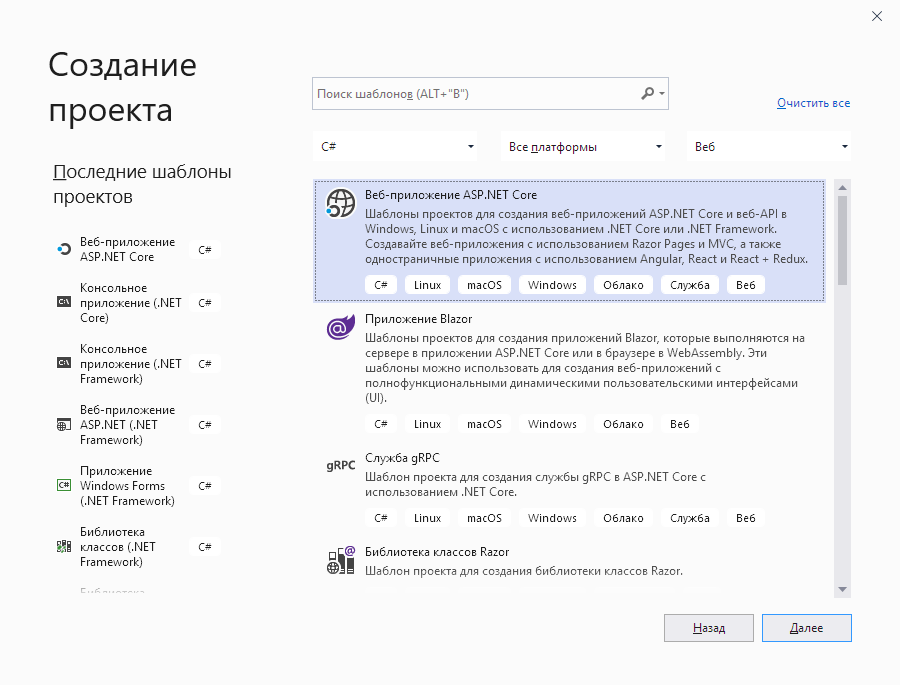
# Разработка Web API «Список дел»

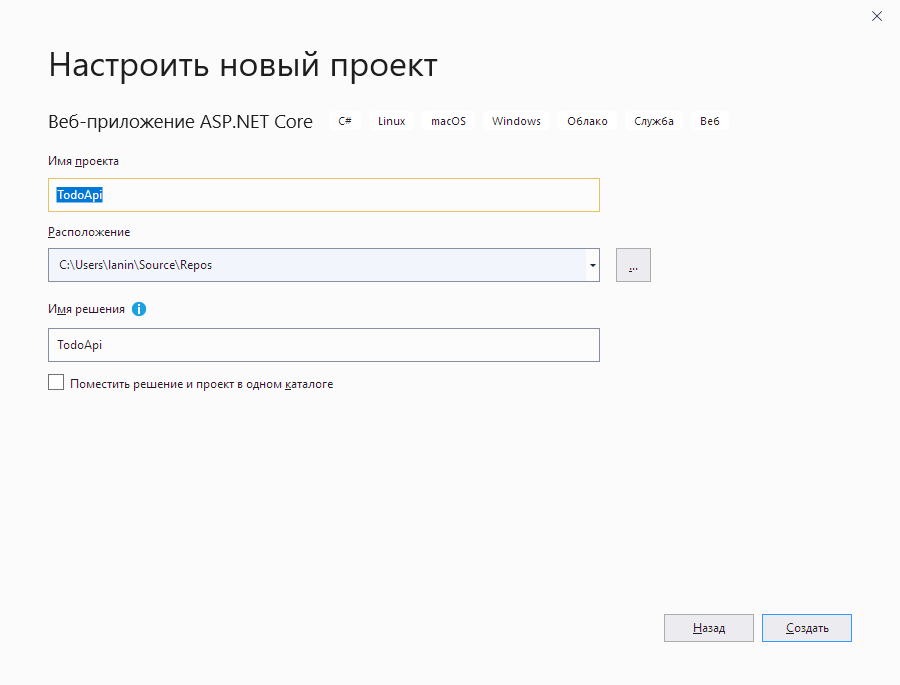
1. Убедитесь, что установлены компоненты SQL Server Express LocalDB (может быть установлен вместе с Visual Studio) и SQL Server Management Studio (SSMS) (для удобства работы с базой)

При необходимости установки обратитесь к документации:

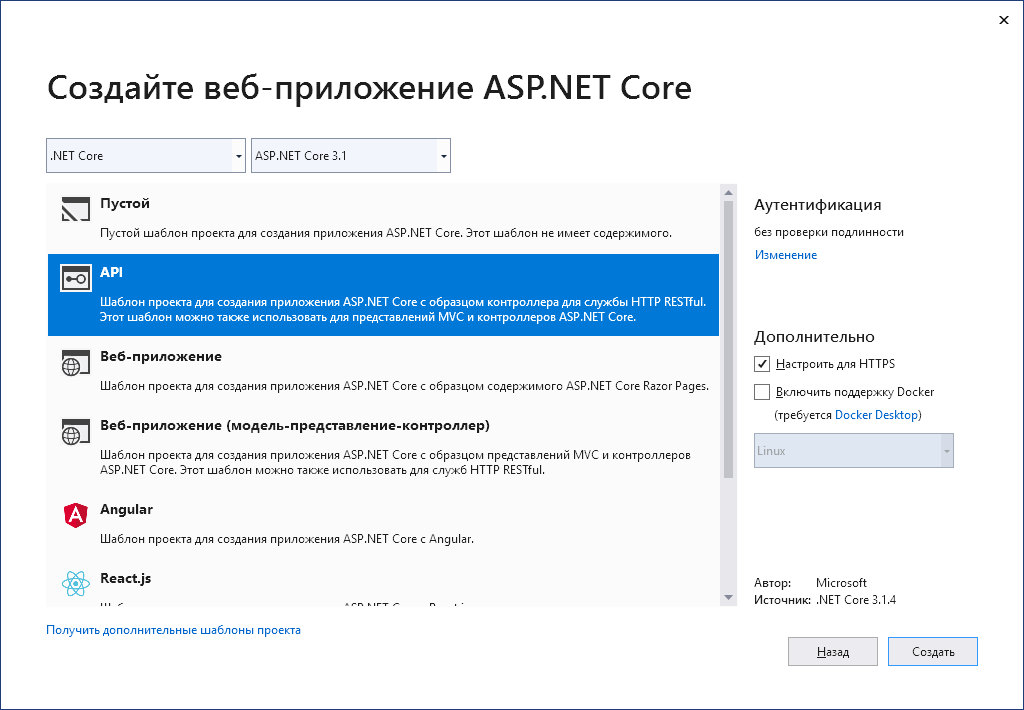
* SQL Server Express LocalDB <https://docs.microsoft.com/ru-ru/sql/database-engine/configure-windows/sql-server-express-localdb?view=sql-server-ver15>
* SQL Server Management Studio (SSMS) <https://docs.microsoft.com/ru-RU/sql/ssms/download-sql-server-management-studio-ssms?view=aps-pdw-2016>

1. Убедитесь, что установлена программа Postman, при необходимости установите ее: <https://www.postman.com/downloads/>.
2. В Visual Studio cоздайте проект «Веб-приложение ASP.NET Core», назвав его TodoApi.

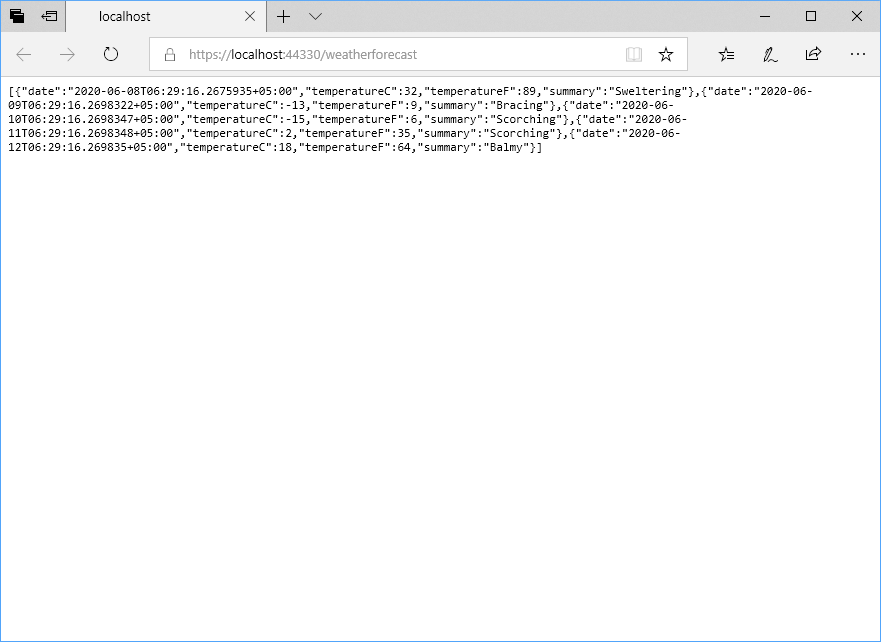




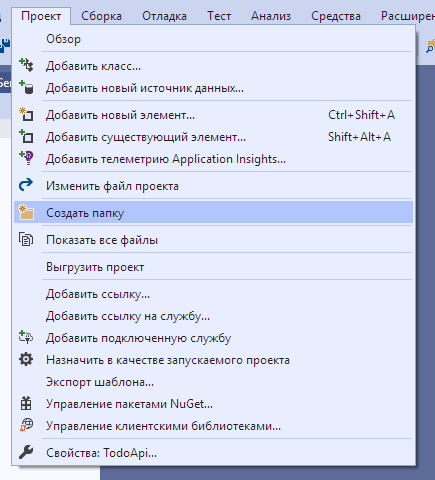
1. В диалоговом окне Создание веб-приложения ASP.NET Core убедитесь в том, что выбраны платформы .NET Core и ASP.NET Core 3.1. Выберите шаблон API и нажмите кнопку «Создать».



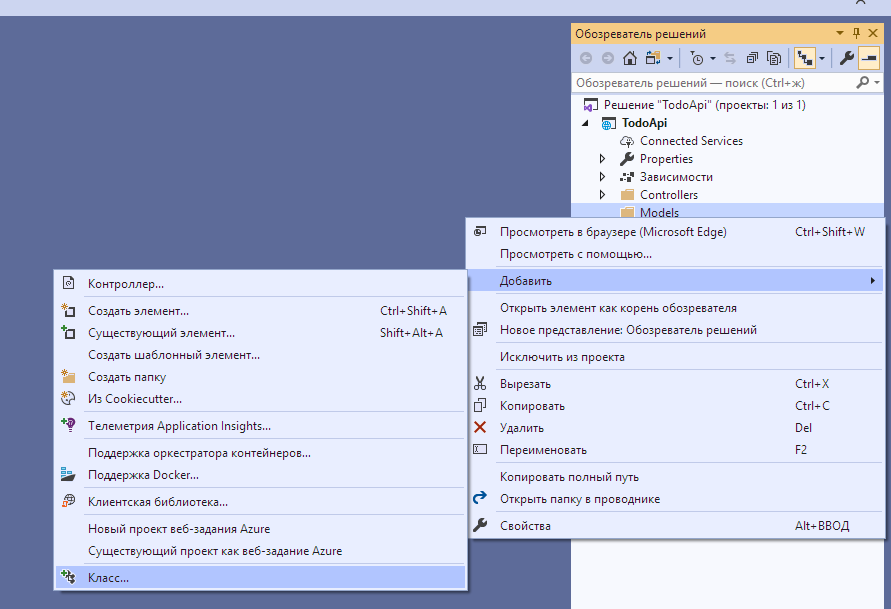
1. Убедитесь, что созданное по шаблону тестовое приложение работает, запустив его на выполнение.

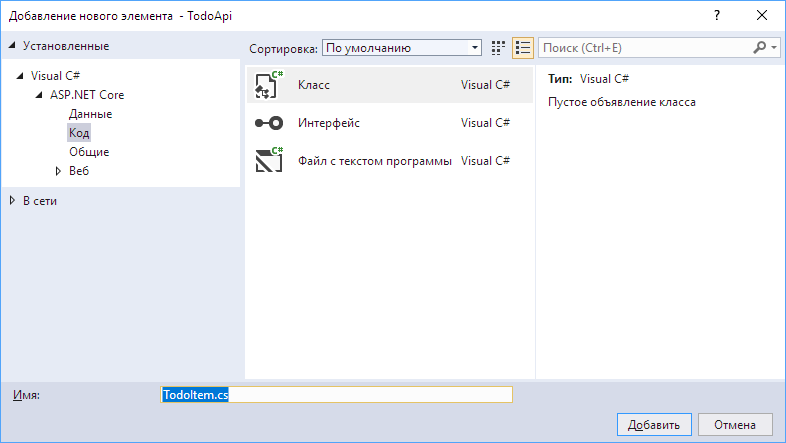


1. Вызовите пункт меня «Проект\Создать папку» и создайте папку «Models».



1. Щелкните папку Models правой кнопкой мыши и выберите «Добавить\Класс». Присвойте классу имя TodoItem.





1. Опишите поля класса TodoItem.

namespace TodoApi.Models

{

public class TodoItem

{

public long Id { get; set; }

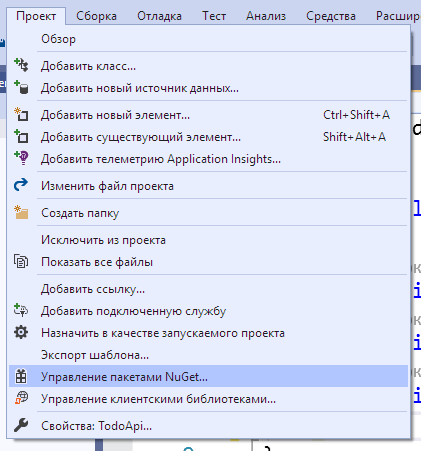
public string Name { get; set; }

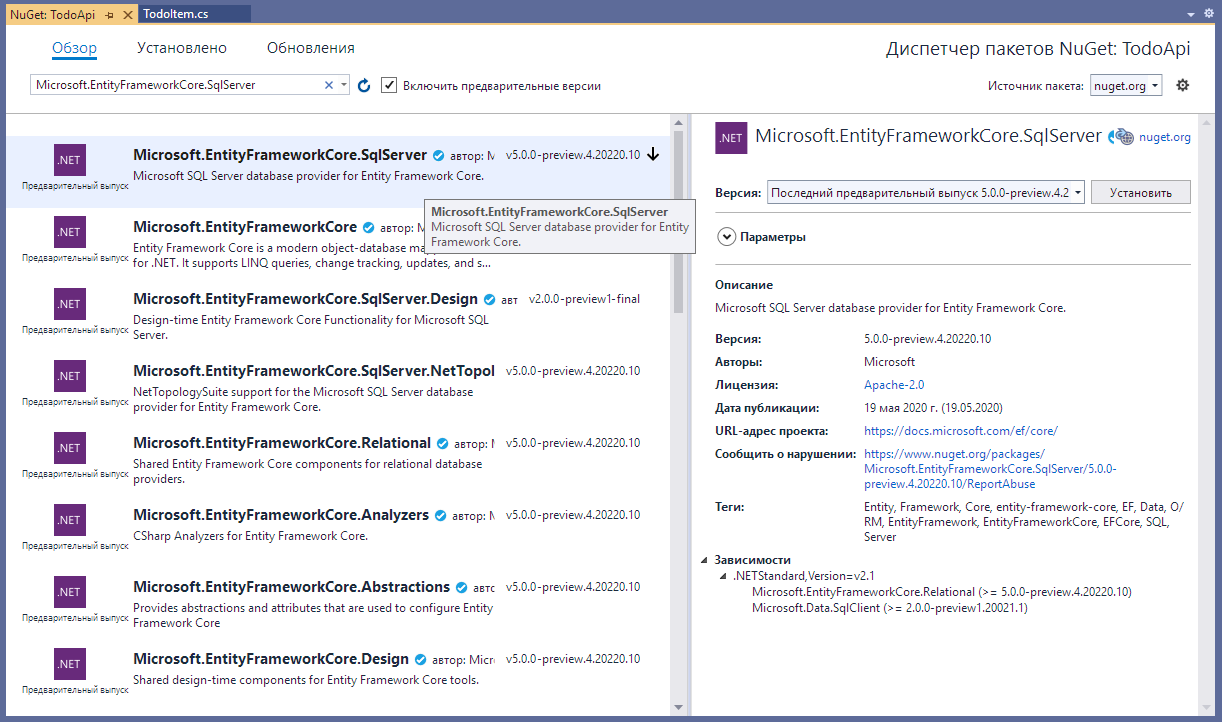
public bool IsComplete { get; set; }

}

}

1. Выполните команду «Проект\Управление пакетами NuGet…». Перейдите на вкладку Обзор и введите «Microsoft.EntityFrameworkCore.SqlServer» в поле поиска. На панели слева выберите Microsoft.EntityFrameworkCore.SqlServer и нижмите кнопку «Установить».





1. Щелкните папку Models правой кнопкой мыши и выберите «Добавить\Класс». Назовите класс TodoContext.

using Microsoft.EntityFrameworkCore;

namespace TodoApi.Models

{

public class TodoContext : DbContext

{

public DbSet<TodoItem> TodoItems { get; set; }

public TodoContext(DbContextOptions<TodoContext> options)

: base(options)

{

Database.EnsureCreated();

}

}

}

1. Добавьте в файл appsettings.json информацию о строке подключения к базе данных.

{

"ConnectionStrings": {

"ToDoContext": "Server=(localdb)\\mssqllocaldb;Database=ToDoDB;Trusted\_Connection=True;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

1. Добавьте в файл Startup.cs выделенный код. Используя сочетание клавиш «Alt+Enter» добавьте необходимые инструкции using.

public void ConfigureServices(IServiceCollection services)

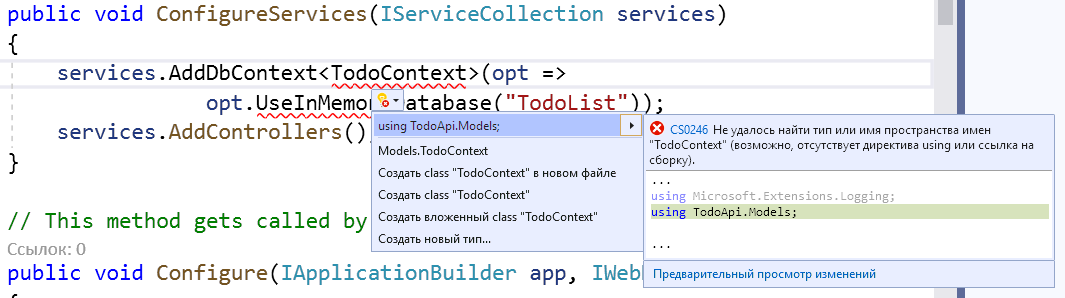
{

string connectionString = Configuration.GetConnectionString("ToDoContext");

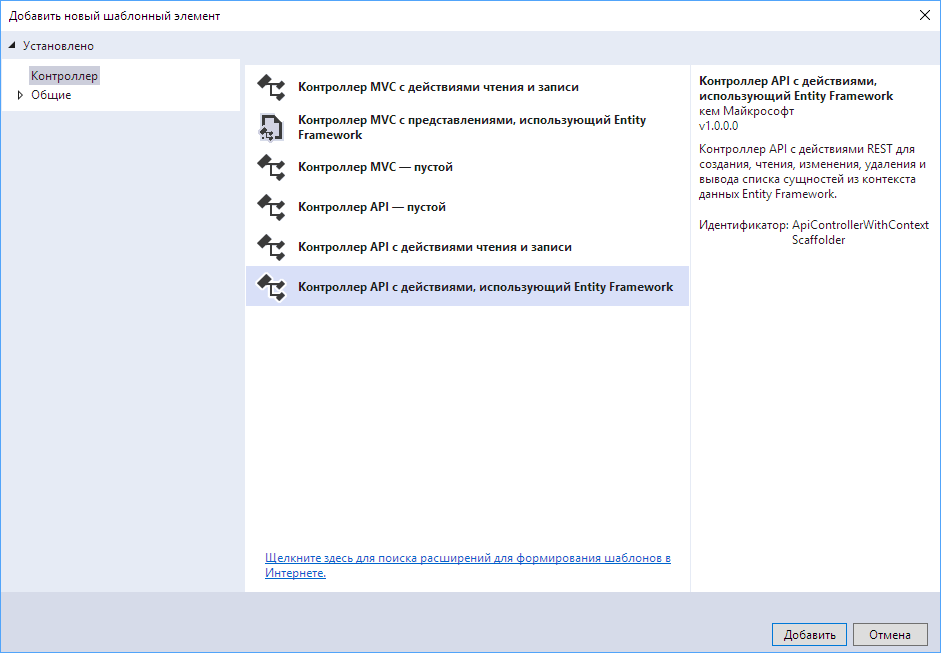
services.AddDbContext<TodoContext>(op=>op.UseSqlServer(connectionString));

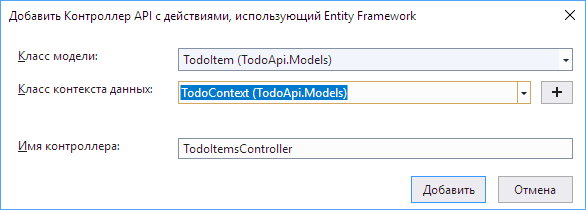
services.AddControllersWithViews();

}



1. Щелкните папку Controllers правой кнопкой мыши. Щелкните «Добавить\Контроллер…». Выберите Контроллер API с действиями, использующий Entity Framework. В диалоговом окне «Контроллер API с действиями, использующий Entity Framework» выберите TodoItem (TodoApi.Models) в поле Класс модели, выберите TodoContext (TodoApi.Models) в поле Класс контекста данных.

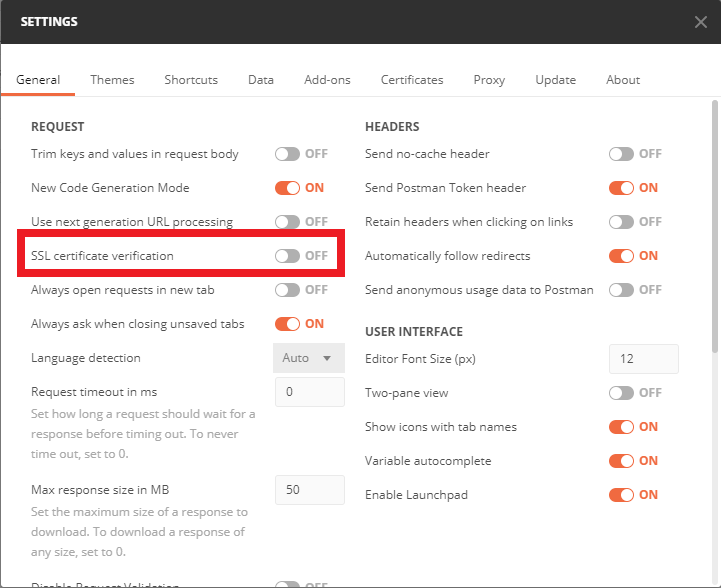




1. Изучите сгенерированный код контроллера.

# Тестрование web-api

1. Запустите Postman. В меню «Файл\Параметры» (вкладка General) отключите параметр Проверка SSL-сертификата.



1. Для тестирования метода Post в приложения в Postman
2. Создайте новый запрос.
3. Установите HTTP-метод POST.
4. Откройте вкладку Body.
5. Установите переключатель raw.
6. Задайте тип JSON.
7. В теле запроса введите код JSON для элемента списка дел:

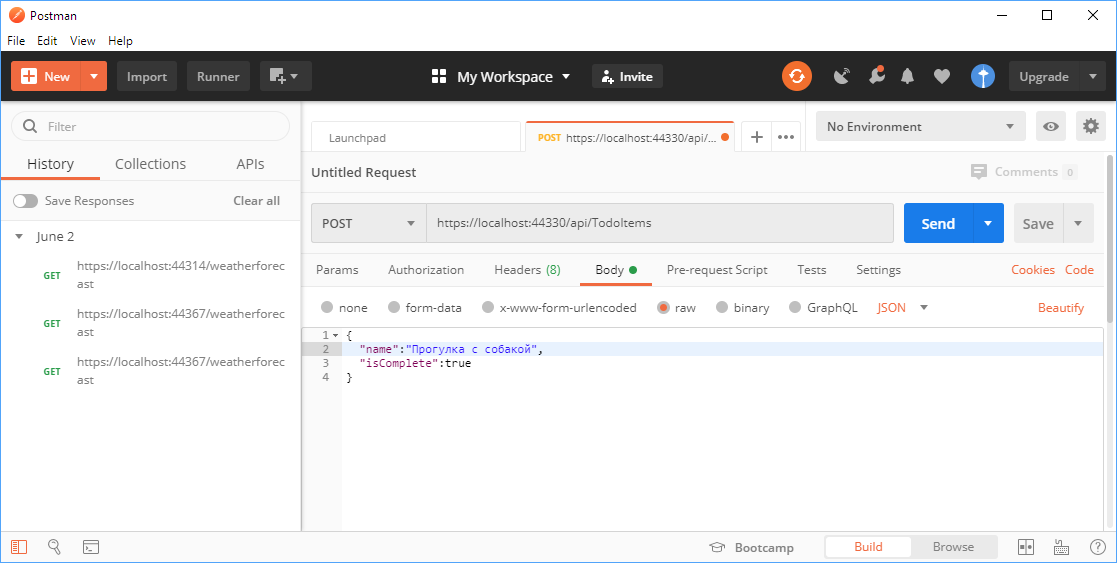
{

"name":"walk dog",

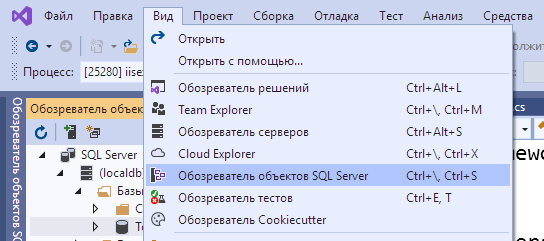
"isComplete":true

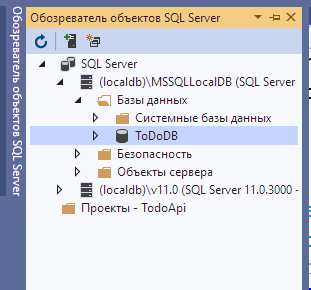
}

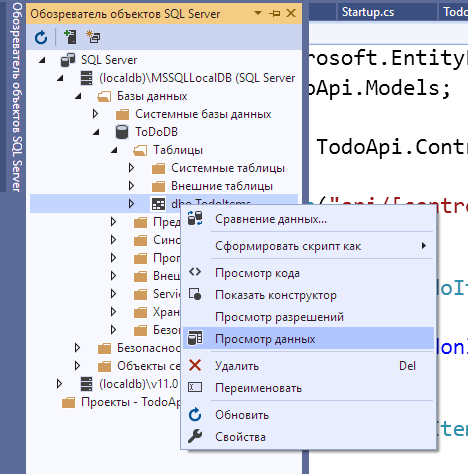
1. Нажмите кнопку Отправить.

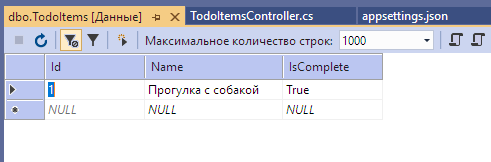


1. Выполните команду «Вид\Обозреватель объектов SQL Server» для отображения доступных экземпляров SQL Server. Убедитесь, что база данных создана и таблица TodoItems содержит данные.









1. Протестируйте остальные методы разработанного API.

# Разработка web-клиента

1. Вставьте в метод Configure в файле Startup.cs следующий выделенный код:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseDefaultFiles();

app.UseStaticFiles();

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

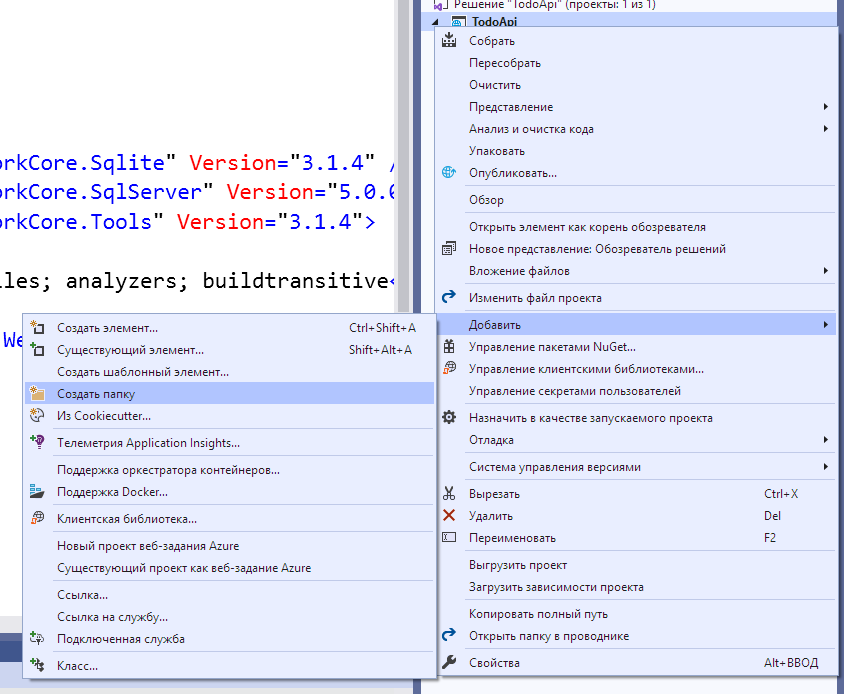
{

endpoints.MapControllers();

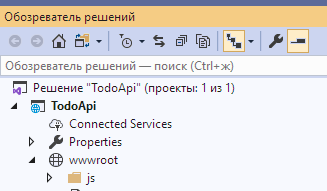
});

}

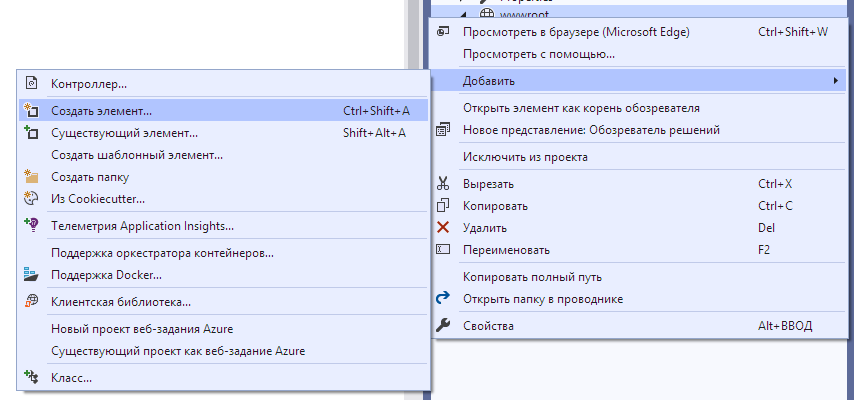
1. Создайте папку wwwroot в корневом каталоге проекта.

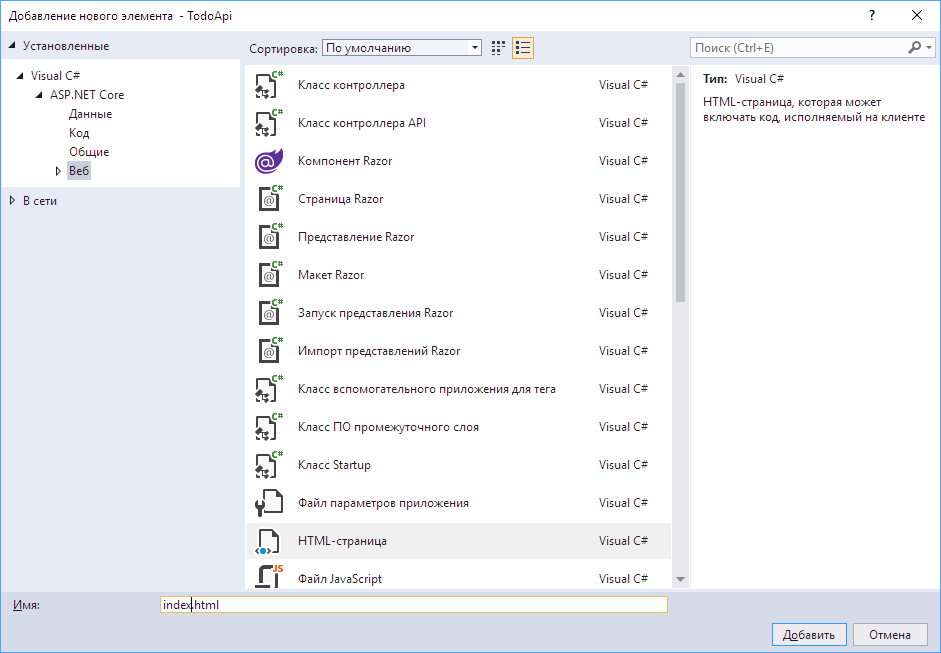


1. Создайте папку js в папке wwwroot.



1. Добавьте HTML-файл index.html в папку wwwroot.





1. Замените содержимое файла index.html следующей разметкой.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>To-do CRUD</title>

<link rel="stylesheet" href="css/site.css" />

</head>

<body>

<h1>To-do CRUD</h1>

<h3>Add</h3>

<form action="javascript:void(0);" method="POST" onsubmit="addItem()">

<input type="text" id="add-name" placeholder="New to-do">

<input type="submit" value="Add">

</form>

<div id="editForm">

<h3>Edit</h3>

<form action="javascript:void(0);" onsubmit="updateItem()">

<input type="hidden" id="edit-id">

<input type="checkbox" id="edit-isComplete">

<input type="text" id="edit-name">

<input type="submit" value="Save">

<a onclick="closeInput()" aria-label="Close">&#10006;</a>

</form>

</div>

<p id="counter"></p>

<table>

<tr>

<th>Is Complete?</th>

<th>Name</th>

<th></th>

<th></th>

</tr>

<tbody id="todos"></tbody>

</table>

<script src="js/site.js" asp-append-version="true"></script>

<script type="text/javascript">

getItems();

</script>

</body>

</html>

1. Добавьте файл JavaScript с именем *site.js* в папку *wwwroot/js*. Замените содержимое файла *site.js* следующим кодом:

const uri = 'api/TodoItems';

let todos = [];

function getItems() {

fetch(uri)

.then(response => response.json())

.then(data => \_displayItems(data))

.catch(error => console.error('Unable to get items.', error));

}

function addItem() {

const addNameTextbox = document.getElementById('add-name');

const item = {

isComplete: false,

name: addNameTextbox.value.trim()

};

fetch(uri, {

method: 'POST',

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json'

},

body: JSON.stringify(item)

})

.then(response => response.json())

.then(() => {

getItems();

addNameTextbox.value = '';

})

.catch(error => console.error('Unable to add item.', error));

}

function deleteItem(id) {

fetch(`${uri}/${id}`, {

method: 'DELETE'

})

.then(() => getItems())

.catch(error => console.error('Unable to delete item.', error));

}

function displayEditForm(id) {

const item = todos.find(item => item.id === id);

document.getElementById('edit-name').value = item.name;

document.getElementById('edit-id').value = item.id;

document.getElementById('edit-isComplete').checked = item.isComplete;

document.getElementById('editForm').style.display = 'block';

}

function updateItem() {

const itemId = document.getElementById('edit-id').value;

const item = {

id: parseInt(itemId, 10),

isComplete: document.getElementById('edit-isComplete').checked,

name: document.getElementById('edit-name').value.trim()

};

fetch(`${uri}/${itemId}`, {

method: 'PUT',

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json'

},

body: JSON.stringify(item)

})

.then(() => getItems())

.catch(error => console.error('Unable to update item.', error));

closeInput();

return false;

}

function closeInput() {

document.getElementById('editForm').style.display = 'none';

}

function \_displayCount(itemCount) {

const name = (itemCount === 1) ? 'to-do' : 'to-dos';

document.getElementById('counter').innerText = `${itemCount} ${name}`;

}

function \_displayItems(data) {

const tBody = document.getElementById('todos');

tBody.innerHTML = '';

\_displayCount(data.length);

const button = document.createElement('button');

data.forEach(item => {

let isCompleteCheckbox = document.createElement('input');

isCompleteCheckbox.type = 'checkbox';

isCompleteCheckbox.disabled = true;

isCompleteCheckbox.checked = item.isComplete;

let editButton = button.cloneNode(false);

editButton.innerText = 'Edit';

editButton.setAttribute('onclick', `displayEditForm(${item.id})`);

let deleteButton = button.cloneNode(false);

deleteButton.innerText = 'Delete';

deleteButton.setAttribute('onclick', `deleteItem(${item.id})`);

let tr = tBody.insertRow();

let td1 = tr.insertCell(0);

td1.appendChild(isCompleteCheckbox);

let td2 = tr.insertCell(1);

let textNode = document.createTextNode(item.name);

td2.appendChild(textNode);

let td3 = tr.insertCell(2);

td3.appendChild(editButton);

let td4 = tr.insertCell(3);

td4.appendChild(deleteButton);

});

todos = data;

}

1. Убедитесь в работоспособности приложения по адресу https://localhost:<номер порта>/index.html.

