

ЛАБОРАТОРНАЯ РАБОТА №7

Разработка программного интерфейса приложения (API)

ЦЕЛЬ РАБОТЫ

Приобрести умения и практические навыки для разработки API.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

- 1) Получить у преподавателя индивидуальный вариант по лабораторной работе.
- 2) Разработать базу данных с необходимым перечнем таблиц согласно индивидуальному варианту в СУБД Microsoft SQL Server.
- 3) Используя технологию Web API разработайте контроллер, который будет содержать действия, обрабатывающие GET, POST, PUT, DELETE запросы к ранее созданной базе данных.
- 4) Разработайте веб-клиент для демонстрации возможностей ранее разработанных методов.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

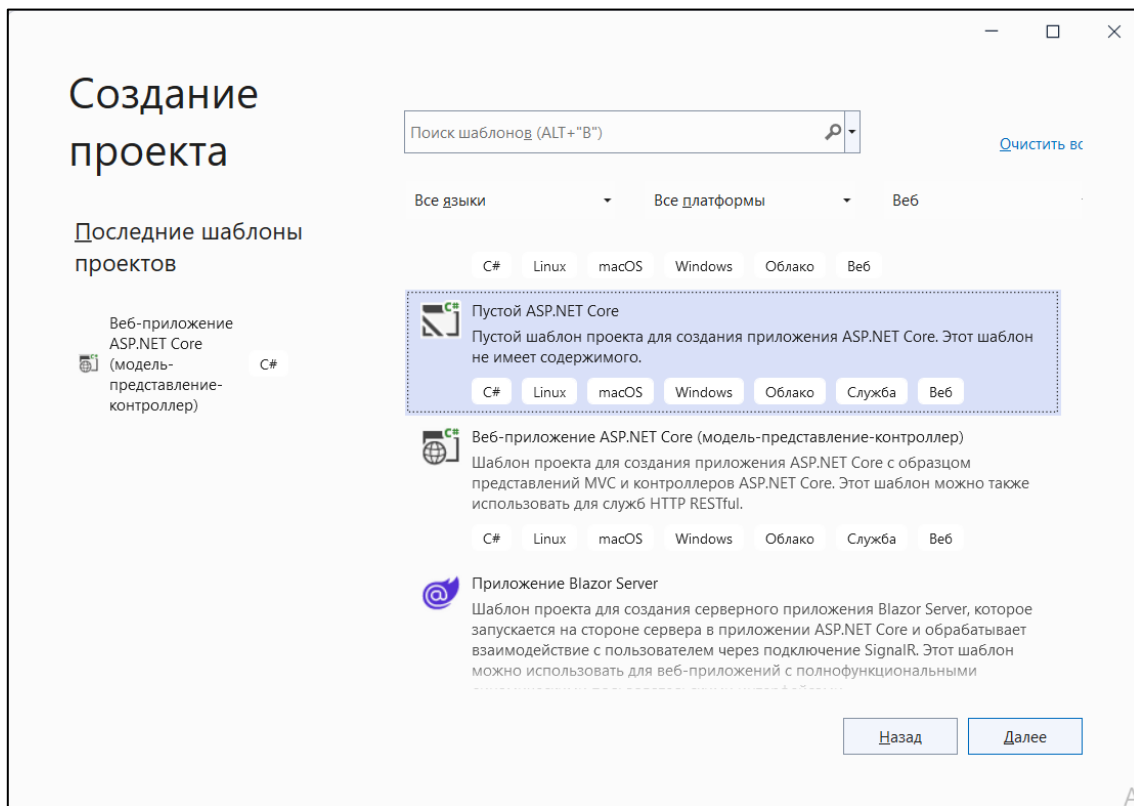
API — это программный интерфейс приложения, позволяющий двум независимым компонентам программного обеспечения обмениваться информацией. API играет роль посредника между внутренними и внешними программными функциями, обеспечивая настолько эффективный обмен информацией, что конечный пользователь обычно его просто не замечают.

Например, посредством API может быть передана геолокация из мобильного приложения одного человека на сервер, а затем на мобильное устройство другого. В рамках разработки программного решения можно добиться наибольшей эффективности в том случае, если и настольное, и мобильное приложения будут работать с единой базой данных и получать актуальную информацию в режиме реального времени.

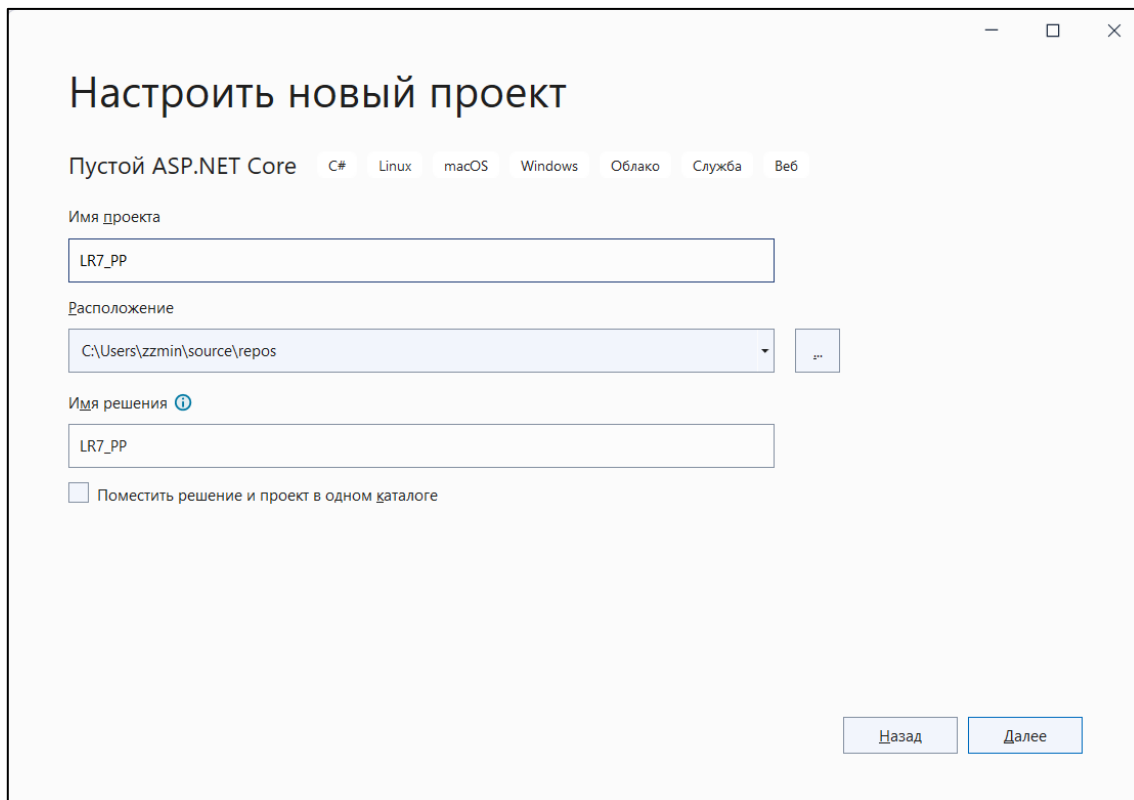
С этой целью создадим собственный API, который сам будет работать с базой данных и возвращать ответ в удобном для приложения виде.

ХОД РАБОТЫ

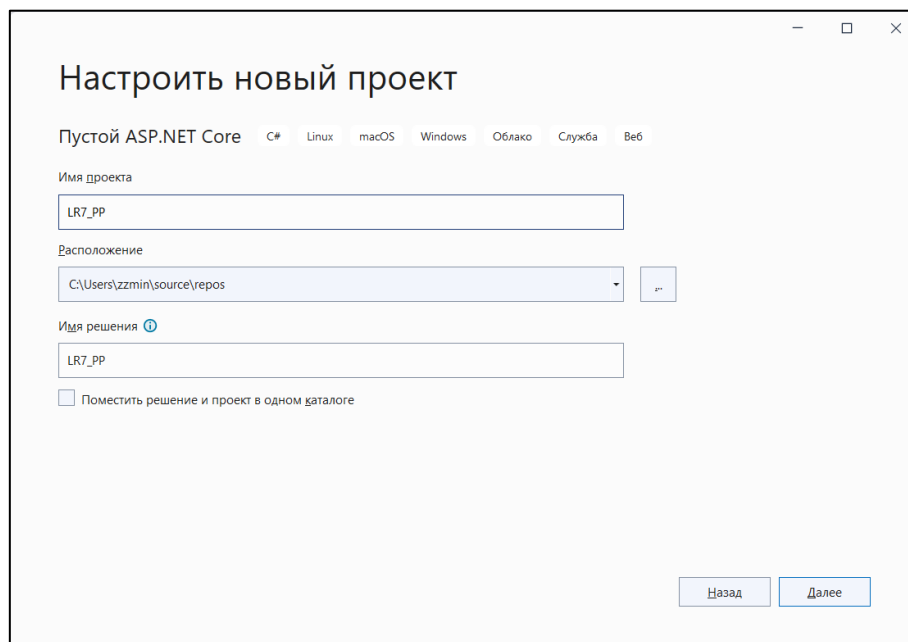
1. Создаем проект «Web API», которое будет выполнять все основные операции с данными. Для этого создадим проект по типу Пустой ASP.NET Core:



- Даем название проекту.



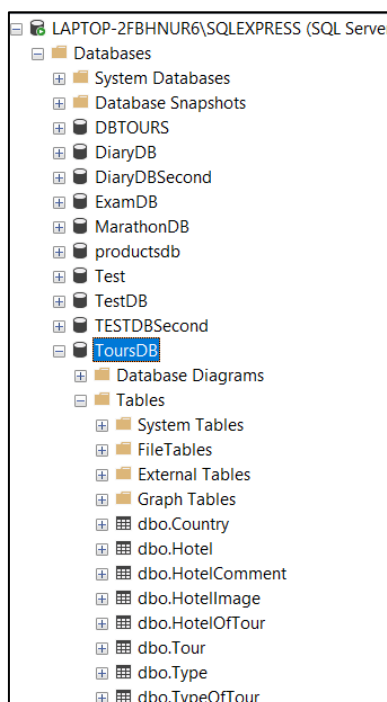
- Убираем галочку с конфигурации https.



2. На данном этапе нам необходимо реализовать три метода в рамках API:

- для получения списка отелей.
- для добавления нового отеля.

Работать будем с базой данных ToursDB. Скрипт для ее восстановления будет приложен к методическому пособию.



3. В проект добавляем модель Entity Framework.

Для работы с данными мы остановимся на формате json. Это набор из двух пар: ключ — значение и упорядоченный набор значений. Это универсальные структуры данных. Как правило, любой современный язык программирования поддерживает их в той или иной форме.

Добавим в проект новую папку **Models**, а в нее поместим новый класс **Hotel**:

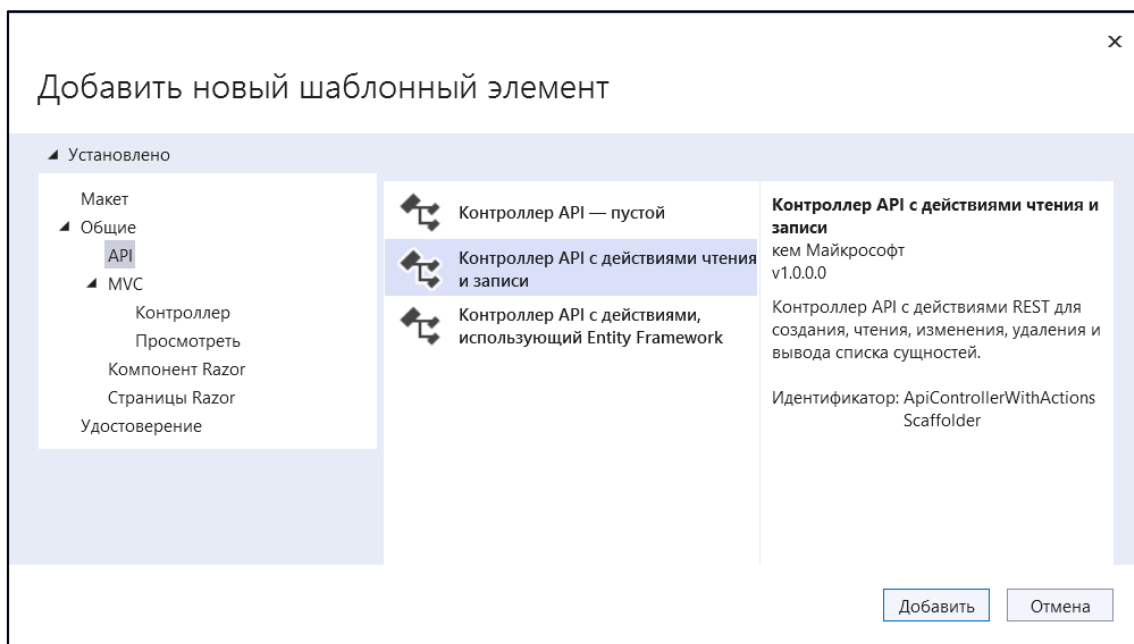
1	<code>public class Hotel</code>
2	<code>{</code>
3	<code>public int Id { get; set; }</code>
4	<code>public string Name { get; set; }</code>
5	<code>public int CountOfStars { get; set; }</code>
6	<code>public string CountryCode { get; set; }</code>
7	<code>public string Description { get; set; }</code>
8	<code>}</code>

Для взаимодействия с MS SQL Server через Entity Framework через пакетный менеджер Nuget добавим в проект пакет **Microsoft.EntityFrameworkCore.SqlServer**.

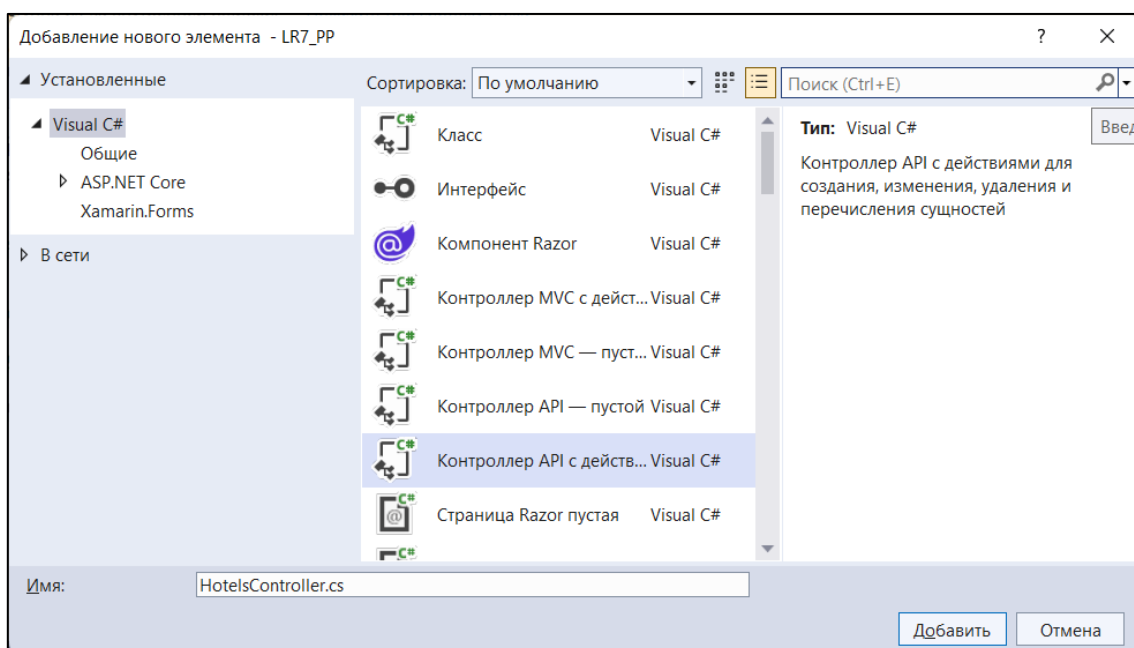
Также добавим в папку **Models** новый класс **HotelsContext** для взаимодействия с базой данных:

1	<code>using Microsoft.EntityFrameworkCore;</code>
2	<code>namespace LR7_PP.Models</code>
3	<code>{</code>
4	<code>public class HotelsContext : DbContext</code>
5	<code>{</code>
6	<code>public DbSet<Hotel> Hotel { get; set; }</code>
7	<code>public HotelsContext (DbContextOptions<HotelsContext></code>
8	<code>options) : base(options)</code>
9	<code>{</code>
10	<code>Database.EnsureCreated();</code>
11	<code>}</code>
12	<code>}</code>

4. Далее добавим в проект новую папку **Controllers**, а в ней создадим новый api-контроллер. Для этого при добавлении нового элемента в проект можно использовать шаблон **Контроллер API с действиями чтения и записи**:



Назовем новый элемент HotelsController.



После его создания изменим его код следующим образом:

1	<code>using LR7_PP.Models;</code>
2	<code>using Microsoft.AspNetCore.Mvc;</code>
3	<code>using Microsoft.EntityFrameworkCore;</code>
4	<code>namespace LR7_PP.Controllers</code>
5	<code>{</code>
6	<code>[Route("api/[controller]")]</code>
7	<code>[ApiController]</code>
8	<code>public class HotelsController : ControllerBase</code>
9	<code>{</code>
10	<code>private HotelsContext? _db;</code>
11	<code>public HotelsController(HotelsContext hotelsContext)</code>

12	{
13	_db = hotelsContext;
14	}
15	// GET: api/<HotelsController>
16	[HttpGet]
17	public async Task<ActionResult<IEnumerable<Hotel>>> Get()
18	{
19	return await _db.Hotel.ToListAsync(); ;
20	}
21	// GET api/<HotelsController>/5
22	[HttpGet("{id}")]
23	public async Task<ActionResult<Hotel>> Get(int id)
24	{
25	Hotel hotel = await _db.Hotel.FirstOrDefaultAsync(x => x.Id == id);
26	if (hotel == null)
27	return NotFound();
28	return new ObjectResult(hotel);
29	}
30	// POST api/<HotelsController>
31	[HttpPost]
32	public async Task<ActionResult<Hotel>> Post(Hotel hotel)
33	{
34	if (hotel == null)
35	{
36	return BadRequest();
37	}
38	_db.Hotel.Add(hotel);
39	await _db.SaveChangesAsync();
40	return Ok(hotel);
41	}
42	// PUT api/<HotelsController>/5
43	[HttpPut("{id}")]
44	public async Task<ActionResult<Hotel>> Put(Hotel hotel)
45	{
46	if (hotel == null)
47	{
48	return BadRequest();
49	}
50	if (!_db.Hotel.Any(x => x.Id == hotel.Id))
51	{
52	return NotFound();
53	}

54	<code>_db.Update(hotel);</code>
55	<code>await _db.SaveChangesAsync();</code>
56	<code>return Ok(hotel);</code>
57	<code>}</code>
58	<code>// DELETE api/<HotelsController>/5</code>
59	<code>[HttpDelete("{id}")]</code>
60	<code>public async Task<ActionResult<Hotel>> Delete(int id)</code>
61	<code>{</code>
62	<code>Hotel hotel = _db.Hotel.FirstOrDefault(x => x.Id == id);</code>
63	<code>if (hotel == null)</code>
64	<code>{</code>
65	<code>return NotFound();</code>
66	<code>}</code>
67	<code>_db.Hotel.Remove(hotel);</code>
68	<code>await _db.SaveChangesAsync();</code>
69	<code>return Ok(hotel); ;</code>
70	<code>}</code>
71	<code>}</code>
72	<code>}</code>

Прежде всего к контроллеру применяется атрибут `[ApiController]`, который позволяет использовать ряд дополнительных возможностей, в частности, в плане привязки модели и ряд других. Также к контроллеру применяется атрибут маршрутизации, который указывает, как контроллер будет сопоставляться с запросами.

В конструкторе контроллера получаем контекст данных и используем его для операций с данными.

Контроллер API предназначен преимущественно для обработки запросов протокола HTTP: Get, Post, Put, Delete, Patch, Head, Options. В данном случае для каждого типа запросов в контроллере определен свои методы. Так, метод `Get()` обрабатывает запросы типа GET и возвращает коллекцию объектов из базы данных.

Если запрос GET содержит параметр `id` (идентификатор объекта), то он обрабатывается другим методом - `Get(int id)`, который возвращает объект по переданному `id`.

Запросы типа POST обрабатываются методом `Post(Hotel hotel)`, который получает из тела запроса отправленные данные и добавляет их в базу данных.

Метод Put(Hotel hotel) обрабатывает запросы типа Put - получает данные из запроса и изменяет ими объект в базе данных.

И метод Delete(int id) обрабатывает запросы типа DELETE, то есть запросы на удаление - получает из запроса параметр id и по данному идентификатору удаляет объект из БД.

5. Теперь, чтобы это все использовать, изменим код класса Startup:

1	using LR7_PP.Models;
2	using Microsoft.AspNetCore.Builder;
3	using Microsoft.Data.SqlClient;
4	using Microsoft.EntityFrameworkCore;
5	namespace LR7_PP
6	{
7	public class Startup
8	{
9	public void ConfigureServices(IServiceCollection services)
10	{
11	// устанавливаем контекст данных
12	services.AddDbContext<HotelsContext>(options =>
13	options.UseSqlServer(SqlConnectionIntegratedSecurity));
14	services.AddControllers(); // используем контроллеры без представлений
15	}
16	public static string SqlConnectionIntegratedSecurity
17	{
18	get
19	{
20	var sb = new SqlConnectionStringBuilder
21	{
22	DataSource = "LAPTOP-2BI2ASH4",
23	// Подключение будет с проверкой подлинности пользователя Windows
24	IntegratedSecurity = true,
25	// Название целевой базы данных.
26	InitialCatalog = "ToursDB"
27	};
28	return sb.ConnectionString;
29	}
30	public void Configure(IApplicationBuilder app)
31	{
32	app.UseDeveloperExceptionPage();

33	<code>app.UseRouting();</code>
34	<code>app.UseEndpoints(endpoints =></code>
35	<code>{</code>
36	<code>endpoints.MapControllers(); // подключаем маршрутизацию на</code> <code>контроллеры</code>
37	<code>});</code>
38	<code>}</code>
39	<code>}</code>
40	<code>}</code>

Строку подключения к MS SQL Server удобно создавать с помощью строителей. Например, с помощью класса `SqlConnectionStringBuilder`, входящего в состав пространства имён `Microsoft.Data.SqlClient` (пакет `Microsoft.EntityFrameworkCore.SqlServer`). Таким способом создаётся синтаксически правильная строка подключения к серверу базы данных.

Строка подключения отправляет серверу информацию для идентификации клиента, способе проверки клиента и выборе базы данных для последующих запросов. Разные режимы проверки подлинности пользователя требуют соответствующее содержание строки подключения.

Чтобы задействовать контроллеры, в методе `ConfigureServices()` вызывается метод `services.AddControllers()`.

Чтобы подключить маршрутизацию контроллеров на основе атрибутов, в методе `Configure()` вызывается метод `endpoints.MapControllers()`. После этого мы сможем обращаться к контроллеру через запрос

`api/hotels,`

поскольку к контроллеру применяется атрибут маршрутизации `[Route("api/[controller]")]`, где параметр "controller" указывает на название контроллера.

Запустим приложение и обратимся по пути `api/hotels`:



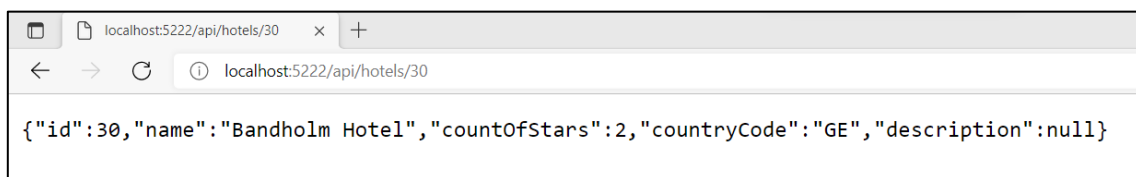
```
[{"id":1,"name":"Hotel Artemide","countOfStars":5,"countryCode":"ES","description":null}, {"id":2,"name":"H10 Madison","countOfStars":4,"countryCode":"FI","description":null}, {"id":3,"name":"A Room With A View","countOfStars":3,"countryCode":"ES","description":null}, {"id":4,"name":"Hotel Rec Barcelona","countOfStars":5,"countryCode":"ES","description":null}, {"id":5,"name":"Aydinli Cave Hotel","countOfStars":2,"countryCode":"ES","description":null}, {"id":6,"name":"Titanic Business Kartal","countOfStars":3,"countryCode":"ES","description":null}, {"id":7,"name":"Amber Design Residence","countOfStars":3,"countryCode":"FI","description":null}, {"id":8,"name":"Hotel Al Ponte Mocenigo","countOfStars":5,"countryCode":"RU","description":null}, {"id":9,"name":"La Cachette","countOfStars":4,"countryCode":"RU","description":null}, {"id":10,"name":"Celsus Boutique Hotel","countOfStars":3,"countryCode":"GB","description":null}, {"id":11,"name":"Ashford Castle","countOfStars":2,"countryCode":"RU","description":null}, {"id":12,"name":"Agarta Cave Hotel","countOfStars":5,"countryCode":"GB","description":null}, {"id":13,"name":"Sofitel Grand Sopot","countOfStars":5,"countryCode":"FI","description":null}, {"id":14,"name":"Grand Resort Bad Ragaz","countOfStars":4,"countryCode":"RU","description":null}, {"id":15,"name":"Kelebek Special Cave Hotel","countOfStars":5,"countryCode":"GB","description":null}, {"id":16,"name":"A Room With A View","countOfStars":3,"countryCode":"GB","description":null}, {"id":17,"name":"Aren Cave Hotel & Art Gallery","countOfStars":5,"countryCode":"GE","description":null}, {"id":18,"name":"La Cachette","countOfStars":4,"countryCode":"RU","description":null}, {"id":19,"name":"Castle Hotel Auf Schoenburg","countOfStars":3,"countryCode":"GE","description":null}, {"id":20,"name":"Lawton & Lauriston Court Hotel","countOfStars":5,"countryCode":"GB","description":null}, {"id":21,"name":"Elif Hanim Hotel & Spa","countOfStars":3,"countryCode":"RU","description":null}, {"id":22,"name":"A Room With A View","countOfStars":3,"countryCode":"ES","description":null}]
```

Поскольку запрос из адресной строки браузера представляет GET-запрос, то его будет обрабатывать метод

```
// GET: api/<HotelsController>
[HttpGet]
public async Task<ActionResult<IEnumerable<Hotel>>>
Get()
{
    return await _db.Hotel.ToListAsync(); ;
}
// GET api/<HotelsController>/5
```

Этот метод возвратит все отели из базы данных. Поэтому в браузере мы увидим все те данные, которые были добавлены в конструкторе.

Передадим параметр id:



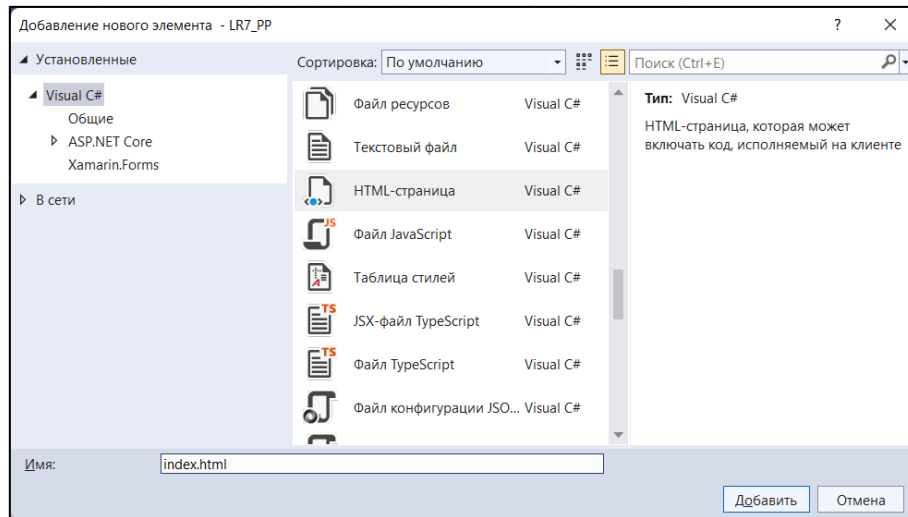
```
{"id":30,"name":"Bandholm Hotel","countOfStars":2,"countryCode":"GE","description":null}
```

Поскольку это также запрос типа GET, но теперь также передается параметр id, то работает следующий метод:

```
[HttpGet("{id}")]
public async Task<ActionResult<Hotel>> Get(int id)
{
    Hotel hotel = await _db.Hotel.FirstOrDefaultAsync(x => x.Id == id);
    if (hotel == null)
        return NotFound();
    return new ObjectResult(hotel);
}
```

6. Создадим часть, которая будет представлять веб-страницу. То есть из веб-страницы мы будем отправлять запросы к контроллеру и обрабатывать ответ от контроллера.

Для создания веб-клиента добавим в проект папку **wwwroot** и затем в ней определим новый элемент **HTML-страница**, который назовем **index.html**, и **Файл JavaScript**, который назовем **script.js**:



Затем изменим класс `Startup`, добавив в методе `Configure()` два вызова для работы со статическими файлами:

```
app.UseDefaultFiles();
app.UseStaticFiles();
```

Благодаря этому мы сможем обратиться напрямую к веб-странице, например, по пути `http://localhost:xxxx/index.html`. Для этого изменим файл `index.html`:

1	<code><!DOCTYPE html></code>
2	<code><html></code>
3	<code><head></code>
4	<code><script src="script.js"></script></code>
5	<code><meta charset="utf-8" /></code>
6	<code><meta name="viewport" content="width=device-width" /></code>
7	<code><title>Список отелей</title></code>
8	<code><link href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.0/css/bootstrap.min.css" rel="stylesheet" /></code>
9	<code></head></code>
10	<code><body onload="InitialFunction();"></code>
11	<code><h2>Список отелей</h2></code>
12	<code><form name="hotelForm"></code>
13	<code><input type="hidden" name="id" value="0" /></code>

14	<div class="form-group col-md-5">
15	<label for="name">Наименование:</label>
16	<input class="form-control" name="name" />
17	</div>
18	<div class="form-group col-md-5">
19	<label for="countOfStars">Количество звезд:</label>
20	<input class="form-control" name="countOfStars" type="number" />
21	</div>
22	<div class="form-group col-md-5">
23	<label for="countryCode">Код страны:</label>
24	<input class="form-control" name="countryCode" />
25	</div>
26	<div class="panel-body">
27	<button type="submit" id="submit" class="btn btn-primary">Сохранить</button>
28	Сбросить
29	</div>
30	</form>
31	<table class="table table-condensed table-striped col-md-6">
32	<thead><tr><th>Id</th><th>Наименование</th>
	<th>Количество звезд</th><th></th></tr></thead>
33	<tbody>
34	</tbody>
35	</table>
36	<div>2022 Заид Мингалиев</div>
37	</body>
38	</html>

Изменим файл script.js:

1	// Получение всех отелей
2	async function GetHotel() {
3	// отправляет запрос и получаем ответ
4	const response = await fetch("/api/hotels", {
5	method: "GET",
6	headers: { "Accept": "application/json" }
7	});
8	// если запрос прошел нормально

```

9  if (response.ok === true) {
10 // получаем данные
11 const hotels = await response.json();
12 let rows = document.querySelector("tbody");
13 hotels.forEach(hotel => {
14 // добавляем полученные элементы в таблицу
15 rows.append(row(hotel));
16 });
17 }
18 }

19 // Получение одного отеля
20 async function GetHotelById(id) {
21 const response = await fetch("/api/hotels/" + id, {
22 method: "GET",
23 headers: { "Accept": "application/json" }
24 });
25 if (response.ok === true) {
26 const hotel = await response.json();
27 const form = document.forms["hotelForm"];
28 form.elements["id"].value = hotel.id;
29 form.elements["name"].value = hotel.name;
30 form.elements["countOfStars"].value = hotel.countOfStars;
31 }
32 }

    async function CreateHotel(hotelName, hotelCountOfStars,
33 hotelCountryCode) {
34 const response = await fetch("api/hotels", {
35 method: "POST",

```

```

headers: { "Accept": "application/json", "Content-Type":
36 "application/json" },
37 body: JSON.stringify({
38   name: hotelName,
39   countOfStars: parseInt(hotelCountOfStars, 10),
40   countryCode: hotelCountryCode
41 })
42 });
43 if (response.ok === true) {
44   const hotel = await response.json();
45   reset();
46   document.querySelector("tbody").append(row(hotel));
47 }
48 }

49 async function EditHotel(hotelId, hotelName, hotelCountOfStars,
50 hotelCountryCode) {
51   const response = await fetch("/api/hotels/" + hotelId, {
52     method: "PUT",
53     headers: { "Accept": "application/json", "Content-Type":
54       "application/json" },
55     body: JSON.stringify({
56       id: parseInt(hotelId, 10),
57       name: hotelName,
58       countOfStars: parseInt(hotelCountOfStars, 10),
59       countryCode: hotelCountryCode
60     })
61   });
62   if (response.ok === true) {
63     const hotel = await response.json();

```

62	reset();
63	document.querySelector("tr[data-rowid='" + hotel.id + "']").replaceWith(row(hotel));
64	}
65	}
66	// Удаление пользователя
67	async function DeleteHotel(id) {
68	const response = await fetch("/api/hotels/" + id, {
69	method: "DELETE",
70	headers: { "Accept": "application/json" }
71	});
72	if (response.ok === true) {
73	const hotel = await response.json();
74	document.querySelector("tr[data-rowid='" + hotel.id + "']").remove();
75	}
76	}
77	// сброс формы
78	function reset() {
79	const form = document.forms["hotelForm"];
80	form.reset();
81	form.elements["id"].value = 0;
82	}
83	// создание строки для таблицы
84	function row(hotel) {
85	const tr = document.createElement("tr");
86	tr.setAttribute("data-rowid", hotel.id);
87	const idTd = document.createElement("td");
88	idTd.append(hotel.id);

```
89 tr.append(idTd);
90 const nameTd = document.createElement("td");
91 nameTd.append(hotel.name);
92 tr.append(nameTd);
93 const countOfStarsTd = document.createElement("td");
94 countOfStarsTd.append(hotel.countOfStars);
95 tr.append(countOfStarsTd);
96 const linksTd = document.createElement("td");
97 const editLink = document.createElement("a");
98 editLink.setAttribute("data-id", hotel.id);
99 editLink.setAttribute("style", "cursor:pointer;padding:15px;");
100 editLink.append("Изменить");
101 editLink.addEventListener("click", e => {
102 e.preventDefault();
103 GetHotelById(hotel.id);
104 });
105 linksTd.append(editLink);
106 const removeLink = document.createElement("a");
107 removeLink.setAttribute("data-id", hotel.id);
108 removeLink.setAttribute("style", "cursor:pointer;padding:15px;");
109 removeLink.append("Удалить");
110 removeLink.addEventListener("click", e => {
111 e.preventDefault();
112 DeleteHotel(hotel.id);
113 });
114 linksTd.append(removeLink);
115 tr.appendChild(linksTd);
116 return tr;
117 }
```


118	<code>function</code> InitialFunction() {
119	<code>// сброс значений формы</code>
120	<code>document.getElementById("reset").click(function (e) {</code>
121	<code>e.preventDefault();</code>
122	<code>reset();</code>
123	<code>})</code>
124	<code>// отправка формы</code>
125	<code>document.forms["hotelForm"].addEventListener("submit", e => {</code>
126	<code>e.preventDefault();</code>
127	<code>const</code> form = document.forms["hotelForm"];
128	<code>const</code> id = form.elements["id"].value;
129	<code>const</code> name = form.elements["name"].value;
130	<code>const</code> countOfStars = form.elements["countOfStars"].value;
131	<code>const</code> countryCode = form.elements["countryCode"].value;
132	<code>if</code> (id == 0)
133	<code>CreateHotel</code> (name, countOfStars, countryCode);
134	<code>else</code>
135	<code>EditHotel</code> (id, name, countOfStars, countryCode);
136	<code>});</code>
137	<code>GetHotel</code> ();
138	<code>}</code>

Основная логика здесь заключена в коде javascript. При загрузке страницы в браузере получаем все объекты из БД с помощью функции `GetUsers` (строка 1–18).

Fetch API предоставляет метод `fetch()` для работы с запросами. Данный метод поддерживается всеми современными браузерами.

Базовый синтаксис:

```
let promise = fetch(url, [options])
```

- `url` – URL для отправки запроса.
- `options` – дополнительные параметры: метод, заголовки и так далее.

Без `options` это простой GET-запрос, скачивающий содержимое по адресу `url`.

Браузер сразу же начинает запрос и возвращает промис¹, который внешний код использует для получения результата.

Процесс получения ответа обычно происходит в два этапа.

Во-первых, промис выполняется с объектом встроенного класса `Response` в качестве результата, как только сервер пришлёт заголовки ответа.

На этом этапе мы можем проверить статус HTTP-запроса и определить, выполнен ли он успешно, а также посмотреть заголовки, но пока без тела ответа.

Промис завершается с ошибкой, если `fetch` не смог выполнить HTTP-запрос, например при ошибке сети или если нет такого сайта. HTTP-статусы 404 и 500 не являются ошибкой.

Мы можем увидеть HTTP-статус в свойствах ответа:

- `status` – код статуса HTTP-запроса, например 200.
- `ok` – логическое значение: будет `true`, если код HTTP-статуса в диапазоне 200–299.

Во-вторых, для получения тела ответа нам нужно использовать дополнительный вызов метода.

`Response` предоставляет несколько методов, основанных на промисах, для доступа к телу ответа в различных форматах:

- `response.text()` – читает ответ и возвращает как обычный текст,
 - `response.json()` – декодирует ответ в формате JSON,
- и другие форматы.

Для добавления строк в таблицу используется функция `row()` (строка 83–117), которая возвращает строку. В этой строке будут определены ссылки для изменения и удаления отеля.

Ссылка для изменения отеля с помощью функции `GetHotelById()` (строка 19–32) получает с сервера выделенный отель и выделенный отель добавляется в форму над таблицей. Эта же форма применяется и для добавления объекта. С помощью скрытого

¹ Промис — это объект, представляющий результат успешного или неудачного завершения асинхронной операции.

поля, которое хранит `id` пользователя, мы можем узнать, какое действие выполняется - добавление или редактирование. Если `id` равен 0, то выполняется функция `CreateHotel()`, которая отправляет данные в POST-запросе (строка 33–48).

Если же ранее отель был загружен на форму, и в скрытом поле сохранился его `id`, то выполняется функция `EditHotel()` (строка 49–65), которая отправляет PUT-запрос.

При нажатии на ссылку «Удалить» выполняется DELETE-запрос, который вызывает функцию `DeleteHotel()` (строка 66–76) по `id` удаляет пользователя.