Дисциплина: «Прикладное программирование»

**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.

1

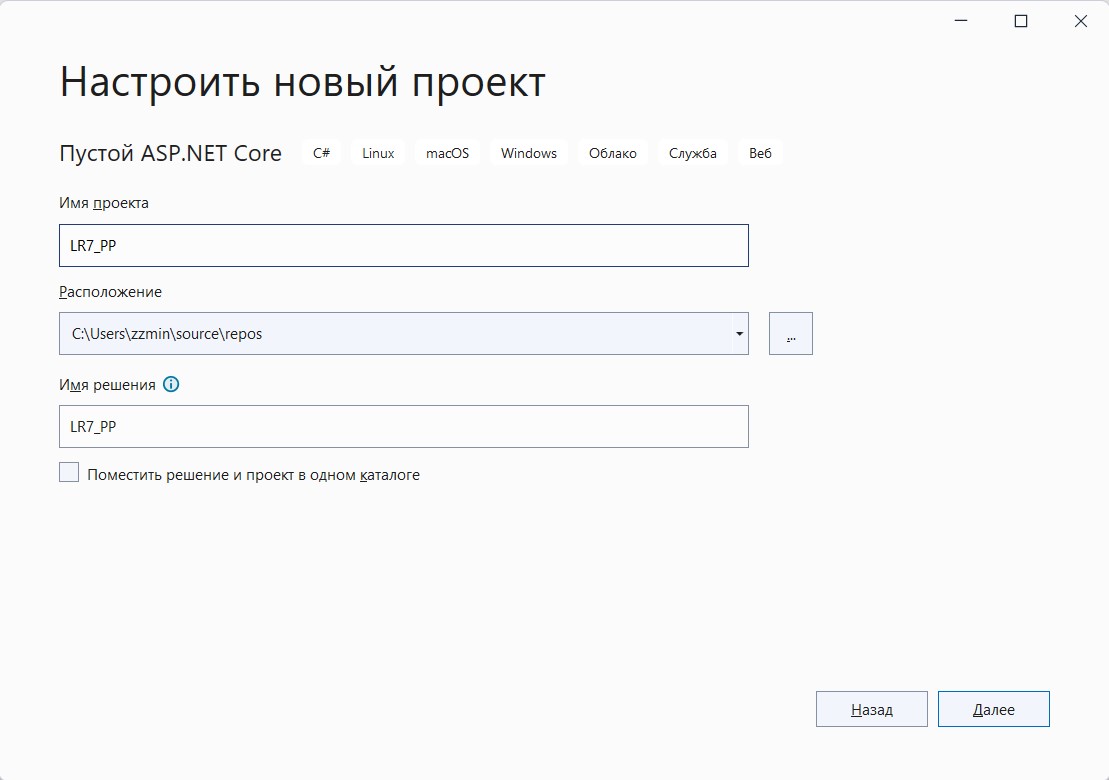
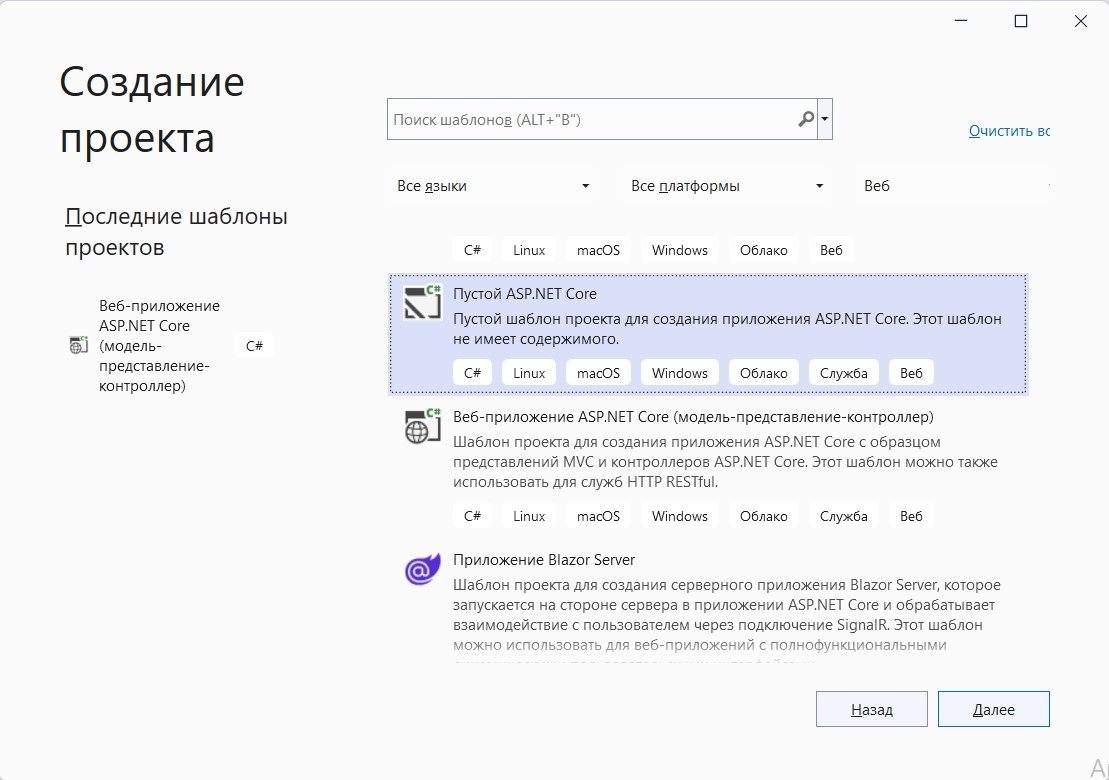
ХОД РАБОТЫ

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

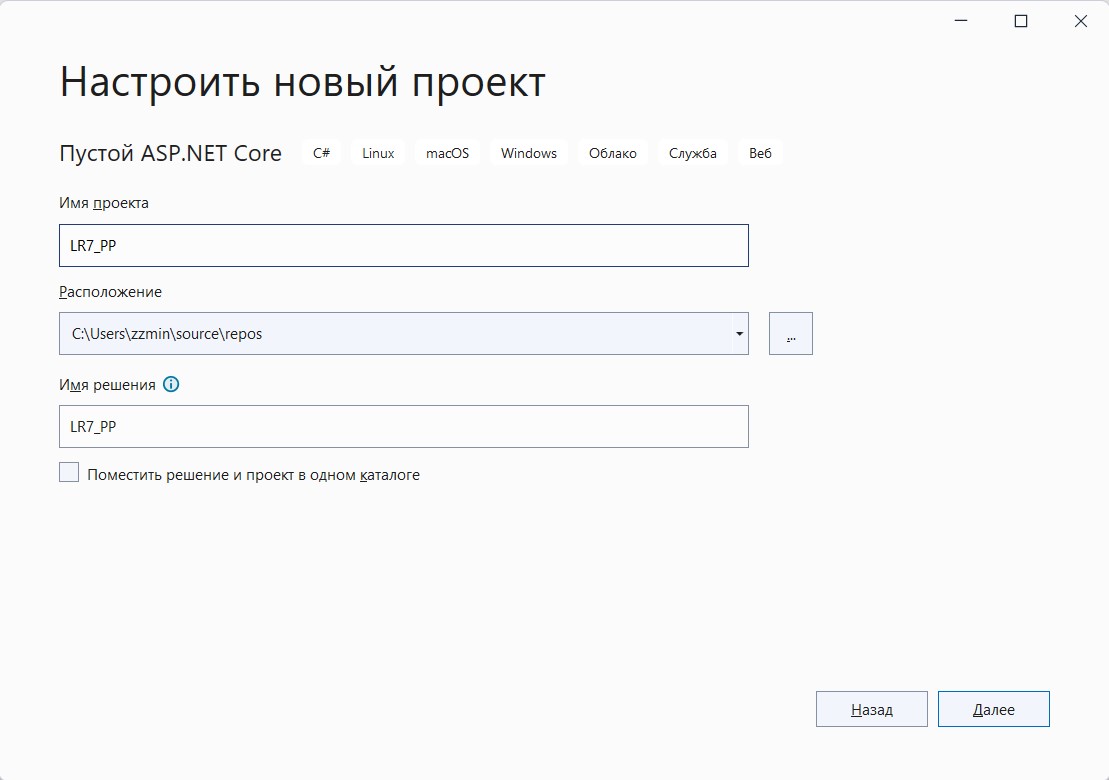
•

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.

1. 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:

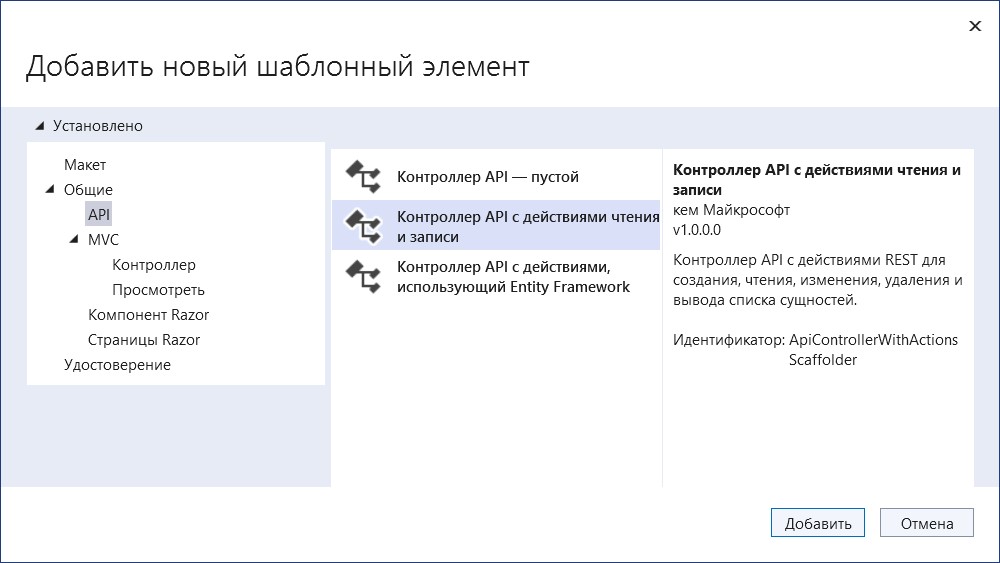
|  |  |
| --- | --- |
| 1 | public class Hotel |
| 2 | { |
| 3 | public int Id { get; set; } |
| 4 | public string Name { get; set; } |
| 5 | public int CountOfStars { get; set; } |
| 6 | public string CountryCode { get; set; } |
| 7 | public string Description { get; set; } |
| 8 | } |

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

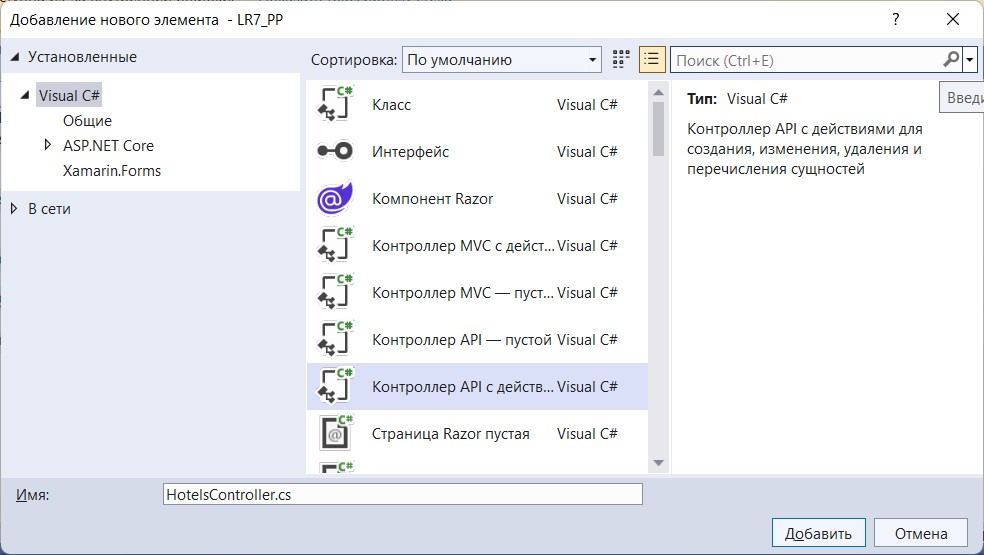
Также добавим в папку **Models** новый класс HotelsContext для взаимодействия с базой данных:

|  |  |
| --- | --- |
| 1 | using Microsoft.EntityFrameworkCore; |
| 2 | namespace LR7\_PP.Models |
| 3 | { |
| 4 | public class HotelsContext : DbContext |
| 5 | { |
| 6 | public DbSet<Hotel> Hotel { get; set; } |
| 7 | public HotelsContext (DbContextOptions<HotelsContext> options) : base(options) |
| 8 | { |
| 9 | Database.EnsureCreated(); |
| 10 | } |
| 11 | } |
| 12 | } |

1. Далее добавим в проект новую папку **Controllers**, а в ней создадим новый apiконтроллер. 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 | using LR7\_PP.Models; |
| 2 | using Microsoft.AspNetCore.Mvc; |
| 3 | using Microsoft.EntityFrameworkCore; |
| 4 | namespace LR7\_PP.Controllers |
| 5 | { |
| 6 | [Route("api/[controller]")] |
| 7 | [ApiController] |
| 8 | public class HotelsController : ControllerBase |
| 9 | { |
| 10 | private HotelsContext? \_db; |
| 11 | public HotelsController(HotelsContext hotelsContext) |

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

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.

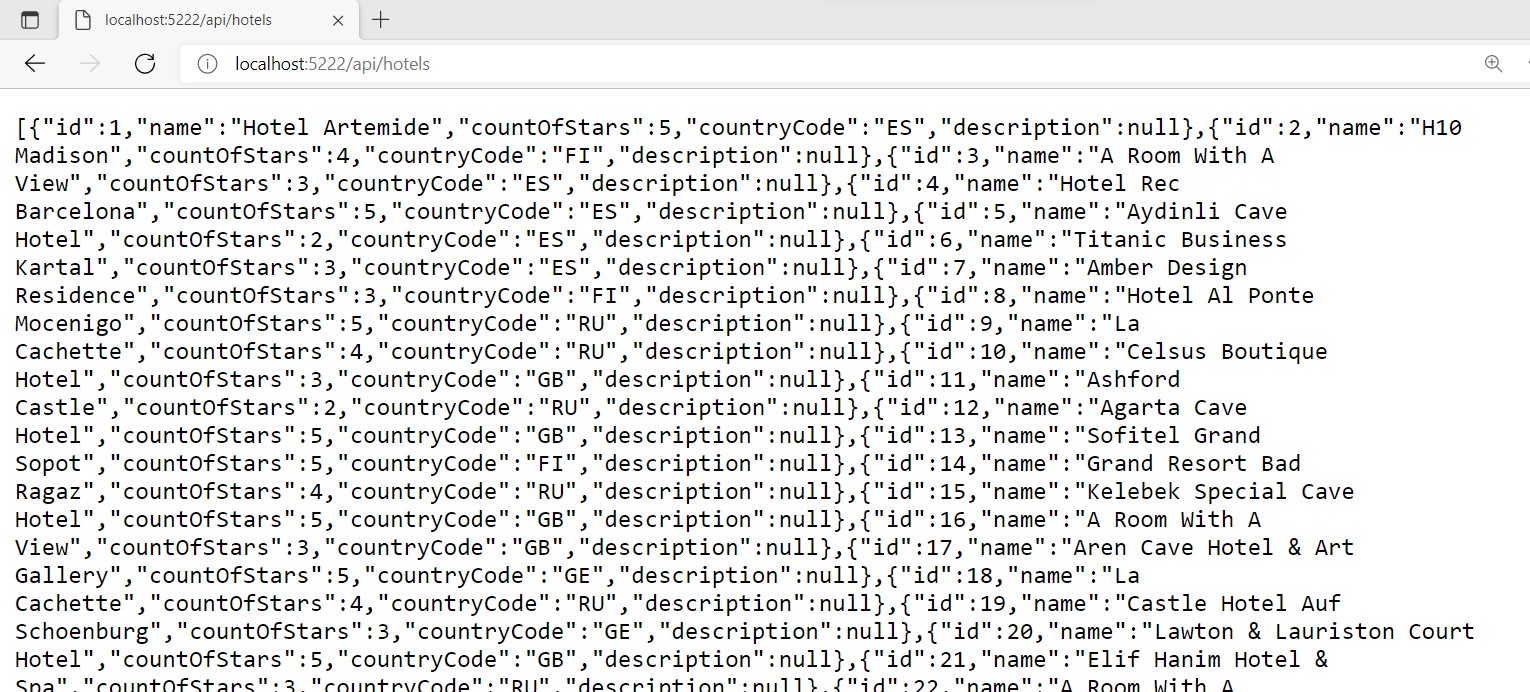
5. Теперь, чтобы это все использовать, изменим код класса Programs:

|  |  |
| --- | --- |
|  | using Microsoft.EntityFrameworkCore; |
|  |  |
|  | var builder = WebApplication.CreateBuilder(args); |
|  | string connection = builder.Configuration.GetConnectionString("DefaultConnection"); |
|  |  |
|  | builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(connection)); |
|  |  |
|  | var app = builder.Build(); |
|  | app.UseDefaultFiles(); |
|  | app.UseStaticFiles(); |
|  |  |
|  | app.MapGet("/api/hotels", async (ApplicationDbContext db) => await db.Hotel.ToListAsync()); |
|  | app.MapGet("/api/hotels/{id:int}", async (int id, ApplicationDbContext db) => |
|  | { |
|  | // получаем пользователя по id |
|  | Hotel? hotel = await db.Hotel.FirstOrDefaultAsync(u => u.Id == id); |
|  |  |
|  | // если не найден, отправляем статусный код и сообщение об ошибке |
|  | if (hotel == null) return Results.NotFound(new { message = "Отель не найден" }); |
|  |  |
|  | // если пользователь найден, отправляем его |
|  | return Results.Json(hotel); |
|  | }); |
|  |  |
|  | app.MapDelete("/api/hotels/{id:int}", async (int id, ApplicationDbContext db) => |
|  | { |
|  | // получаем пользователя по id |
|  | Hotel? hotel = await db.Hotel.FirstOrDefaultAsync(u => u.Id == id); |
|  |  |
|  | // если не найден, отправляем статусный код и сообщение об ошибке |
|  | if (hotel == null) return Results.NotFound(new { message = "Отель не найден" }); |
|  |  |
|  | // если пользователь найден, удаляем его |
|  | db.Hotel.Remove(hotel); |
|  | await db.SaveChangesAsync(); |
|  | return Results.Json(hotel); |
|  | }); |
|  |  |
|  | app.MapPost("/api/hotels", async (Hotel hotel, ApplicationDbContext db) => |
|  | { |
|  | // добавляем пользователя в массив |
|  | await db.Hotel.AddAsync(hotel); |
|  | await db.SaveChangesAsync(); |
|  | return hotel; |
|  | }); |
|  |  |
|  | app.MapPut("/api/hotels", async (Hotel HotelData, ApplicationDbContext db) => |
|  | { |
|  | // получаем пользователя по id |
|  | var hotel = await db.Hotel.FirstOrDefaultAsync(u => u.Id == HotelData.Id); |
|  |  |
|  | // если не найден, отправляем статусный код и сообщение об ошибке |
|  | if (hotel == null) return Results.NotFound(new { message = "Отель не найден" }); |
|  |  |
|  | // если пользователь найден, изменяем его данные и отправляем обратно клиенту |
|  | hotel.Name = HotelData.Name; |
|  | hotel.CountOfStars = HotelData.CountOfStars; |
|  | hotel.CountryCode = HotelData.CountryCode; |
|  | await db.SaveChangesAsync(); |
|  | return Results.Json(hotel); |
|  | }); |
|  |  |
|  |  |
|  |  |
|  | app.Run(); |
|  | var builder = WebApplication.CreateBuilder(args); |
|  | string connection = builder.Configuration.GetConnectionString("DefaultConnection"); |
|  |  |
|  | builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(connection)); |
|  |  |

Далее изменим appsettings.json:

|  |  |
| --- | --- |
|  | { |
|  | "ConnectionStrings": { |
|  | "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=ToursDB;Trusted\_Connection=True;" |
|  | }, |
|  | "Logging": { |
|  | "LogLevel": { |
|  | "Default": "Information", |
|  | "Microsoft.AspNetCore": "Warning" |
|  | } |
|  | }, |
|  | "AllowedHosts": "\*" |
|  | } |

Запустим приложение и обратимся по пути api/hotels:

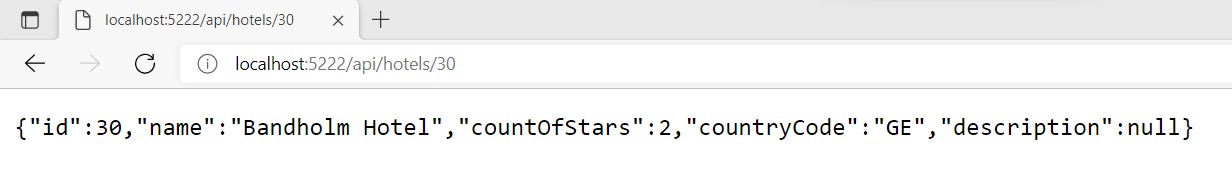


Поскольку запрос из адресной строки браузера представляет GET-запрос, то его будет обрабатывать метод

|  |
| --- |
| // 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:

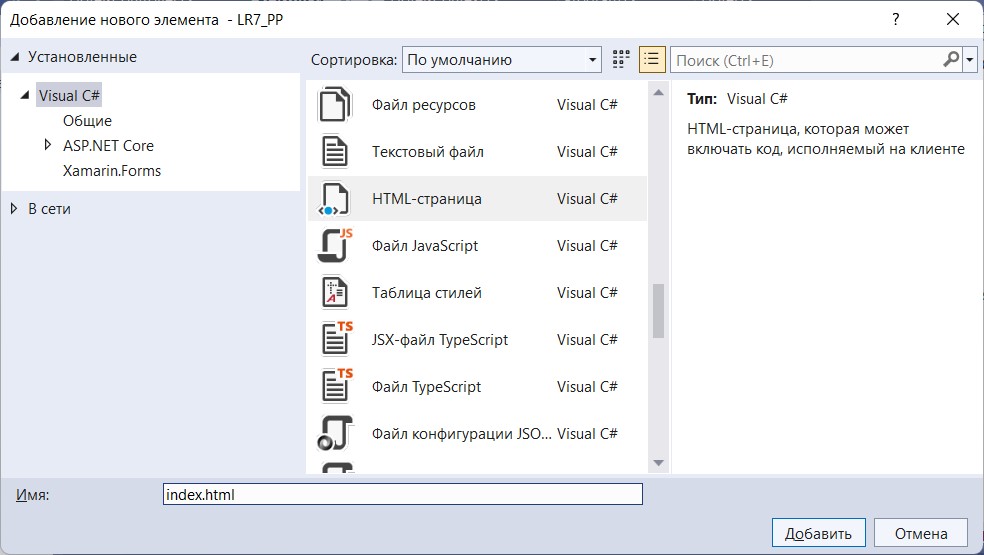


Поскольку это также запрос типа GET, но теперь также передается параметр id, то сработает следующий метод:

|  |
| --- |
| [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 вебстраницы 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 JavaScrip**t, 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:

|  |  |
| --- | --- |
|  | <!DOCTYPE html> |
|  |  |
|  | <html> |
|  |  |
|  | <head> |
|  |  |
|  | <meta charset="utf-8" /> |
|  |  |
|  | <meta name="viewport" content="width=device-width" /> |
|  |  |
|  | <title>Список отелей</title> |
|  |  |
|  | <link href="https://stackpath.bootstrapcdn.com/ |
|  | bootstrap/4.4.0/css/bootstrap.min.css" rel="stylesheet" /> |
|  |  |
|  | </head> |
|  |  |
|  | <body> |
|  | <h2>Список Отелей</h2> |
|  | <div class="form-group col-md-5"> |
|  | <input type="hidden" id="HotelId" /> |
|  | <p> |
|  | Название отеля:<br /> |
|  | <input class="form-control" n id="HotelName" /> |
|  | </p> |
|  | <p> |
|  | Количество звёзд:<br /> |
|  | <input class="form-control" n id="HotelCount" type="number" /> |
|  | </p> |
|  | <p> |
|  | Код страны:<br /> |
|  | <input class="form-control" n id="HotelCode" /> |
|  | </p> |
|  | <p> |
|  | <button class="btn btn-primary" id="saveBtn">Сохранить</button> |
|  | <button class="btn btn-primary" id="resetBtn">Сбросить</button> |
|  | </p> |
|  | </div> |
|  | <table class="table table-condensed table-striped col-md-6"> |
|  | <thead> |
|  | <tr><th>ID</th><th>Название отеля</th><th>Количество звёзд</th><th>Код страны</th><th></th></tr></thead> |
|  | <tbody> |
|  | </tbody> |
|  | </table> |
|  | <script src="script.js"></script> |
|  | <div>2024 Борлаков Мухамат</div> |
|  |  |
|  | </body> |
|  |  |
|  | </html> |

Изменим файл script.js:

|  |  |
| --- | --- |
|  | async function getHotels() { |
|  | // отправляет запрос и получаем ответ |
|  | const response = await fetch("/api/hotels", { |
|  | method: "GET", |
|  | headers: { "Accept": "application/json" } |
|  | }); |
|  | // если запрос прошел нормально |
|  | if (response.ok === true) { |
|  | // получаем данные |
|  | const hotels = await response.json(); |
|  | const rows = document.querySelector("tbody"); |
|  | // добавляем полученные элементы в таблицу |
|  | hotels.forEach(hotel => rows.append(row(hotel))); |
|  | } |
|  | } |
|  | // Получение одного пользователя |
|  | async function getHotel(id) { |
|  | const response = await fetch(`/api/hotels/${id}`, { |
|  | method: "GET", |
|  | headers: { "Accept": "application/json" } |
|  | }); |
|  | if (response.ok === true) { |
|  | const hotel = await response.json(); |
|  | document.getElementById("HotelId").value = hotel.id; |
|  | document.getElementById("HotelName").value = hotel.name; |
|  | document.getElementById("HotelCount").value = hotel.countOfStars; |
|  | document.getElementById("HotelCode").value = hotel.countryCode; |
|  | } |
|  | else { |
|  | // если произошла ошибка, получаем сообщение об ошибке |
|  | const error = await response.json(); |
|  | console.log(error.message); // и выводим его на консоль |
|  | } |
|  | } |
|  | // Добавление пользователя |
|  | async function createHotel(HotelName, HotelCount, HotelCode) { |
|  |  |
|  | const response = await fetch("api/hotels", { |
|  | method: "POST", |
|  | headers: { "Accept": "application/json", "Content-Type": "application/json" }, |
|  | body: JSON.stringify({ |
|  | name: HotelName, |
|  | countOfStars: parseInt(HotelCount, 10), |
|  | countryCode: HotelCode |
|  | }) |
|  | }); |
|  | if (response.ok === true) { |
|  | const hotels = await response.json(); |
|  | document.querySelector("tbody").append(row(hotels)); |
|  | } |
|  | else { |
|  | const error = await response.json(); |
|  | console.log(error.message); |
|  | } |
|  | } |
|  | // Изменение пользователя |
|  | async function editHotel(HotelId, HotelName, HotelCount, HotelCode) { |
|  | const response = await fetch("api/hotels", { |
|  | method: "PUT", |
|  | headers: { "Accept": "application/json", "Content-Type": "application/json" }, |
|  | body: JSON.stringify({ |
|  | id: HotelId, |
|  | name: HotelName, |
|  | countOfStars: parseInt(HotelCount, 10), |
|  | countryCode: HotelCode |
|  | }) |
|  | }); |
|  | if (response.ok === true) { |
|  | const hotel = await response.json(); |
|  | document.querySelector(`tr[data-rowid='${hotel.id}']`).replaceWith(row(hotel)); |
|  | } |
|  | else { |
|  | const error = await response.json(); |
|  | console.log(error.message); |
|  | } |
|  | } |
|  | // Удаление пользователя |
|  | async function deleteHotel(id) { |
|  | const response = await fetch(`/api/hotels/${id}`, { |
|  | method: "DELETE", |
|  | headers: { "Accept": "application/json" } |
|  | }); |
|  | if (response.ok === true) { |
|  | const hotel = await response.json(); |
|  | document.querySelector(`tr[data-rowid='${hotel.id}']`).remove(); |
|  | } |
|  | else { |
|  | const error = await response.json(); |
|  | console.log(error.message); |
|  | } |
|  | } |
|  |  |
|  | // сброс данных формы после отправки |
|  | function reset() { |
|  | document.getElementById("HotelId").value = |
|  | document.getElementById("HotelName").value = |
|  | document.getElementById("HotelCode").value = |
|  | document.getElementById("HotelCount").value = ""; |
|  | } |
|  | // создание строки для таблицы |
|  | function row(hotel) { |
|  |  |
|  | const tr = document.createElement("tr"); |
|  | tr.setAttribute("data-rowid", hotel.id); |
|  |  |
|  | const IdTd = document.createElement("td"); |
|  | IdTd.append(hotel.id); |
|  | tr.append(IdTd); |
|  |  |
|  | const nameTd = document.createElement("td"); |
|  | nameTd.append(hotel.name); |
|  | tr.append(nameTd); |
|  |  |
|  | const countTd = document.createElement("td"); |
|  | countTd.append(hotel.countOfStars); |
|  | tr.append(countTd); |
|  |  |
|  | const codeTd = document.createElement("td"); |
|  | codeTd.append(hotel.countryCode); |
|  | tr.append(codeTd); |
|  |  |
|  | const linksTd = document.createElement("td"); |
|  |  |
|  | const editLink = document.createElement("a"); |
|  | editLink.setAttribute("data-id", hotel.id); |
|  | editLink.setAttribute("style", "cursor:pointer;padding:15px;"); |
|  | editLink.append("Изменить"); |
|  | editLink.addEventListener("click", async () => await getHotel(hotel.id)); |
|  | linksTd.append(editLink); |
|  |  |
|  | const removeLink = document.createElement("a"); |
|  | removeLink.setAttribute("data-id", hotel.id); |
|  | removeLink.setAttribute("style", "cursor:pointer;padding:15px;"); |
|  | removeLink.append("Удалить"); |
|  | removeLink.addEventListener("click", async () => await deleteHotel(hotel.id)); |
|  |  |
|  | linksTd.append(removeLink); |
|  | tr.appendChild(linksTd); |
|  |  |
|  | return tr; |
|  | } |
|  | // сброс значений формы |
|  | document.getElementById("resetBtn").addEventListener("click", () => reset()); |
|  |  |
|  | // отправка формы |
|  | document.getElementById("saveBtn").addEventListener("click", async () => { |
|  |  |
|  | const id = document.getElementById("HotelId").value; |
|  | const name = document.getElementById("HotelName").value; |
|  | const count = document.getElementById("HotelCount").value; |
|  | const code = document.getElementById("HotelCode").value; |
|  | if (id === "") |
|  | await createHotel(name, count, code); |
|  | else |
|  | await editHotel(id, name, count, code); |
|  | reset(); |
|  | }); |
|  |  |
|  | getHotels(); |

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 промис[[1]](#footnote-1), 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, выполнился 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

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.

1. Промис — это объект, представляющий результат успешного или неудачного завершения асинхронной операции. [↑](#footnote-ref-1)