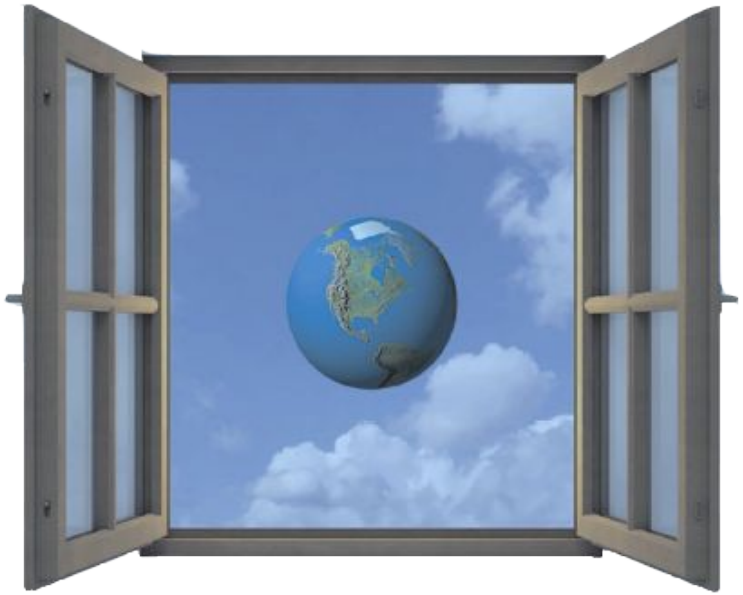
Введение в Web API

Введение в MVC

Web API = API через Web (HTTP)

API (Application Programming Interface – программный интерфейс приложения) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой. Обычно входит в описание какого-либо интернет-протокола (например, RFC), программного каркаса (фреймворка) или стандарта вызовов функций операционной системы. Часто реализуется отдельной программной библиотекой или сервисом операционной системы. Используется программистами при написании всевозможных приложений.

API – окно в мир (ключ к автоматизированному использованию вашего сервиса)



https://docs.microsoft.com/ru-ru/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio

REST

REST (Representational State Transfer - «передача состояния представления») - архитектурный стиль взаимодействия компонентов распределённого приложения в сети (термин введен Roy Thomas Fielding). REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой системы, в определённых случаях приводящих к повышению производительности и упрощению архитектуры.
В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов в WWW.
REST является альтернативой другим вариантам RPC (SOAP, COM+, etc…).

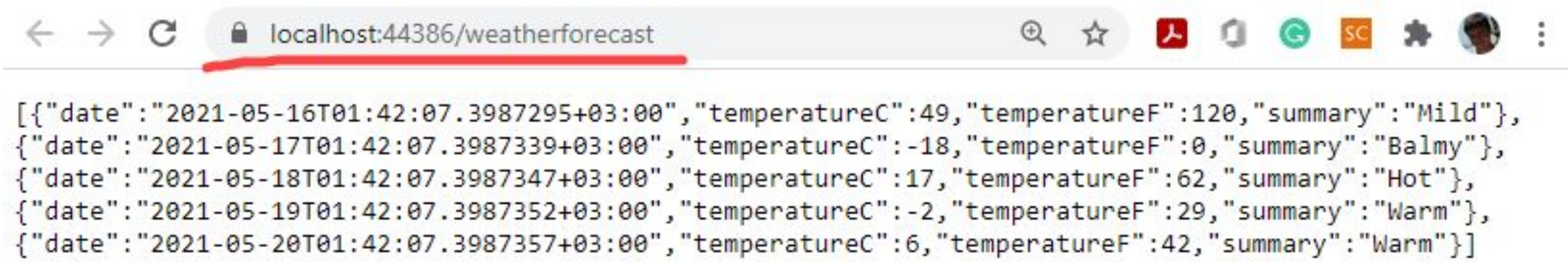
Требования к архитектуре REST:

- Модель клиент-сервер.
- Отсутствие состояния.
- Кэширование.
- Единообразие интерфейса.
- Слои.
- Код по требованию (необязательное ограничение).

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Задача 1. Web API и список товаров

Создайте с помощью мастера в Visual Studio новый проект Web API (ASP.Net Core Web API). Изучите проект, в нем уже есть контроллер и при запуске можно обратиться к нему по относительному пути /WeatherForecast/, получив JSON:



Вам необходимо внести изменения в проект, добавив возможность работы с товарами через Web API.
Для этого создайте в папке Controllers новый контроллер с именем `ProductsController`, осуществляющий работу со списком товаров, добавив перед классом атрибут:

```
[Route("/api/[controller]")]
```

Задача 1. Создание модели данных

Для Web API необходимо определить модели. Для этого создайте папку Models (если ее нет) и разместите в ней код, описывающий товар:

```
public class Product
{
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    [Required]
    public decimal Price { get; set; }
}
```

Подсказка: атрибут `[Required]` из `System.ComponentModel.DataAnnotations` указывает на обязательность заполнения данных.

Задача 1. Список товаров в контроллере

В соответствии с моделью данных определите в классе контроллера `ProductsController` список товаров (для простоты – в виде статического поля):

```
private static List<Product> products = new List<Product>(new[] {
        new Product() { Id = 1, Name = "Notebook", Price = 100000 },
        new Product() { Id = 2, Name = "Car", Price = 2000000 },
        new Product() { Id = 3, Name = "Apple", Price = 30 },
});
```

Поскольку список товаров необходимо получать через API, добавьте метод для обработки HTTP GET-запроса к приложению:

```
[HttpGet]
public IEnumerable<Product> Get() => products;
```

Запустите приложение и убедитесь, что при обращении из браузера к относительному адресу `/api/products/`
получаете список товаров в формате JSON.

Задача 1. Получение конкретного товара

Для получения сведений по конкретному товару вам необходимо добавить в класс контролера код, отвечающий за поиск и предоставление данных по товару с выбранным идентификатором (id):

```
[HttpGet("{id}")]          // параметр для маршрутизации
public IActionResult Get(int id)
{
    var product = products.SingleOrDefault(p => p.Id == id);

    if (product == null)
    {
        return NotFound();
    }

    return Ok(product);
}
```

Проверьте, что по относительным адресам вы получаете правильный ответ:

/api/products/1/ **- есть товар**

/api/products/1234/ **- нет товара (HTTP 404)**

CRUD = Create, Read, Update, Delete

CRUD – аббревиатура, обозначающая четыре базовые функции, используемые при работе с базами данных: создание (create), чтение (read), модификация (update), удаление (delete). Термин введен James Martin в 1983 г. как стандартная классификация функций по манипуляции данными.

В системах, реализующих доступ к базе данных через API в стиле REST, эти функции реализуются зачастую (но не обязательно) через HTTP-методы POST, GET, PUT и DELETE соответственно:

Действие -> HTTP-метод

Create -> POST
Read -> GET
Update -> PUT
Delete -> DELETE

https://habr.com/ru/post/483202/
https://metanit.com/sharp/mvc/12.1.php

Задача 2. Удаление товара

Для удаления товара с использованием метода DELETE разместите в контролере следующий код:

```
[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
    products.Remove(products.SingleOrDefault(p => p.Id == id));
    return Ok();
}
```

Проверьте, что вы можете удалить товар с идентификатором 1, отправив DELETE-запрос по относительному адресу **/api/products/1/**
Получите список товаров, чтобы убедиться, что товар с идентификатором 1 был удален.

Подсказка:
Для отправки DELETE-запроса можно воспользоваться утилитой CURL (встроена в Windows):
curl -X DELETE https://localhost:44386/api/products/1

https://curl.se/

Задача 3. Свободный идентификатор

Как вы знаете, идентификаторы должны быть уникальны, поэтому при добавлении товара, необходимо использовать уникальный id. Добавьте в класс контроллера соответствующий код:

```
private int NextProductId =>
    products.Count()==0 ? 1 : products.Max(x => x.Id) + 1;


[HttpGet("GetNextProductId")] //  проверка: /api/GetNextProductId/
public int GetNextProductId() {
        return NextProductId;
}
```

Проверьте с помощью браузера правильность нахождения следующего свободного идентификатора.

Задача 3. Добавление товара

```
[HttpPost]
public IActionResult Post(Product product)
{
    if (!ModelState.IsValid) {
        return BadRequest(ModelState);
    }
    product.Id = NextProductId;
    products.Add(product);
    return CreatedAtAction(nameof(Get), new { id = product.Id },
product);
}

[HttpPost("AddProduct")]
public IActionResult PostBody([FromBody] Product product) =>
    Post(product);
```

Обратите внимание, что мы можем добавлять товар двумя способами:

- передавая характеристики товара в виде полей формы:

curl -X POST -F Name=someName -F Price=123 https://localhost:44386/api/products/

- передавая характеристики товара в виде JSON-объекта:

curl -X POST -H "Content-Type: application/json" -d "{\"name\":\"JsonName\",\"price\":1234}" https://localhost:44386/api/products/AddProduct/

Задача 3. Изменение товара

Добавьте в код контроллера поддержку обработки метода HTTP PUT для обновления данных по товару:

```
[HttpPut]
public IActionResult Put(Product product)
{
    if (!ModelState.IsValid) {
        return BadRequest(ModelState);
    }
    var storedProduct = products.SingleOrDefault(p => p.Id == product.Id);
    if (storedProduct == null) return NotFound();
    storedProduct.Name = product.Name;
    storedProduct.Price = product.Price;
    return Ok(storedProduct);
}
```

Обратите внимание, что изменение реализовано только через поля формы:
curl -X PUT -F Id=3 -F Name=UpdatedName -F Price=1234 https://localhost:44386/api/products/

TODO: самостоятельно реализуйте и протестируйте изменение товара через JSON (подсказка: параметр метода необходимо пометить атрибутом [FromBody]).

Задача 4. MVC: подготовка данных

Создайте новый проект ASP.Net Core WebApp (Model-View-Controller) под .Net 5 или выше.
В данном проекте будем работать со списком товаров, получаемом из JSON-файла **products.json**, который вы найдете в MS Teams. Скопируйте его в проект в папку **wwwroot/data**. Пример представления товара:

```
{
    "Id": "jenlooper-light",
    "Maker": "@jenlooper",
    "img": "https://........jpeg",
    "Url": "https://...",
    "Title": "A beautiful switch-on book light",
    "Description": "Use craft items you have around the house, plus two LEDs and a
LilyPad battery holder, to create a useful book light for reading in the dark.",
    "Ratings": null
}
```

Также вам понадобятся CSS-стили из файла **site.css**, который необходимо разместить в папке **wwwroot/css** (перезаписав файл).

Задача 4. MVC: создание модели товара

В папку Models добавьте класс для представления товара:

```
using System.Text.Json;
using System.Text.Json.Serialization;


public class Product
{
    public string Id { get; set; }
    public string Maker { get; set; }
    [JsonPropertyName("img")]
    public string Image { get; set; }
    public string Url { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public int[] Ratings { get; set; }

    public override string ToString() =>
JsonSerializer.Serialize<Product>(this);
}
```

Задача 4. MVC: создание сервиса для получения списка товаров

Создайте папку `Services` и добавьте класс для сервиса к которому будут обращаться все компоненты проекта для работы со списком товаров:

```
public class JsonFileProductService {
    public JsonFileProductService(IWebHostEnvironment webHostEnvironment)
    {
        WebHostEnvironment = webHostEnvironment;
    }

    public IWebHostEnvironment WebHostEnvironment { get; }

    private string JsonFileName =>
        Path.Combine(WebHostEnvironment.WebRootPath, "data", "products.json");

    public IEnumerable<Product> GetProducts()
    {
        using (var jsonFileReader = File.OpenText(JsonFileName)) {
            return JsonSerializer.Deserialize<Product[]>(jsonFileReader.ReadToEnd(),
                new JsonSerializerOptions {
                    PropertyNameCaseInsensitive = true
                });
        }
    }
}
```

В метод `ConfigureServices` (файл `Startup.cs`) добавьте строку:
`services.AddTransient<JsonFileProductService>();`

Документация по внедрению зависимостей:
https://docs.microsoft.com/ru-ru/dotnet/core/extensions/dependency-injection-usage

Задача 4. MVC: создание представления для списка товаров

Создайте папку Views/Products и добавьте представление Index.cshtml со следующим кодом:

```
@model IEnumerable<Product>
@{
    ViewData["Title"] = "Список товаров";
}

<div class="text-center">
    <h1 class="display-4">Список товаров</h1>
<div class="card-columns">
@foreach (var product in Model)
{
    <div class="card">
        <div class="card-img" style="background-image: url('@product.Image');"></div>
        <div class="card-body">
            <h5 class="card-title">@product.Title</h5>
        </div>
        <div class="card-footer">
            <small class="text-muted"><a href="/product/@product.Id/">Перейти в карточку товара</a>
            </small>
        </div>
    </div>
}
</div>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
</div>
```

Обратите внимание на тип модели для представления - IEnumerable<Product> - он совпадает с типом возвращаемого значения из метода `GetProducts()`.

Задача 4. MVC: контроллер

В папку Controllers добавьте новый класс-контроллер со следующим содержимым (обратите внимание на внедрение зависимостей: через конструктор передается сервис для доступа к списку товаров):

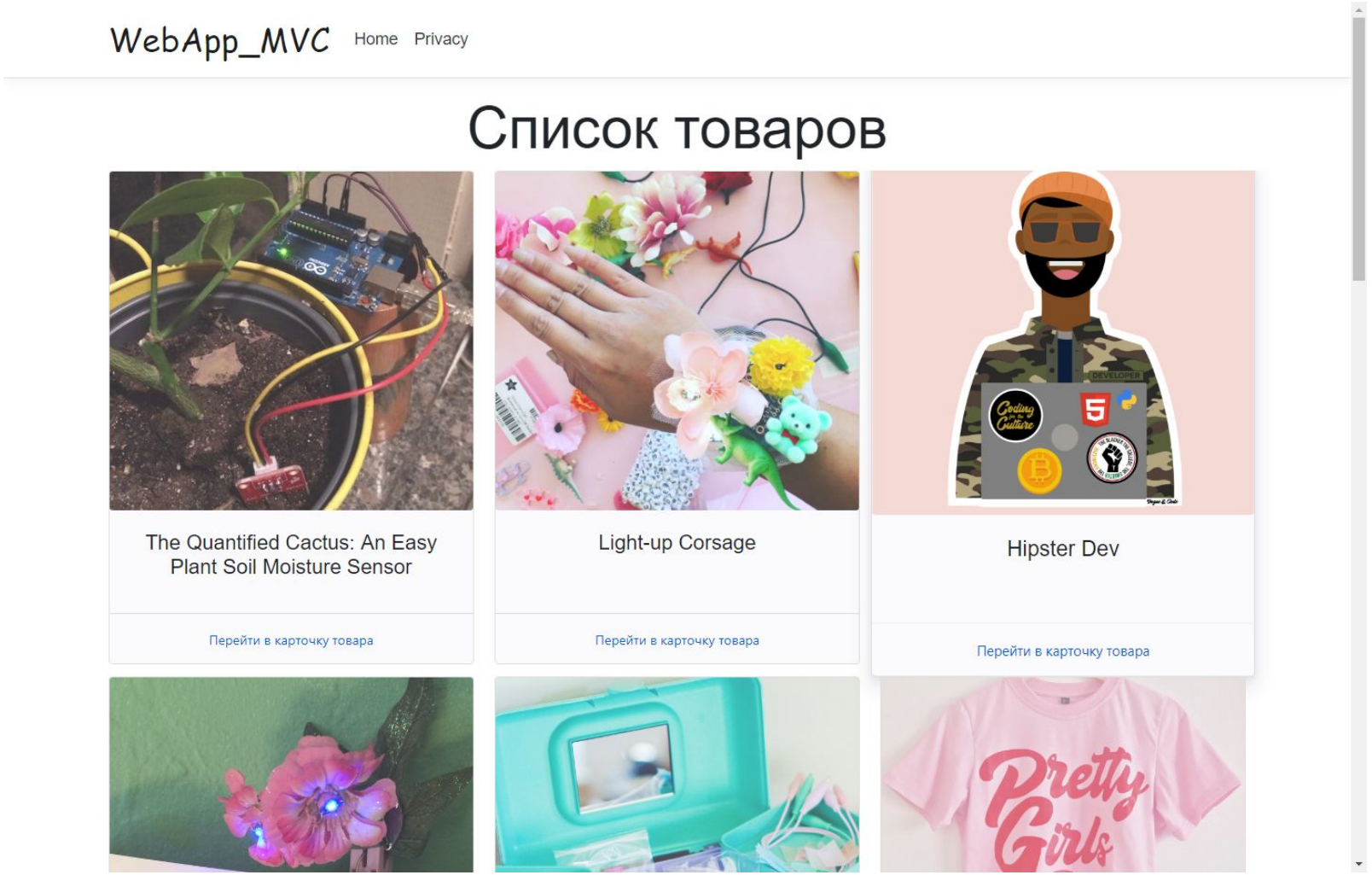
```
public class ProductsController : Controller
{
    public ProductsController(JsonFileProductService productService) {
        ProductService = productService;
    }

    public JsonFileProductService ProductService { get; }

    [HttpGet]
    public IActionResult Index() {
        return View(ProductService.GetProducts());
    }
}
```

В метод Configure файла Startup.cs внесите изменения, касательно умалчиваемого контролера:

```
endpoints.MapControllerRoute(
                  name: "default",
                  pattern: "{controller=Products}/{action=Index}/{id?}");
```

Задача 4. MVC: итог

Проверьте работоспособность приложения в браузере:



Обратите внимание, что переход на карточку товара пока не работает..
Самое время исправить это в следующем задании...

Задача 5. MVC: создание представления для карточки товара

В папку Views/Products добавьте представление ViewItem.cshtml с содержимым:

```
@model Product
@{
    ViewData["Title"] = Model.Title;
}
<div class="text-center">
    <h1 class="display-4">@Model.Title</h1>
    <div class="card">
        <img class="card-img" src="@Model.Image" />
        <div class="card-body">
            <p class="card-text">@Model.Description</p>
        </div>
    </div>
</div>
```

Обратите внимание на тип модели для представления – Product – объект этого типа должен передаваться в представление для отображения.

Задача 6. MVC: модификация контроллера

В класс-контроллер `ProductsController` добавьте метод `Product` для обработки адресов вида `/product/<id_товара>`:

```
[Route("product/{id}")]
public IActionResult Product(string id) =>
 View("ViewItem", ProductService.GetProducts().First(x => x.Id == id));
```

Обратите внимание, что метод использует сервис для получения списка товаров, разработанный ранее и LINQ для нахождения запрашиваемого товара по идентификатору. После этого найденный товар в качестве модели передается в представление ViewItem.cshtml, созданное ранее.

Если вы все сделали верно, то при переходе в карточку товара вы увидите единственный товар. Обратите внимание на адаптивность полученной страницы (см. на скриншоте меню справа вверху).

Задача 7. MVC: каталог товаров

По аналогии со списком товаров добавьте в MVC-приложение вывод информации по списку разделов (рубрик).
Источник для хранения разделов определите на свое усмотрение.
Для решения задачи вам потребуется создать новый контроллер (нагромождать текущий неразумно) и представления для вывода списка разделов и конкретного раздела пользователю.

Дополнительное задание:
Добавьте в JSON с товарами ссылку на принадлежность товара конкретному разделу и реализуйте вывод списка товаров в каждом разделе.