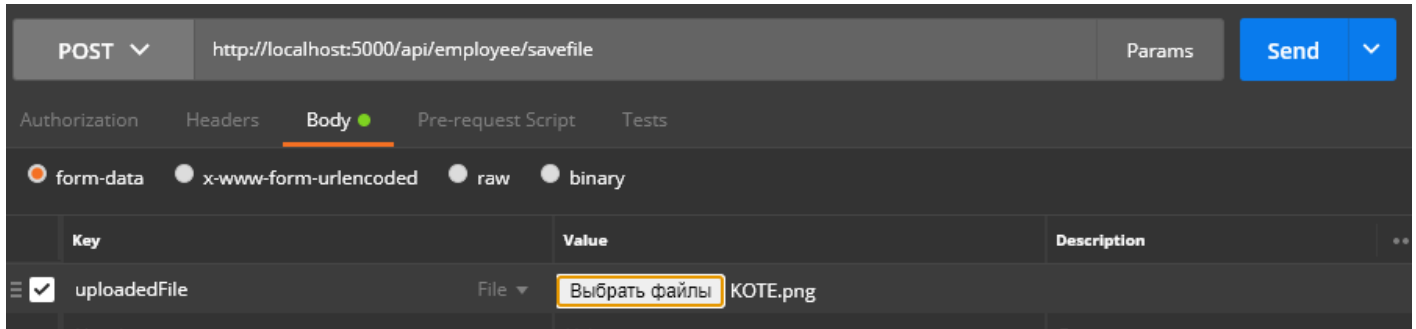
И добавим новый метод, который будет загружать новые файлы:

```
[Route("SaveFile")]
[HttpPost]
public JsonResult SaveFile()
{
    try
    {
        var httpRequest = Request.Form;
        var postedFile = httpRequest.Files[0];
        string filename = postedFile.FileName;
        var physicalPath = _env.ContentRootPath + "/Photos/" + filename;

        using(var stream = new FileStream(physicalPath, FileMode.Create))
        {
            postedFile.CopyTo(stream);
        }

        return new JsonResult(filename);
    }
    catch (Exception)
    {
        return new JsonResult("cat1.png");
    }
}
```

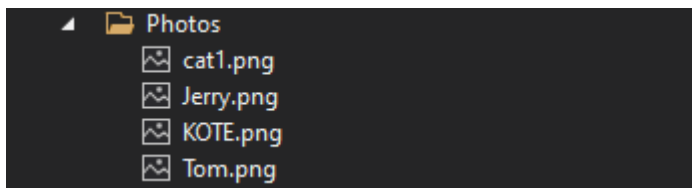
Проверим работу нового метода в Postman:



Ответ от API:



API вернул имя файла, значит файл был успешно загружен и появился в папке с изображениями:



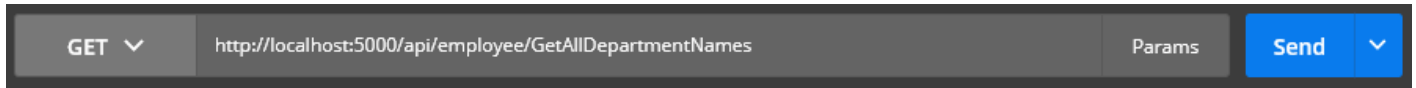
И последний метод, который нужно добавить в контролер, это метод для выборки названий всех отделов для создания выпадающих списков (к этому мы ещё дойдём):

```
[Route("GetAllDepartmentNames")]
public JsonResult GetAllDepartmentNames()
{
    string query = @"
        select DepartmentName from dbo.Department
    ";
    DataTable table = new DataTable();
    string sqlDataSource =
        _configuration.GetConnectionString("EmployeeAppCon");
    SqlDataReader myReader;
    using (SqlConnection myCon = new SqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (SqlCommand myCommand = new SqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
        }
    }
}
```

```
        table.Load(myReader); ;

        myReader.Close();
        myCon.Close();
    }
}
return new JsonResult(table);
}
```

Проверим работу нового метода в Postman:



Ответ от API:

```
1 [
2   {
3     "DepartmentName": "IT"
4   },
5   {
6     "DepartmentName": "Support"
7   }
8 ]
```

Как видим всё работает верно.

На этом этапе разработка Веб-приложения завершена. В следящей части мы начнём Front-End разработку используя Angular.

### Часть 3. Создание Веб-страницы ( Представление )

Для использования Angular на ПК должен быть установлен Node.js. Скачать и установить можно на оф сайте <https://nodejs.org/uk/>.

После того как пакетный менеджер npm установлен, переходим к установке Angular. В командной строке исполняем команду (установка займёт какое-то время):

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

Для создания проекта можно использовать хоть обычный блокнот, но для удобства будем использовать лёгкий и быстрый редактор кода Visual Studio Code, который уже преднастроен для нашей работы и понимает различные нужные нам синтаксисы «с коробки».

Откроем директорию, в которой будем создавать фронт (в примере это D:\For\_university\TRPZ\_4\final\_project, но это не так и важно). Далее будем работать с терминала самого редактора. Для создания нового проекта Angular в терминале пропишем команду:

```
ng new Frontend
```

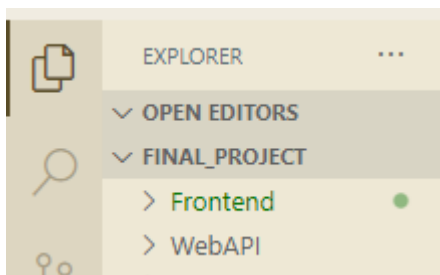
, где Frontend – название проекта

Сразу будет задано несколько вопросов. Не соглашаемся на строгую типизацию данных и добавления роутинга в проект. Роутинг в данной работе не понадобится, но его можно потом подключить вручную. Но если включить строгую типизацию, проект не соберётся. Мы не будем использовать css-препроцессоры, поэтому выбираем чистый css. В итоге имеем:

```
D:\For_university\TRPZ_4\final_project>ng new Frontend
? Do you want to enforce stricter type checking and stricter bundle budgets in the workspace?
  This setting helps improve maintainability and catch bugs ahead of time.
  For more information, see https://angular.io/strict No
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
```



После начнётся процесс создания. По завершению будет создана папка с названием проекта:

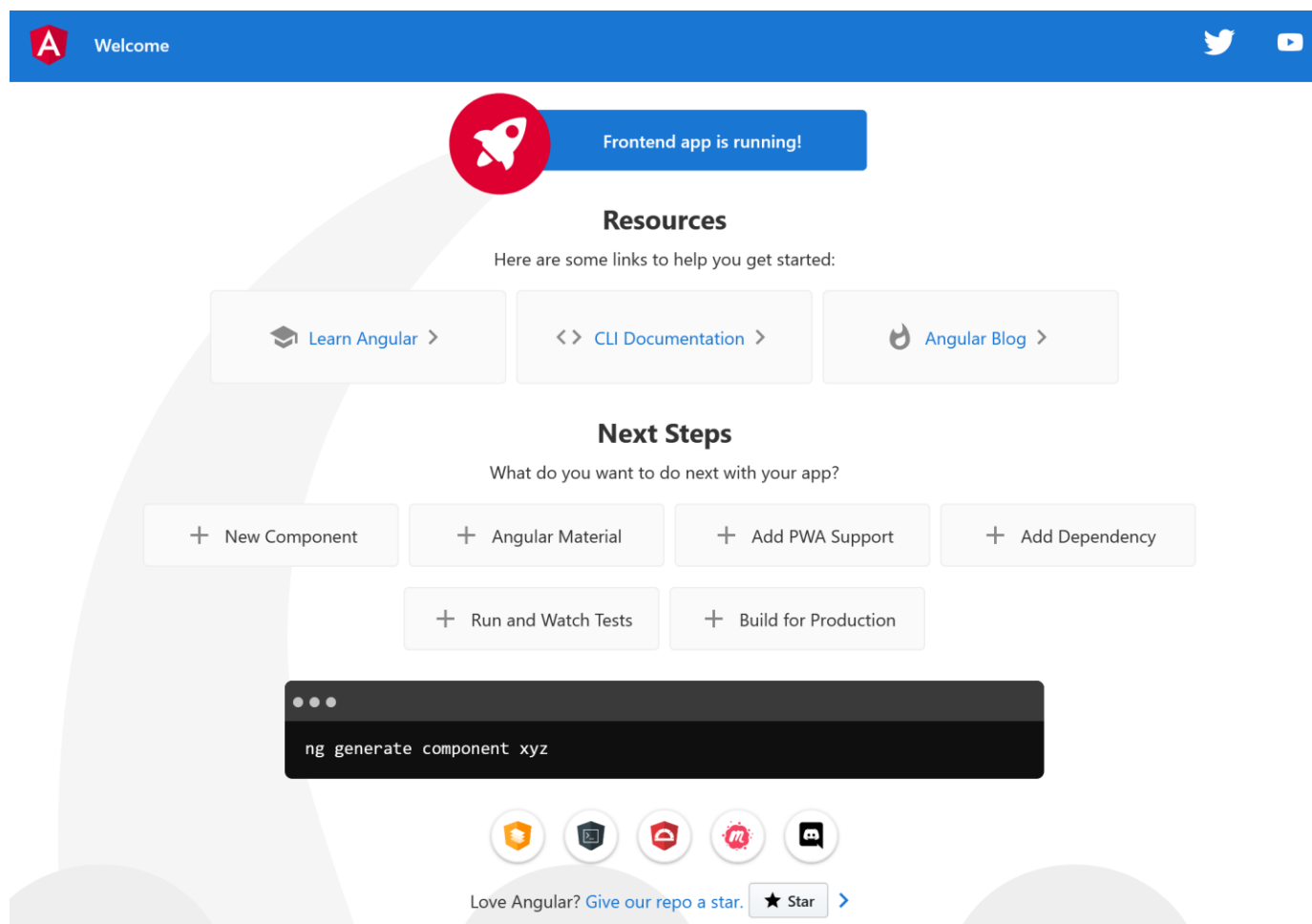


Перейдём в эту папку:

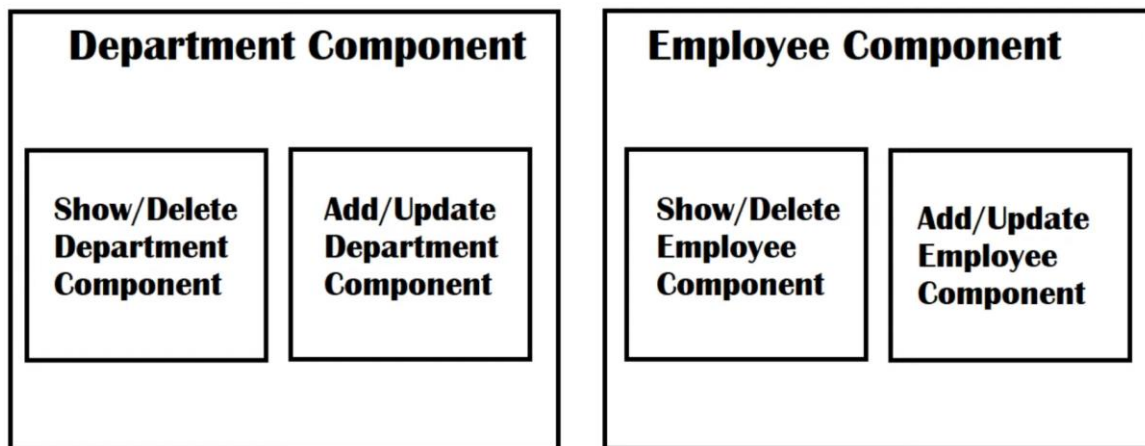
```
cd Frontend
```

Убедиться, что всё установлено верно, можно запустив макет созданного проекта:

```
ng serve --open
```

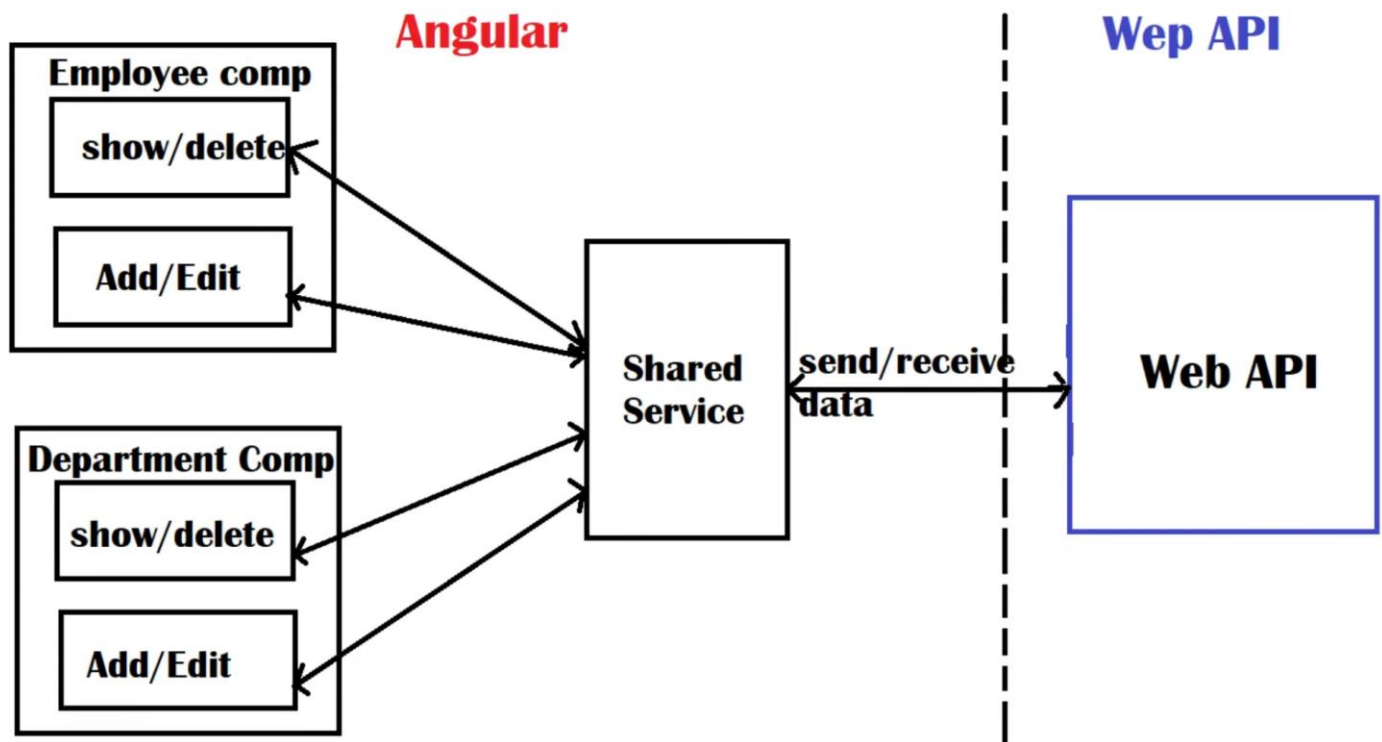


Для нашего проекта нужно создать 2 базовых компонента для работы с каждой таблицей:



Каждый компонент будет иметь по два дочерних компонента. Первый для отображения и удаления данных таблицы и второй для добавления новых данных и их редактирования.

Общая структура нашего Angular проекта будет выглядеть так:



Созданные компоненты будут отправлять и получать данные с API нашего приложения через так названный Shared Service.

Приступим к созданию новых компонентов и сервисов. В командной строке выполним команду (разумеется из папки Angular проекта):

```
ng generate component department
```

Можем видеть, что компонент `department` был создан в директории `src/app`.

Аналогично создаём дочерние компоненты:

```
ng generate component department/show-dep
ng generate component department/add-edit-dep
```

Повторяем действия для создания компонента `employee`:

```
ng generate component employee
ng generate component employee/show-emp
ng generate component employee/add-edit-emp
```

Создаём сервис:

```
ng generate service shared
```

В `src\app\app.module.ts` можем увидеть все созданные компоненты :

```
@NgModule({
  declarations: [
    AppComponent,
    DepartmentComponent,
    ShowDepComponent,
    AddEditDepComponent,
    EmployeeComponent,
    ShowEmpComponent,
    AddEditEmpComponent
  ],
```

В этот же файл мы должны подключить созданный сервис:

```
import { SharedService } from './shared.service';
```

И добавляем его имя в `Providers`:

```
@NgModule({
  declarations: [
    AppComponent,
    DepartmentComponent,
```

```

    ShowDepComponent,
    AddEditDepComponent,
    EmployeeComponent,
    ShowEmpComponent,
    AddEditEmpComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [SharedService],
  bootstrap: [AppComponent]
}))

```

Добавим в `src\app\shared.service.ts` методы, которые будут общаться с API:

```

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

@Injectable({
  providedIn: 'root'
})
export class SharedService {
  readonly APIUrl="http://localhost:5000/api";
  readonly PhotoUrl = "http://localhost:5000/Photos/";

  constructor(private http:HttpClient) { }

  getDepList():Observable<any[]>{
    return this.http.get<any>(this.APIUrl+'/department');
  }

  addDepartment(val:any){
    return this.http.post(this.APIUrl+'/Department',val);
  }

  updateDepartment(val:any){
    return this.http.put(this.APIUrl+'/Department',val);
  }

  deleteDepartment(val:any){
    return this.http.delete(this.APIUrl+'/Department/'+val);
  }
}

```

```

getEmpList():Observable<any[]>{
    return this.http.get<any>(this.apiUrl+'/Employee');
}

addEmployee(val:any){
    return this.http.post(this.apiUrl+'/Employee',val);
}

updateEmployee(val:any){
    return this.http.put(this.apiUrl+'/Employee',val);
}

deleteEmployee(val:any){
    return this.http.delete(this.apiUrl+'/Employee/'+val);
}

UploadPhoto(val:any){
    return this.http.post(this.apiUrl+'/Employee/SaveFile',val);
}

getAllDepartmentNames():Observable<any[]>{
    return this.http.get<any[]>(this.apiUrl+'/Employee/GetAllDepartmentNames');
}
}

```

Далее сделаем роутинг по нашей странице. Роутинг – возможность перемещаться по странице, изменяя ссылку страницы. Так по дефолту страница запускается по адресу localhost:4200. Роутинг позволит нам вводя localhost:4200/department или localhost:4200/employee перемещаться соответственно на страницу с таблицей department и employee.

Для добавления роутинга в проект выполним в командной строке:

```
ng generate module app-routing --flat --module=app
```

Содержимое только что создавшегося файла src\app\app-routing.module.ts изменим на:

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import {EmployeeComponent} from './employee/employee.component';
import {DepartmentComponent} from './department/department.component';

```

```
const routes: Routes = [
  {path: 'employee', component: EmployeeComponent},
  {path: 'department', component: DepartmentComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Содержимое файла src/app/app.component.html изменим на:

```
<h2> Routing test</h2>
<router-outlet></router-outlet>
```

Запустим сервер уже известной для нас командой:

```
ng serve --open
```



## Routing test



## Routing test

department works!



## Routing test

employee works!

То есть по дополнении ссылки на место тега `<router-outlet>` подставляется содержания разметки компонентов

`src\app\department\department.component.html`

или же

`src\app\employee\employee.component.html`

Содержимое файла `src\app\app.component.html` снова изменим на:

```
<div class="container">
  <h5 class="d-flex justify-content-
center">Angular 10, Web API, SQL Server App Lesson </h5>
  <nav class="navbar navbar-expand-sm bg-light navbar-dark">
    <ul class="navbar-nav">
      <li class="nav-item">
        <button routerLink="department"
        class="m-1 btn btn-light btn-outline-
primary" Button>Departments</button>
      </li>

      <li class="nav-item">
        <button routerLink="employee"
        class="m-1 btn btn-light btn-outline-
primary" Button>Employees</button>
      </li>
    </ul>
  </nav>

  <router-outlet></router-outlet>
</div>
```

В результате на странице появятся 2 кнопки, используя которые можно перемещаться между таблицами:

---

Angular 10, Web API, SQL Server App Lesson

Departments

Employees

employee works!

По дефолту Bootstrap уже подключённый в проект Angular. Но если страница имеет вид:

#### Angular 10, Web API, SQL Server App Lesson

- Departments
- Employees

то он не подключён и это придётся сделать руками. Для этого в файл `src\index.html` в начало тега `<head>` подключаем стили Bootstrap-a:

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css" integrity="sha384-B0vP5xmATw1+K9KRQJQERJvTumQW0nPEzvF6L/Z6nronJ3oU0FUFPcJEUQouq2+1" crossorigin="anonymous">
```

А в самый конец тега `<body>` нужные для его работы скрипты:

```
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/reFTGAW83EW2RDu2S0VKAiZap3H66lZHN81PoYlFhbGU+6BZp6G7niu735Sk7lN" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/js/bootstrap.min.js" integrity="sha384-+YQ4JLhgyBLPDQt//I+STsc9iw4uQqACwlvpslubQzn4u2UU2UFM80nGisd026JF" crossorigin="anonymous"></script>
```

После этого страница примет вид, как было обозначено ранее.

Теперь создадим таблицу, которая будет отображать данные таблицы Department из нашей базы данных. Первым делом импортируем в `src\app\app.module.ts` модули для работы с Http клиентом. И сразу же подключим модули для форм, чтобы не возвращаться сюда позже:

```
import {HttpClientModule} from '@angular/common/http';
import {FormsModule, ReactiveFormsModule} from '@angular/forms';
```



И не забываем добавить имена модулей в список `imports`:

```
imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule,
  FormsModule,
  ReactiveFormsModule
],
```

Заменим содержание `src/app/department/show-dep/show-dep.component.html` на:

```
import { Component, OnInit } from '@angular/core';
import { SharedService } from 'src/app/shared.service';

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

  constructor(private service: SharedService) { }

  DepartmentList: any = [];

  ngOnInit(): void {
    this.refreshDepList();
  }

  refreshDepList(){
    this.service.getDepList().subscribe(data=>{
      this.DepartmentList=data;
    });
  }
}
```

В `DepartmentList` хранятся записи с таблицы с базы данных. Они получаются в методе `refreshDepList`, в котором и происходит GET запрос на API. Метод

ngOnInit выполняется самым первым при отрисовке данного компонента, то есть когда мы нажимаем на соответствующую кнопку.

Заменим содержание `src\app\department\show-dep\show-dep.component.html` на:

```
<table class="table table-striped">
  <thead>
    <tr>
      <th> DepartmentId</th>
      <th>Department Name</th>
      <th>Options</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let dataItem of DepartmentList">
      <td>{{dataItem.DepartmentId}}</td>
      <td>{{dataItem.DepartmentName}}</td>
      <td>
        <button type="button" class="btn btn-light mr-1">
          Edit
        </button>
        <button type="button" class="btn btn-light mr-
1"
          >
          Delete
        </button>
      </td>
    </tr>
  </tbody>
</table>
```

А в файл `src\app\department\department.component.html` пропишем тег – селектор компонента `show-dep`:

```
<app-show-dep></app-show-dep>
```

Теперь страница приобретёт вид (помним, что API должен быть запущен!):

## Angular 10, Web API, SQL Server App Lesson

<div>Departments</div> <div>Employees</div>		
DepartmentId	Department Name	Options
1	IT	<div>Edit</div> <div>Delete</div>
2	Support	<div>Edit</div> <div>Delete</div>

Видим, что в таблицу загружаются все записи из таблицы из базы данных. Для каждой записи есть кнопки для опции редактирования и удаления. На данный момент они не исполняют никакого функционала. Этот функционал будет реализован позже.

Создадим модальное окно, которое будет открываться для добавления новых данных или редактирования существующих.

Дополним класс в `src\app\department\show-dep\show-dep.component.ts`:

```
ModalTitle:string;
ActivateAddEditDepComp:boolean=false;
dep:any;

addClick(){
  this.dep={
    DepartmentId:0,
    DepartmentName:""
  }
  this.ModalTitle="Add Department";
  this.ActivateAddEditDepComp=true;
}

closeClick(){
  this.ActivateAddEditDepComp=false;
  this.refreshDepList();
}

editClick(item){
```

```

    this.dep=item;
    this.ModalTitle="Edit Department";
    this.ActivateAddEditDepComp=true;
  }

  deleteClick(item){
    console.log('Will be done soon');
  }

```

А в начало `src\app\department\show-dep\show-dep.component.html` добавим:

```

<button type="button" class="btn btn-primary float-right m-2"
data-toggle="modal" data-target="#exampleModal"
(click)="addClick()"
data-backdrop="static" data-keyboard="false">
  Add Department
</button>

<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" role="dialog" aria-
labelledby="exampleModalLabel" aria-hidden="true">
  <div class="modal-dialog modal-dialog-centered modal-xl" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">{{ModalTitle}}</h5>
        <button type="button" class="close"
data-dismiss="modal" aria-label="Close"
(click)="closeClick()" >
          <span aria-hidden="true">&times;</span></button>
      </div>
      <div class="modal-body">
        <app-add-edit-dep [dep]="dep" *ngIf="ActivateAddEditDepComp">
        </app-add-edit-dep>
      </div>
    </div>
  </div>
</div>
</div>

```

А кнопки в таблице `src\app\department\show-dep\show-dep.component.html` заменим на иконки и пропишем исполняемые по нажатию методы:

```
<button type="button" class="btn btn-light mr-1"
      data-toggle="modal" data-target="#exampleModal"
      (click)="editClick(dataItem)"
      data-backdrop="static" data-keyboard="false">
  <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-pencil-square" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
    <path d="M15.502 1.94a.5.5 0 0 1 0 .706L14.459 3.691-2-2L13.502.646a.5.5 0 0 1 .707 0l1.293 1.293zm-1.75 2.456l-2-2L4.939 9.21a.5.5 0 0 0-.121.196l-.805 2.414a.25.25 0 0 0 .316.316l2.414-.805a.5.5 0 0 0 .196-.121l6.813-6.814z"/>
    <path fill-rule="evenodd" d="M1 13.5A1.5 1.5 0 0 0 2.5 15h11a1.5 1.5 0 0 0 1.5-1.5v-6a.5.5 0 0 0-1 0v6a.5.5 0 0 1-.5.5h-11a.5.5 0 0 1-.5-.5v-11a.5.5 0 0 1 .5-.5H9a.5.5 0 0 0 0-1H2.5A1.5 1.5 0 0 0 1 2.5v11z"/>
  </svg>
</button>
<button type="button" class="btn btn-light mr-1"
      (click)="deleteClick(dataItem)">
  <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-trash-fill" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
    <path fill-rule="evenodd" d="M2.5 1a1 1 0 0 0-1 1v1a1 1 0 0 0 1 1H3v9a2 2 0 0 0 2 2h6a2 2 0 0 0 2-2V4h.5a1 1 0 0 0 1-1V2a1 1 0 0 0-1-1H10a1 1 0 0 0-1-1H7a1 1 0 0 0-1 1H2.5zm3 4a.5.5 0 0 1 .5.5v7a.5.5 0 0 1-1 0v-7a.5.5 0 0 1 .5-.5zM8 5a.5.5 0 0 1 .5.5v7a.5.5 0 0 1-1 0v-7a.5.5 0 0 1 .5-.5zM1 10v7a.5.5 0 0 1 1 0v-7a.5.5 0 0 1-1 0v-7z"/>
  </svg>
</button>
```

На странице появится, новая кнопка, которая открывает модальное окно. Логика следующая:

по нажатию кнопки добавления отдела или редактирования сработает метод `addClick` или `editClick` соответственно. В каждом из этих методов устанавливается название модального окна `ModalTitle`, которое отображается с помощью `two way banding`, и устанавливается `ActivateAddEditDepComp = true`. Если `ActivateAddEditDepComp = true` с помощью директив рисуется разметка селектора `app-add-edit-dep`, который принадлежит компоненту `add-edit-dep`;

По нажатию кнопки удаления сработает метод `deleteClick`, но он пока ничего не делает, только выводит лог в консоль (можете проверить самостоятельно).

Как можем видеть в методы `editClick` и `deleteClick` передаётся объект, который соответствует той строке, на которой и была нажата кнопка.





Новый вид страницы:

## Angular 10, Web API, SQL Server App Lesson

Departments

Employees

Add Department

DepartmentId	Department Name	Options
1	IT	 
2	Support	 

Модальное окно по нажатию кнопки добавления:

Add Department

add-edit-dep works!

Модальное окно по нажатию кнопки редактирования:

Edit Department

add-edit-dep works!

Создадим логику работы add-edit-dep компонента:

Содержание src\app\department\add-edit-dep\add-edit-dep.component.ts изменим на:

```
import { Component, OnInit, Input } from '@angular/core';
import { SharedService } from 'src/app/shared.service';

@Component({
  selector: 'app-add-edit-dep',
  templateUrl: './add-edit-dep.component.html',
  styleUrls: ['./add-edit-dep.component.css']
})
export class AddEditDepComponent implements OnInit {

  constructor(private service: SharedService) { }

  @Input() dep: any;
  DepartmentId: string;
  DepartmentName: string;

  ngOnInit(): void {
    this.DepartmentId = this.dep.DepartmentId;
    this.DepartmentName = this.dep.DepartmentName;
  }

  addDepartment(){
    var val = {DepartmentId: this.DepartmentId,
               DepartmentName: this.DepartmentName};
    this.service.addDepartment(val).subscribe(res => {
      alert(res.toString());
    });
  }

  updateDepartment(){
    var val = {DepartmentId: this.DepartmentId,
               DepartmentName: this.DepartmentName};
    this.service.updateDepartment(val).subscribe(res => {
      alert(res.toString());
    });
  }
}
```

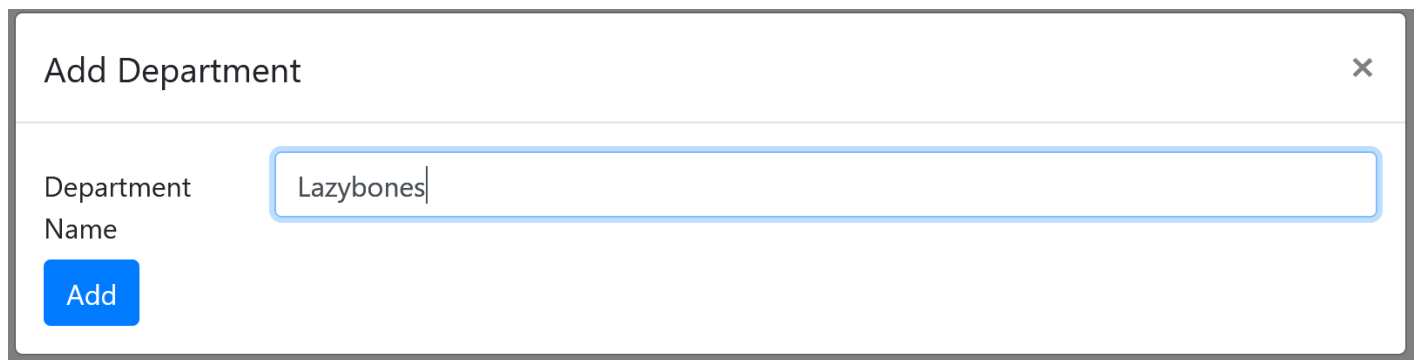
A src\app\department\add-edit-dep\add-edit-dep.component.html на:

```
<div class="form-froup row">
  <label class="col-sm-2 col-form-label">Department Name</label>
  <div class="col-sm-10">
    <input type="text" class="form-control" [(ngModel)]="DepartmentName"
      placeholder="Enter Department Name">
  </div>
</div>

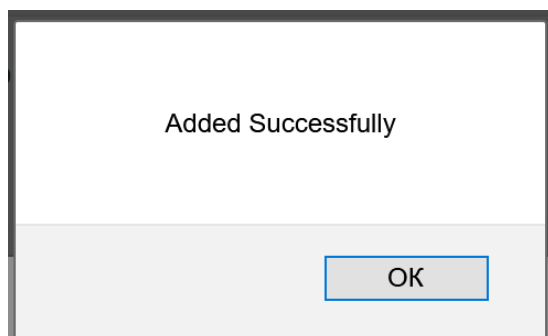
<button (click)="addDepartment()" *ngIf="dep.DepartmentId==0" class="btn btn-
primary">
  Add
</button>

<button (click)="updateDepartment()" *ngIf="dep.DepartmentId!=0" class="btn btn
-primary">
  Update
</button>
```

Теперь модальное окно примет следующий вид:



По нажатию получим уведомление, что запись отдела Lazybones, прошла успешно.











Таблица, как помним, обновляется по закрытию модального окна:

### Angular 10, Web API, SQL Server App Lesson

Departments

Employees

Add Department

DepartmentId	Department Name	Options
1	IT	 
2	Support	 
4	Lazybones	 

Видим запись успешно добавлена! (Рекомендуется ещё самостоятельно перепроверить это в SSMS).

Проверим функцию редактирования:

Edit Department

Department Name

Lazybones

Update







Updated Successfully

OK

Departments

Employees

Add Department

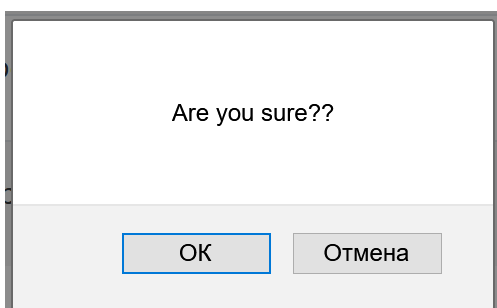
DepartmentId	Department Name	Options
1	IT	 
2	Support	 
4	LazybonesDep	 

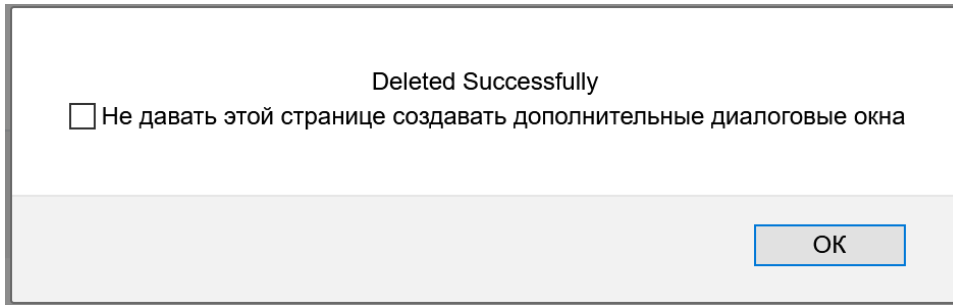
Видим, что всё прекрасно работает!

Содержать отдел лентяев никто не хочет, поэтому нужно написать логику для метода `deleteClick`, чтобы его удалить этот отдел. изменим содержание данного метода на:

```
if(confirm('Are you sure??')){  
    this.service.deleteDepartment(item.DepartmentId).subscribe(data=>{  
        alert(data.toString());  
        this.refreshDepList();  
    })  
}
```

Проверяем:





(Вторую строку добавил сам браузер, так что не будем на него обижаться, он хотел как лучше)

---

#### Angular 10, Web API, SQL Server App Lesson

Departments

Employees

Add Department

DepartmentId	Department Name	Options
1	IT	<div><div></div><div></div></div>
2	Support	<div><div></div><div></div></div>

Видим, что всё прекрасно работает!

Теперь всё это нужно повторить для таблицы Employee и соответствующих компонентов.

Содержимое `src\app\employee\employee.component.html` заменим на:

```
<app-show-emp></app-show-emp>
```

Содержимое `src\app\employee\show-emp\show-emp.component.ts` заменим на:

```
import { Component, OnInit } from '@angular/core';
import { SharedService } from 'src/app/shared.service';

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

```

constructor(private service:SharedService) { }

EmployeeList:any=[];

ModalTitle:string;
ActivateAddEditEmpComp:boolean=false;
emp:any;

ngOnInit(): void {
    this.refreshEmpList();
}

addClick(){
    this.emp={
        EmployeeId:0,
        EmployeeName:"",
        Department:"",
        DateOfJoining:"",
        PhotoFileName:"cat1.png"
    }
    this.ModalTitle="Add Employee";
    this.ActivateAddEditEmpComp=true;
}

editClick(item){
    console.log(item);
    this.emp=item;
    this.ModalTitle="Edit Employee";
    this.ActivateAddEditEmpComp=true;
}

deleteClick(item){
    if(confirm('Are you sure??')){
        this.service.deleteEmployee(item.EmployeeId).subscribe(data=>{
            alert(data.toString());
            this.refreshEmpList();
        })
    }
}

```

```

closeClick(){
  this.ActivateAddEditEmpComp=false;
  this.refreshEmpList();
}

refreshEmpList(){
  this.service.getEmpList().subscribe(data=>{
    this.EmployeeList=data;
  });
}
}

```

Содержимое src\app\employee\show-emp\show-emp.component.html заменим на:

```

<button type="button" class="btn btn-primary float-right m-2"
data-toggle="modal" data-target="#exampleModal"
(click)="addClick()"
data-backdrop="static" data-keyboard="false"
>
  Add Employee
</button>
<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" role="dialog" aria-
labelledby="exampleModalLabel" aria-hidden="true">
  <div class="modal-dialog modal-dialog-centered modal-xl" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">{{ModalTitle}}</h5>
        <button type="button" class="close"
data-dismiss="modal" aria-label="Close"
(click)="closeClick()" >
          <span aria-hidden="true">&times;</span></button>
      </div>
      <div class="modal-body">
        <app-add-edit-emp [emp]="emp" *ngIf="ActivateAddEditEmpComp">
        </app-add-edit-emp>
      </div>
    </div>
  </div>
</div>
</div>

```

```

<table class="table table-striped">
  <thead>
    <tr>
      <th>EmployeeId</th>
      <th>EmployeeName</th>
      <th>Department</th>
      <th>DateOfJoining</th>
      <th>Options</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let dataItem of EmployeeList">
      <td>{{dataItem.EmployeeId}}</td>
      <td>{{dataItem.EmployeeName}}</td>
      <td>{{dataItem.Department}}</td>
      <td>{{dataItem.DateOfJoining}}</td>
      <td>
        <button type="button" class="btn btn-light mr-1"
          data-toggle="modal" data-target="#exampleModal"
          (click)="editClick(dataItem)"
          data-backdrop="static" data-keyboard="false"
        >
          <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-pencil-square" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
            <path d="M15.502 1.94a.5.5 0 0 1 0 .706L14.459 3.691-2-
            2L13.502.646a.5.5 0 0 1 .707 0l1.293 1.293zm-1.75 2.456l-2-
            2L4.939 9.21a.5.5 0 0 0-.121.196l-.805 2.414a.25.25 0 0 0 .316.316l2.414-
            .805a.5.5 0 0 0 .196-.121l6.813-6.814z"/>
            <path fill-
            rule="evenodd" d="M1 13.5A1.5 1.5 0 0 0 2.5 15h11a1.5 1.5 0 0 0 1.5-1.5v-
            6a.5.5 0 0 0-1 0v6a.5.5 0 0 1-.5.5h-11a.5.5 0 0 1-.5-.5v-11a.5.5 0 0 1 .5-
            .5H9a.5.5 0 0 0 0-1H2.5A1.5 1.5 0 0 0 1 2.5v11z"/>
          </svg>
        </button>
        <button type="button" class="btn btn-light mr-1"
          (click)="deleteClick(dataItem)"
        >
          <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-trash-fill" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
            <path fill-rule="evenodd" d="M2.5 1a1 1 0 0 0 0-1 1 1 0 0 0 1 1H3v9a2 2 0 0 0 2 2h6a2 2 0 0 0 2-2V4h.5a1 1 0 0 0 1-

```

```

1V2a1 1 0 0 0-1-1H10a1 1 0 0 0-1-1H7a1 1 0 0 0-
1 1H2.5zm3 4a.5.5 0 0 1 .5.5v7a.5.5 0 0 1-1 0v-7a.5.5 0 0 1 .5-
.5zM8 5a.5.5 0 0 1 .5.5v7a.5.5 0 0 1-1 0v-7A.5.5 0 0 1 8 5zm3 .5a.5.5 0 0 0-
1 0v7a.5.5 0 0 0 1 0v-7z"/>
        </svg>
      </button>
    </td>
  </tr>
</tbody>
</table>

```

Страница будет выглядеть:

Angular 10, Web API, SQL Server App Lesson

Departments
Employees

Add Employee

EmployeeId	EmployeeName	Department	DateOfJoining	Options
1	Sam	IT	2021-05-15	<div style="display: flex; gap: 5px;"> <div style="border: 1px solid #ccc; padding: 2px 5px; border-radius: 4px;">✎</div> <div style="border: 1px solid #ccc; padding: 2px 5px; border-radius: 4px;">🗑</div> </div>
2	Tom	Support	2019-06-11	<div style="display: flex; gap: 5px;"> <div style="border: 1px solid #ccc; padding: 2px 5px; border-radius: 4px;">✎</div> <div style="border: 1px solid #ccc; padding: 2px 5px; border-radius: 4px;">🗑</div> </div>
3	Jerry	Support	2019-06-11	<div style="display: flex; gap: 5px;"> <div style="border: 1px solid #ccc; padding: 2px 5px; border-radius: 4px;">✎</div> <div style="border: 1px solid #ccc; padding: 2px 5px; border-radius: 4px;">🗑</div> </div>

На данный момент кнопка удаления уже работает, но протестируем это позже. А вот кнопки добавления и редактирования ещё нет. Для этого нужно дописать логику для компонента `add-edit-emp`.

Содержимое `src\app\employee\add-edit-emp\add-edit-emp.component.ts` заменим на:

```

import { Component, OnInit, Input } from '@angular/core';
import { SharedService } from 'src/app/shared.service';

@Component({
  selector: 'app-add-edit-emp',
  templateUrl: './add-edit-emp.component.html',
  styleUrls: ['./add-edit-emp.component.css']
})

```

```

export class AddEditEmpComponent implements OnInit {

    constructor(private service:SharedService) { }

    @Input() emp:any;
    EmployeeId:string;
    EmployeeName:string;
    Department:string;
    DateOfJoining:string;
    PhotoFileName:string;
    PhotoFilePath:string;

    DepartmentsList:any=[];

    ngOnInit(): void {
        this.loadDepartmentList();
    }

    loadDepartmentList(){
        this.service.getAllDepartmentNames().subscribe((data:any)=>{
            this.DepartmentsList=data;

            this.EmployeeId=this.emp.EmployeeId;
            this.EmployeeName=this.emp.EmployeeName;
            this.Department=this.emp.Department;
            this.DateOfJoining=this.emp.DateOfJoining;
            this.PhotoFileName=this.emp.PhotoFileName;
            this.PhotoFilePath=this.service.PhotoUrl+this.PhotoFileName;
        });
    }

    addEmployee(){
        var val = {EmployeeId:this.EmployeeId,
                    EmployeeName:this.EmployeeName,
                    Department:this.Department,
                    DateOfJoining:this.DateOfJoining,
                    PhotoFileName:this.PhotoFileName};

        this.service.addEmployee(val).subscribe(res=>{
            alert(res.toString());
        });
    }
}

```



```

updateEmployee(){
  var val = {EmployeeId:this.EmployeeId,
    EmployeeName:this.EmployeeName,
    Department:this.Department,
    DateOfJoining:this.DateOfJoining,
    PhotoFileName:this.PhotoFileName};

  this.service.updateEmployee(val).subscribe(res=>{
    alert(res.toString());
  });
}

uploadPhoto(event){
  var file=event.target.files[0];
  const formData:FormData=new FormData();
  formData.append('uploadedFile',file,file.name);

  this.service.UploadPhoto(formData).subscribe((data:any)=>{
    this.PhotoFileName=data.toString();
    this.PhotoFilePath=this.service.PhotoUrl+this.PhotoFileName;
  })
}
}

```

Этот код аналогичный `src\app\department\add-edit-dep\add-edit-dep.component.ts` только добавлены для новых метода: метод `loadDepartmentList`, который загружает список всех доступных отделов, которые будут отображаться в списке при добавлении нового сотрудника или его редактировании, и метод `uploadPhoto`, который даёт возможность загружать фото с ПК пользователя и сохраняет его на стороне API.

Содержимое `src\app\employee\add-edit-emp\add-edit-emp.component.html` заменим на:

```

<div class="d-flex flex-row bd-highlight mb-3">

  <div class="form-froup row" style="width: 60%;">

    <label class="col-sm-2 col-form-label">Employee Name</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" [(ngModel)]="EmployeeName"
        placeholder="Enter Employee Name">
    </div>
  </div>

```

```

</div>

<label class="col-sm-2 col-form-label">Department</label>
<div class="col-sm-10">
  <select class="form-control" [(ngModel)]="Department">
    <option>--Select--</option>
    <option *ngFor="let depName of DepartmentsList">
      {{depName.DepartmentName}}
    </option>
  </select>
</div>

<label class="col-sm-2 col-form-label">Date of joining</label>
<div class="col-sm-10">
  <input type="date" [(ngModel)]="DateOfJoining">
</div>

</div>
<div class="ml-3 bd-highlight" style="width: 40%;">
  <img [src]=PhotoFilePath height="250px;" width="200px;">
  <input type="file" (change)="uploadPhoto($event)" class="mt-2"/>
</div>
</div>

<button (click)="addEmployee()" *ngIf="emp.EmployeeId==0" class="btn btn-
primary">
  Add
</button>

<button (click)="updateEmployee()" *ngIf="emp.EmployeeId!=0" class="btn btn-
primary">
  Update
</button>

```

Протестируем добавление, редактирование и удаление записей в таблице Employee.

По нажатию кнопки добавление откроется модальное окно:

Add Employee

Employee Name

Enter Employee Name

Department

Date of joining

ДД . ММ . ГГГГ

Обзор...

Файл не выбран.

Add

Пока файл не выбран по умолчанию загружается картинка прописанная в методе `addClick`. В данном случае это `cat1.png`.

Заполним форму:

Add Employee

Employee Name

Bob

Department

IT

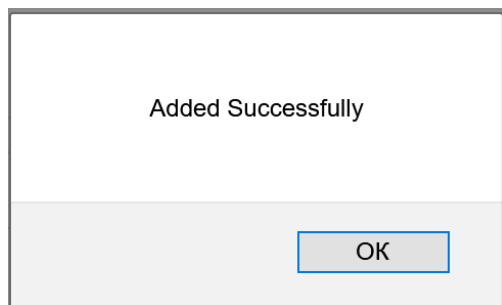
Date of joining

15.05.2021

Обзор...

test.jpg

Add



## Angular 10, Web API, SQL Server App Lesson

<div>Departments</div> <div>Employees</div> <div>Add Employee</div>				
EmployeeId	EmployeeName	Department	DateOfJoining	Options
1	Sam	IT	2021-05-15	<div><div></div><div></div></div>
2	Tom	Support	2019-06-11	<div><div></div><div></div></div>
3	Jerry	Support	2019-06-11	<div><div></div><div></div></div>
5	Bob	IT	2021-05-15	<div><div></div><div></div></div>

Как видим добавление работает! (Рекомендуется ещё самостоятельно перепроверить это в SSMS) (и проверить, что фото и вправду загрузилось в директорию с фото в API приложении).

При выборе редактирования записи открывается форма со всей информацией о сотруднике и его изображением:


Edit Employee

Employee NameTom

DepartmentSupport

Date of joining11.06.2019

Update



Обзор... Файл не выбран.

Но Тома трогать мы не будем, он уже натерпелся от Джерри, поэтому изменим информацию про Боба:

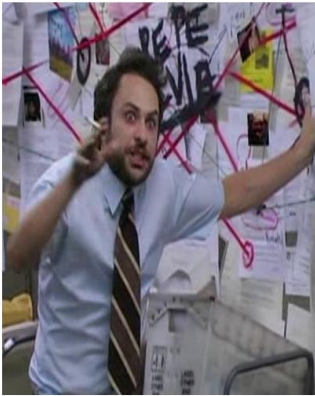
Edit Employee

Employee NameBoby

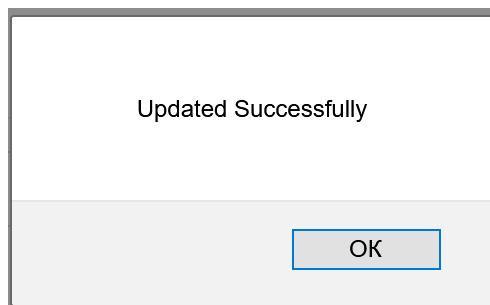
DepartmentSupport

Date of joining15.05.2021

Update



Обзор... Файл не выбран.

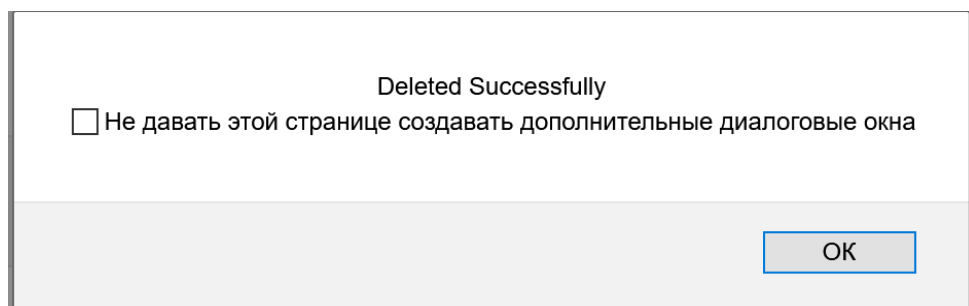
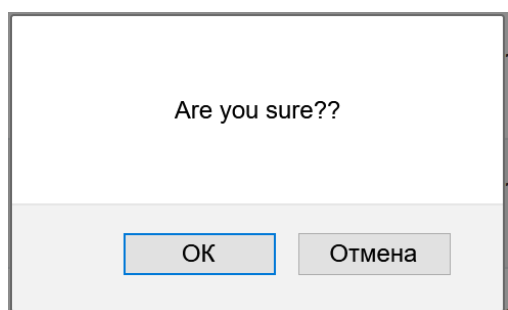


## Angular 10, Web API, SQL Server App Lesson

<div>Departments</div> <div>Employees</div>		<div>Add Employee</div>		
EmployeeId	EmployeeName	Department	DateOfJoining	Options
1	Sam	IT	2021-05-15	<div><div></div><div></div></div>
2	Tom	Support	2019-06-11	<div><div></div><div></div></div>
3	Jerry	Support	2019-06-11	<div><div></div><div></div></div>
5	Boby	Support	2021-05-15	<div><div></div><div></div></div>







Как видим изменение записей работает!

Вид у Боба что-то не очень и ему лучше отдохнуть. Удалим строку с ним:



[Departments](#)
[Employees](#)

Add Employee

EmployeeId	EmployeeName	Department	DateOfJoining	Options
1	Sam	IT	2021-05-15	 
2	Tom	Support	2019-06-11	 
3	Jerry	Support	2019-06-11	 

Удаление работает!

Добавим последний штрих в нашу работу – фильтрацию и сортировку записей в таблице Department. Для этого в класс `src\app\department\show-dep\show-dep.component.ts` добавим переменные, методы для фильтрации и сортировки:

```
deleteClick(item){
  if(confirm('Are you sure??')){
    this.service.deleteDepartment(item.DepartmentId).subscribe(data=>{
      alert(data.toString());
      this.refreshDepList();
    })
  }
}

FilterFn(){
  var DepartmentIdFilter = this.DepartmentIdFilter;
  var DepartmentNameFilter = this.DepartmentNameFilter;

  this.DepartmentList = this.DepartmentListWithoutFilter.filter(function (el)
  {
    return el.DepartmentId.toString().toLowerCase().includes(
      DepartmentIdFilter.toString().trim().toLowerCase()
    )&&
    el.DepartmentName.toString().toLowerCase().includes(
      DepartmentNameFilter.toString().trim().toLowerCase()
    )
  })
}
```





```

1 0v4.793L5.354 7.646a.5.5 0 1 0-.708.708l3 3a.5.5 0 0 0 .708 0l3-3a.5.5 0 0 0-.
.708-.708L8.5 9.793V5z"/>
    </svg>
  </button>
  <button type="button" class="btn btn-light"
    (click)="sortResult('DepartmentId',false)">
    <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-
arrow-up-square-fill" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
      <path fill-rule="evenodd" d="M2 0a2 2 0 0 0-
2 2v12a2 2 0 0 0 2 2h12a2 2 0 0 0 2-2V2a2 2 0 0 0-2-
2H2zm3.354 8.354a.5.5 0 1 1-.708-.708l3-3a.5.5 0 0 1 .708 0l3 3a.5.5 0 0 1-
.708.708L8.5 6.207V11a.5.5 0 0 1-1 1 0V6.207L5.354 8.354z"/>
    </svg>
  </button>

</div>







DepartmentId</th>
<th>
  <div class="d-flex flex-row">
    <input [(ngModel)]="DepartmentNameFilter" class="form-control"
      (keyup)="FilterFn()" placeholder="Filter">

    <button type="button" class="btn btn-light"
      (click)="sortResult('DepartmentName',true)">
      <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-
arrow-down-square-fill" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
        <path fill-rule="evenodd" d="M2 0a2 2 0 0 0-
2 2v12a2 2 0 0 0 2 2h12a2 2 0 0 0 2-2V2a2 2 0 0 0-2-2H2zm6.5 5a.5.5 0 0 0-
1 0v4.793L5.354 7.646a.5.5 0 1 0-.708.708l3 3a.5.5 0 0 0 .708 0l3-3a.5.5 0 0 0-.
.708-.708L8.5 9.793V5z"/>
      </svg>
    </button>
    <button type="button" class="btn btn-light"
      (click)="sortResult('DepartmentName',false)">
      <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-
arrow-up-square-fill" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
        <path fill-rule="evenodd" d="M2 0a2 2 0 0 0-
2 2v12a2 2 0 0 0 2 2h12a2 2 0 0 0 2-2V2a2 2 0 0 0-2-
2H2zm3.354 8.354a.5.5 0 1 1-.708-.708l3-3a.5.5 0 0 1 .708 0l3 3a.5.5 0 0 1-
.708.708L8.5 6.207V11a.5.5 0 0 1-1 1 0V6.207L5.354 8.354z"/>
      </svg>
    </button>

```

```
</div>
  Department Name</th>
<th>Options</th>
</tr>
</thead>
```

На этом наша работа завершена и как видим фильтрация работает правильно. Работу сортировки предлагается проверить самостоятельно.

<input type="text" value="Filter"/>	 	<input type="text" value="su"/>	 	
DepartmentId		Department Name		Options
2		Support		 

## Подведём итоги.

Мы выполнили комплексную работу по созданию веб-приложения с нуля. Приложение соответствует паттерну MVC и разделить его можно так:

### *Модель:*

Для сохранения данных использована база данных MSSQL. Для её создания мы познакомились с работой в СУБД SSMS.

### *Контроллер:*

API создано с помощью c# Visual Studio. Это приложение принимает и обрабатывает запросы для работы с данными в базе данных. Как способ тестирования и проверки работы API мы ознакомились с Postman.

### *Представление*

Была создана Веб страница по технологии single page application. Для её создания был использован популярный фреймворк Angular и соответственно язык TypeScript.

Весь исходный код можно скачать по ссылке:

API:

[https://github.com/MadVitaliy/trpz4\\_final\\_project\\_webAPI](https://github.com/MadVitaliy/trpz4_final_project_webAPI)

Веб-страница:

[https://github.com/MadVitaliy/trpz4\\_final\\_project\\_angular\\_web\\_page](https://github.com/MadVitaliy/trpz4_final_project_angular_web_page)