ASP.NET Core MVC

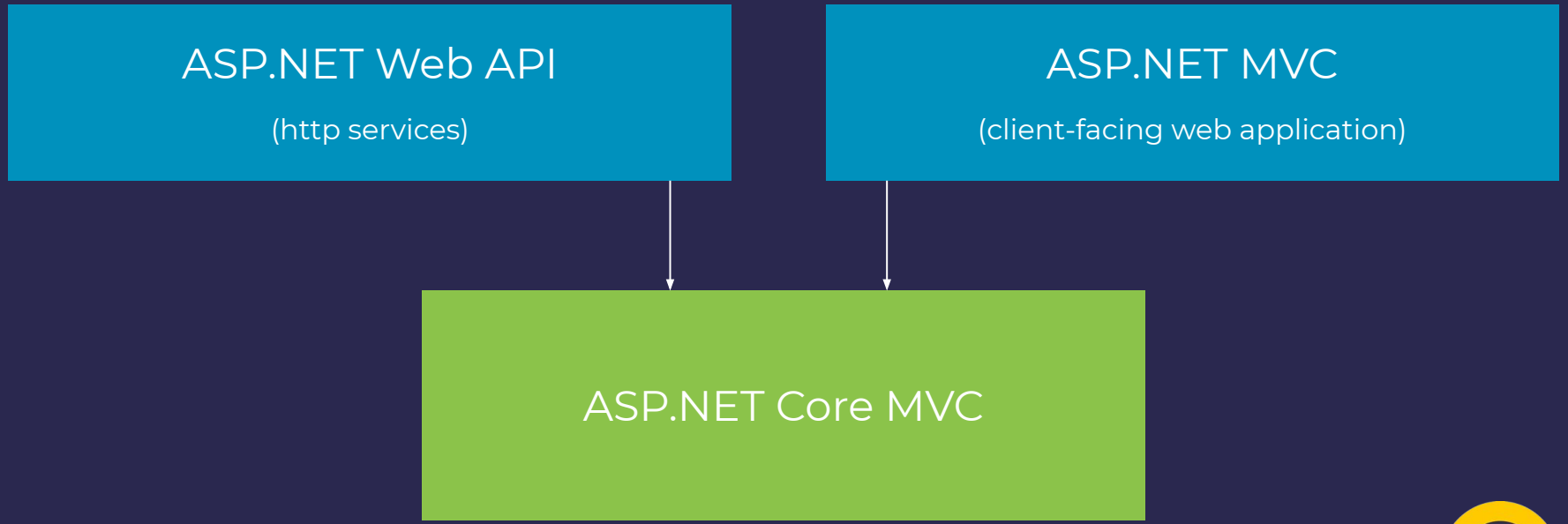
(Сериализация, форматы JSON/XML, Контроллеры, Экшены, Маршрутизация, Коды Статусов, Swagger)

Андрей Голяков

Middleware для API

С прошлого занятия мы помним, что нам нужно добавить `Middleware` в `HTTP Request Pipeline` в метод `Configure` класса `Startup`, а также добавить необходимые framework-сервисы в контейнер внутри метода `ConfigureServices` (также класса `Startup`).

ASP.NET Core MVC

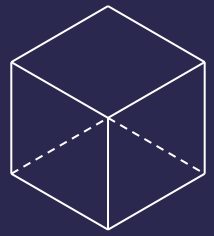


MVC как шаблон проектирования

MVC — Model-View-Controller (Модель-Представление-Контроллер) — это архитектурный шаблон, используемый для проектирования пользовательских интерфейсов.

Он характеризуется низкой связностью и разделением ответственности. Благодаря этому, так построенные интерфейсы проще тестировать и переиспользовать.

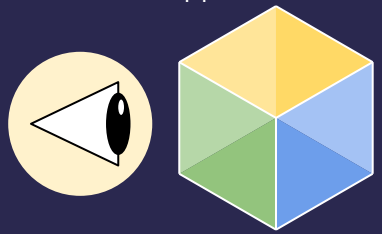
Модель - Представление - Контроллер



Служит для хранения и передачи данных, которые требуется отобразить (в идеале не содержит логики)

Model
(модель)

Отвечает за способ визуализации данных модели



View
(представление)

Отвечает за формирование модели (логика) и её передачу представлению для отображения

Controller
(контроллер)

Совместная работа

Добавление сервиса MVC и middleware MVC в HTTP Request Pipeline.

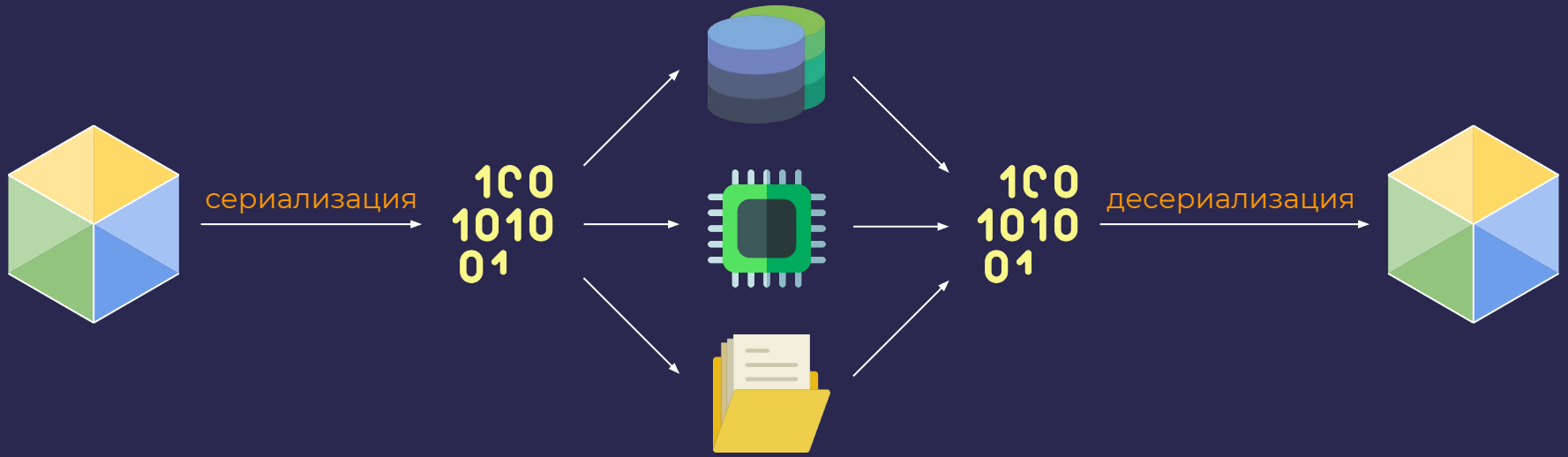
Добавление контроллера (controller) CitiesController

Добавление экшена (action method) GetCities

Сериализация

Сериализация — это процесс преобразования объекта в поток байтов для сохранения или передачи в базу данных, память или файл. Эта операция предназначена для того, чтобы сохранить состояния объекта для последующего воссоздания при необходимости.

Обратный процесс называется десериализацией.



Формат JSON

JSON: JavaScript Object Notation
Content-Type: application/json

Согласно RFC4627 JSON-текст служит для кодирования двух структур: объектов и массивов:

```
{
  "name": "Alena",
  "age": 29,
  "sex": "F",
  "isMarried": true,
  "address": {
    "country": "Russia",
    "city": "Moscow"
  },
  "childrenAges": [4.5, 7, 12]
}
```

- { начало объекта
- } конец объекта
- [начало массива
-] конец массива
- : разделитель поля и значения в объекте
- , разделитель значений
  (пар поле-значение в объекте или значений в массиве)
- числа: 29, 4.5 и т.д.
- строки: "в кавычках"
- литералы: true, false, null
- имена полей: "в кавычках"

Полезные online-инструменты:

- https://beautifier.io - форматирование JavaScript-кода
- https://csharp2json.io и http://json2csharp.com - преобразование структур C# в JSON и обратно
- https://app.quicktype.io/#l=cs&r=json2csharp - генерация классов C# на базе заданного JSON

Совместная работа

Добавление экшена (action method) GetCities, возвращающего ответ в формате JSON

Маршрутизация (Routing)

Маршрутизация MVC разбирает (анализирует) URI HTTP-запроса и пытается сопоставить пришедший URI с определённым контроллером и определённым экшеном (методом) этого контроллера.

Существует два способа, которыми можно задать необходимые маршруты:

- Метод, основанный на соглашении (convention-based)
- Метод, основанный на атрибутах (attribute-based)

Convention-based routing

Маршрутизация, основанная на соглашении, выглядит следующим образом:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    // ...
    app.UseMvc(config =>
        config.MapRoute(
            name: "Default",
            template: "{controller}/{action}/{id?}",
            defaults: new { controller = "Home", action = "Index" })
    );
    // ...
}
```

Например, такой маршрут будет сопоставлять URI /cities/index (согласно наименованиям) экшену Index контроллера CitiesController.

Convention-based vs Attribute-based routing

Маршрутизация, основанная на соглашении, обычно используется в тех случаях, когда фреймворк MVC используется для веб-приложений, возвращающих HTML-представления.

Команда разработчиков .NET Core не рекомендует использование этого метода определения маршрутов для написания API. Вместо этого для API рекомендуется использовать маршрутизацию, основанную на атрибутах.

Attribute-based routing

Маршрутизация, основанная на атрибутах, позволяет задавать маршруты через атрибуты, устанавливаемые как на уровне контроллера, так и на уровне экшена:

```
public class CitiesController : Controller
{
    [HttpGet("/api/cities")]
    public JsonResult GetCities() { // return cities... }
}
```

В примере выше маршрут прописан на уровне экшена, однако если у нас будет много экшенов внутри данного контроллера, мы захотим, чтобы они выглядели консистентно не нарушая DRY

```
[Route("/api/cities")]
public class CitiesController : Controller
{
    [HttpGet()]
    public JsonResult GetCities() { // return cities... }
}
```

Совместная работа: Создаём Модель

Создадим класс модели City и будем использовать этот класс модели вместо анонимного объекта в контроллере.

Используем сниппеты (snippets) Visual Studio 2017, например:

- `prop` + `[Tab]` + `[Tab]` — для создания свойства `{ get; set; }`
- `propg` + `[Tab]` + `[Tab]` — для создания свойства `{ get; private set; }`
- `ctor` + `[Tab]` + `[Tab]` — для создания конструктора объекта

Все возможные сниппеты можно найти здесь:

- https://docs.microsoft.com/en-us/visualstudio/ide/visual-csharp-code-snippets?view=vs-2017

Совместная работа: Хранилище

Создадим папку DataStore, а в ней класс хранилища CitiesDataStore который будет использоваться вместо базы данных нашим API.

Методы контроллера будут работать с этим классом как с полноценным хранилищем.

Само хранилище будет работать на базе предварительно заполненного списка, однако мы реализуем паттерн синглтон для этого класса.

Вносим соответствующие изменения в контроллер, чтобы он использовал наше новое хранилище, которое возвращает типизированные, а не анонимные объекты.

Совместная работа: Добавляем Экшен

Добавляем экшен получения города по идентификатору.

Совместная работа: Добавляем Экшен

```
[HttpGet("{id}")]
public JsonResult GetCity(int id)
{
    var citiesDataStore = CitiesDataStore.GetInstance();
    var city = citiesDataStore
        .Citites
        .Where(x => x.Id == id)
        .FirstOrDefault();

    return new JsonResult(city);
}
```

Совместная работа: Добавляем Экшен

```
[HttpGet("{id}")]
public JsonResult GetCity(int id)
{
    var citiesDataStore = CitiesDataStore.GetInstance();
    var city = citiesDataStore
        .Citites
        .Where(x => x.Id == id)
        .FirstOrDefault();

    return new JsonResult(city);
}
```

Совпадение имени параметра метода и имени шаблона в атрибуте не случайно. Именно благодаря этому значение из строки запроса попадает в параметр нашего экшена.

Совместная работа: Добавляем Экшен

```
[HttpGet("{id}")]
public JsonResult GetCity(int id)
{
    var citiesDataStore = CitiesDataStore.GetInstance();
    var city = citiesDataStore
        .Citites
        .Where(x => x.Id == id)
        .FirstOrDefault();

    return new JsonResult(city);
}
```

Совпадение имени параметра метода и имени шаблона в атрибуте не случайно. Именно благодаря этому значение из строки запроса попадает в параметр нашего экшена.

Response Status Codes

* больше статусов в Википедии

Level 200
Success

200 - OK

201 - Created

204 - No Content

Level 400
Client Error

400 - Bad Request

401 - Unauthorized

403 - Forbidden

404 - Not Found

409 - Conflict

Level 500
Server Error

500 - Internal Server Error

Совместная работа

- Возвращение правильных кодов статуса
  Например, 404 - Not Found для случаев, когда город с запрошенным ID не найден.
- Добавление Status Code Pages middleware
- Добавление возможности возвращать данные не только в формате JSON, а учитывать заголовок Accept пользователя API

  В более общем виде это называется Content Negotiation - возможность предоставлять по одному и тому же URI разные ответы в зависимости от нюансов запроса.

Совместная работа

- Создание метода для добавления нового города
- Чтение параметра экшена из тела запроса
- Возвращение статуса 204 Created

Swagger

- Автодокументация API: https://swagger.io
  - NuGet-пакет Swashbuckle.AspNetCore (Pre-Release version)
- [Habr] Создание справочных страниц веб-API ASP.NET с помощью Swagger https://habr.com/ru/company/microsoft/blog/325872/

Полезные ссылки

- [ms] Основная страница документации Microsoft по ASP.NET Core
  https://docs.microsoft.com/en-us/aspnet
- [ms] Учебник. Начало работы с ASP.NET Core
  https://docs.microsoft.com/ru-ru/aspnet/core/getting-started/?view=aspnetcore-2.2
- [ms] Общие сведения ASP.NET Core MVC
  https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.2
- [habr] Корректный ASP.NET Core
  https://habr.com/ru/post/437002
- [habr] JSON API – работаем по спецификации ★★★★★
  https://habr.com/ru/company/oleg-bunin/blog/433322

Домашняя работа

Дописать методы по замене (PUT) и удалению (DELETE) конкретного города.

Спасибо за внимание.