



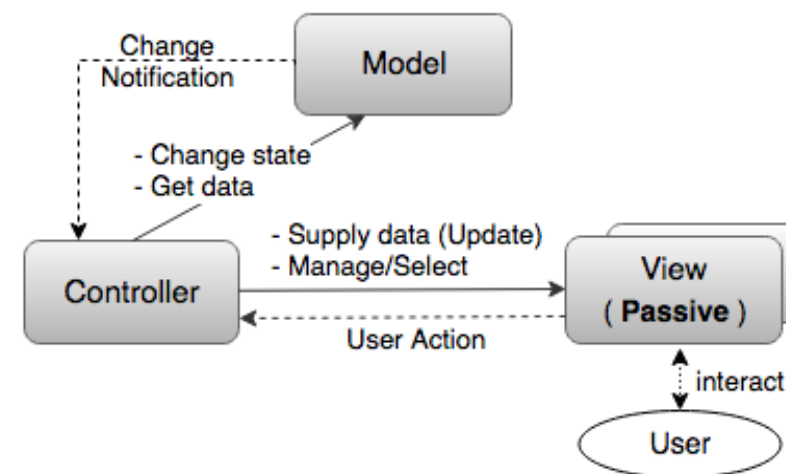
Казанский федеральный  
УНИВЕРСИТЕТ

ВЫСШАЯ ШКОЛА  
информационных технологий  
и информационных систем

# ASP.NET Core MVC

# Вспомним идеи MVC

- Отделение модели от представления
- Независимость модели
- Представление отображает данные модели



# Слои приложения

1. Модель предметной области
2. Доступ к данным
3. Web-приложение

Для уменьшения связности слои  
выделены в отдельные проекты

[https://en.wikipedia.org/wiki/Multitier\\_architecture](https://en.wikipedia.org/wiki/Multitier_architecture)

<https://blogs.msdn.microsoft.com/malaysia/2013/08/02/layered-architecture-for-net/>

# 1. Модель предметной области

- Классы сущностей + механизм работы с ними
  - В случае Entity Framework –  
POCO сущности + DbContext
- В отдельном проекте  
(сборке – .NET Standard class library)

## 2. Доступ к данным

- Отдельный проект
- Классы, предоставляющие доступ к сущностям предметной области
- Предоставляют интерфейс для CRUD операций
- Соккрытие специфики построения запросов с помощью ORM, инициализации объектов

# Repository pattern

- Фасад для доступа к данным
- Слой абстракции между моделью предметной области и бизнес логикой с представлением
- Предоставляет доступ к сущностям с CRUD интерфейсом

# Generic Repository

- Тот же репозиторий, только абстрагированный от конкретных типов сущностей
  - + Можно абстрагироваться от конкретного DbContext
- DRY принцип в действии
- У всех репозиториев один интерфейс
- Легко тестировать (мокать)

<http://blog.byndyu.ru/2011/01/domain-driven-design-repository.html>

# Инъекция Generic Repository

- Можно в качестве сервисов приложения использовать репозитории
- Это неудобно
- Удобнее будет собрать репозитории воедино в Unit of Work
  - Как это происходит с DbSet'ами в DbContext у Entity Framework



# Unit of Work

- Объединяем несколько репозиториев в общую оболочку
- Используем UOW для удобного внедрения зависимостей

# 3. Слой web-приложения

- Проект ASP.NET Core MVC
- Отвечает за обработку запросов и отправку ответов
- Формирует веб-страницы для отображения
- Взаимодействует со слоем доступа к данным

<https://docs.microsoft.com/en-us/aspnet/core/mvc>

# ASP.NET Core MVC

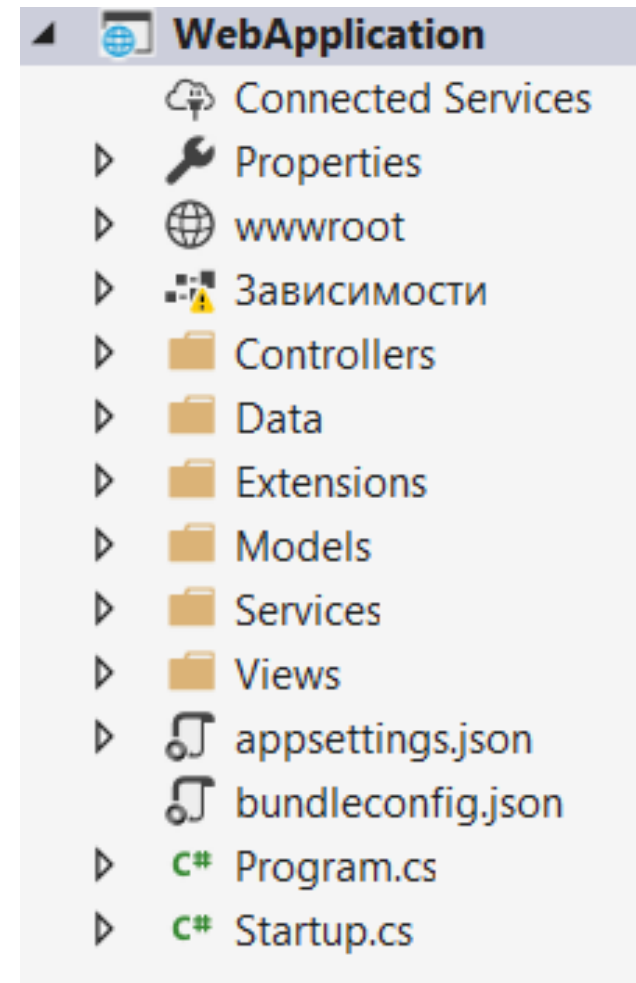
- Сервис и middleware для реализации архитектурного паттерна MVC в ASP.NET Core

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app)
    {
        app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
        });
    }
}
```

# Проект ASP.NET Core MVC

- Контроллеры
- Модели
- Представления
- Конфигурации
- Задачи по минификации
- Program и Startup



# М: Модель

- Описывает данные, используемые в приложении
- Модель MVC не является моделью предметной области
- Описывает классы для отображения и передачи данных

# ViewModel

- Модель представления
- Позволяют передавать в представление часть данных одной модели или объединить информацию нескольких
- Формированием данных занимается контроллер

<http://blog.byndyu.ru/2012/05/viewmodel-domain-model.html>

# C: Контроллеры

- Центральное звено в MVC
- Классы, наследники  
`Microsoft.AspNetCore.Mvc.Controller`
- Имена с суффиксом Controller
- Методы – действия, сопоставляемые с запросами

# V: Представления

- Формируют внешний вид приложения
- Файлы для генерации html страниц
  - Шаблон html страницы
  - Конструкции для связи с кодом на языке C# (Razor)
- Расширение cshtml (cs + html)



# Razor

- Код C# имеет префикс @
- Допускается однострочный и блочный код {...}
- Допускается использование локальных переменных и блоков управления (условий и циклов)
- Допускается определение методов представления в блоке functions

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor>

# Данные представления

- **ViewData, ViewBag, TempData** – хранилища данных
- **Model** – предпочтительный способ передачи данных в представление
  - Указываем тип, с которым работает представление
  - Принимаем модель в качестве аргумента метода `View`

# Скаффолдинг

- Возможность генерации типового контроллера и, при необходимости, представления по классу модели (модели представления)
- Генерирует CRUD методы и соответствующие представления

# Маршрутизация

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app)
    {
        app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
        });
    }
}
```

<https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing>

# Атрибуты маршрутизации

- Атрибуты Route определяют маршрут
- Позволяют комбинировать параметры, задавать префиксы, объединять обработку нескольких маршрутов

```
[Route("{id:int}/{name:maxlength(10)}")]  
[Route("main/{id:int}/{name:maxlength(10)}")]  
[Route("[controller]/[action]/{id?}")]
```

# Действия контроллеров

- Атрибут `[ActionName]` позволяет связывать метод контроллера с действием, имеющим другое имя
  - `[NonAction]` – не рассматривать метод как действие
  - `[NonController]` – не рассматривать класс как контроллер
- `[HttpGet]`, `[HttpPost]`, `[HttpPut]`, `[HttpDelete]` и `[HttpHead]` указывают HTTP метод запроса

# ControllerContext

- ModelState – словарь для валидации
- HttpContext контекст запроса
  - Request
    - Body, Cookies, Form, Headers, Path, Query, QueryString
  - Response
    - Body, Cookies, ContentType, Headers, StatusCode
  - User
  - Session

# Передача параметров в запросе

```
public class ProductsApiController : Controller
{
    [HttpGet("/products/{id}", Name = "Products_List")]
    public IActionResult GetProduct(int id?) { ... }
}
```

Обработка запросов вида /products/5 и /products/

Возможность опускать параметр запроса обеспечивается nullable (int?)



# Загрузка файлов

```
<form method="post" enctype="multipart/form-data"  
asp-controller="UploadFiles" asp-action="Index">
```

```
<input type="file" name="files" multiple />
```

```
[HttpPost("UploadFiles")]  
public async Task<IActionResult> Post(List<IFormFile> files)
```

<https://docs.microsoft.com/en-us/aspnet/core/mvc/models/file-uploads>

# Привязка модели

- Связывание данных HTTP запроса с параметрами методов контроллеров
- Зависит от маршрута  
`{controller=Home}/{action=Index}/{id?}`
- MVC будет привязывать данные по имени
- Для привязки пользовательских типов нужен public конструктор без параметров и доступ к членам класса

<https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding>

# Представления

- Мы можем внедрять сервисы в представления командой `@inject`
- Как добавление свойства к представлению и получение значения с помощью DI
- Синтаксис `@inject <type> <name>`

# Ссылки

1. <https://github.com/aspnet/>
2. <https://metanit.com/sharp/aspnet5/>
3. <https://www.asp.net/mvc/books/aspnet-mvc-2-books>
4. <http://blog.byndyu.ru/2011/01/domain-driven-design-repository.html>
5. <https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design>
6. <https://docs.microsoft.com/en-us/aspnet/core/mvc>
7. <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/index?tabs=aspnetcore2x>

# Темы и предложения для разбора

- MVC, Razor и другие полезные инструменты
- Аутентификация и авторизация
- WebAPI
- Тестирование
- Загрузка файлов. Облачное хранение данных
- Функциональность реального времени
- Развёртывание на Linux с Nginx
- Микросервисы на ASP.NET Core
- Контейнерная виртуализация
- Одностраничные приложения (SPA)
- Web-приложения ASP.NET Core на F#



Казанский федеральный  
УНИВЕРСИТЕТ

ВЫСШАЯ ШКОЛА  
информационных технологий  
и информационных систем

# ASP.NET Core MVC