Веб-API

ASP.NET и Clean Architecture

Неделя 10

ASP.NET Core

ASP.NET Core представляет собой современный, высокопроизводительный фреймворк для создания веб-приложений, который работает на всех основных платформах (Windows, Linux, macOS).

В отличие от своего предшественника, ASP.NET Framework, новая версия полностью переработана для обеспечения модульности и производительности.

Преимущества ASP.NET Core:

- **Кросс-платформенность:** Приложения могут разрабатываться и запускаться на Windows, Linux и macOS
- **Модульная архитектура:** Приложения включают только необходимые зависимости
- **Встроенный контейнер DI (Dependency Injection):** Упрощает тестирование и поддерживает слабосвязанную архитектуру
- **Высокая производительность:** Один из самых быстрых веб-фреймворков согласно TechEmpower benchmarks
- **Единый стек для веб-UI и веб-API:** Разработка различных типов веб-приложений с использованием общих концепций

Контроллеры vs Minimal API

Минимальный API (Minimal API):

- Введён в ASP.NET Core 6.0 как новый, упрощённый способ создания API
- Сокращает шаблонный код (boilerplate) и уменьшает количество файлов
- Всё взаимодействие настраивается непосредственно в Program.cs
- Идеально подходит для микросервисов и небольших проектов
- Оптимизирован для производительности с меньшим использованием рефлексии

```
app.MapGet("/weatherforecast", () =>
{
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)]
        ))
        .ToArray();
    return forecast;
})
.WithName("GetWeatherForecast")
.WithOpenApi();
```

Контроллеры vs Minimal API

Controller-based API:

- Хорошо структурированная модель с разделением ответственности
- Основан на контроллерах, которые группируют связанные конечные точки (endpoints)
- Поддерживает полный набор функций MVC (фильтры, связывание модели, валидация)
- Следует устоявшимся практикам и шаблонам REST
- Существует с начала ASP.NET Core 1.0

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    private readonly ILogger<WeatherForecastController> _logger;

    public WeatherForecastController(ILogger<WeatherForecastController> logger)
    {
        _logger = logger;
    }

    [HttpGet(Name = "GetWeatherForecast")]
    public IEnumerable<WeatherForecast> Get()
    {
        // метод контроллера
    }
}
```

Когда использовать Minimal API

Minimal API лучше подходит для определенных сценариев:

Микросервисы и небольшие проекты

- Упрощенная структура проекта с меньшим количеством файлов
- Более быстрая разработка благодаря меньшему количеству шаблонного кода
- Легче понять весь сервис, так как весь код находится в одном месте

Производительно-критичные приложения

- Меньше накладных расходов на обработку запросов
- Меньше рефлексии во время выполнения

Пример сценария: Микросервис для обработки платежей с несколькими четко определенными эндпоинтами будет более эффективен и прост в реализации с помощью Minimal API.

Когда использовать традиционный API

Традиционный подход с контроллерами является более зрелым и предоставляет ряд преимуществ в определенных сценариях:

Сложные бизнес-логики и большие проекты

- Лучшая организация кода в больших проектах с множеством эндпоинтов
- Естественное группирование связанных эндпоинтов в контроллеры
- Более чёткое разделение ответственности

Расширенные возможности MVC

- Полная поддержка фильтров (авторизация, кэширование, валидация)
- Более мощные механизмы привязки моделей
- Встроенная поддержка валидации с помощью атрибутов

Пример сценария: Корпоративное приложение с множеством взаимосвязанных бизнес-сущностей и сложной логикой авторизации будет лучше структурировано с использованием контроллеров.

REST

REST — это архитектурный стиль для проектирования распределенных систем, особенно веб-сервисов. REST не связан с конкретной технологией и может быть реализован на любой платформе.

REST

Основные принципы REST:

- **Ресурсы:** Данные и функциональность представлены как ресурсы, которые идентифицируются через URI (Uniform Resource Identifier).
- **HTTP методы:** Для операций с ресурсами используются стандартные HTTP методы (GET, POST, PUT, DELETE).
- **Представления:** Ресурсы могут иметь различные представления (JSON, XML, HTML).
- **Отсутствие состояния:** Каждый запрос содержит всю необходимую информацию для его выполнения.
- **HATEOAS** (Hypermedia as the Engine of Application State): API предоставляет гиперссылки, которые клиент может использовать для перехода между ресурсами.

Примеры RESTful API:

- `GET /tasks` — получить список задач
- `GET /tasks/123` — получить информацию о конкретной задаче
- `POST /tasks` — создать новую задачу
- `PUT /tasks/123` — обновить существующую задачу
- `DELETE /tasks/123` — удалить задачу

MVC

MVC — это шаблон проектирования, который разделяет приложение на три основных компонента, обеспечивая разделение ответственности.

Компоненты MVC:

- **Модель (Model):** Представляет данные и бизнес-логику приложения. Модель уведомляет представления об изменениях данных.
- **Представление (View):** Отвечает за визуализацию данных для пользователя. Получает данные от модели и отображает их.
- **Контроллер (Controller):** Обрабатывает входящие запросы, взаимодействует с моделью и выбирает представление для ответа.

В контексте ASP.NET Core API, мы часто используем только компоненты Model и Controller (без View), поскольку API обычно возвращает данные в форматах JSON или XML, а не HTML-страницы.

Clean Architecture