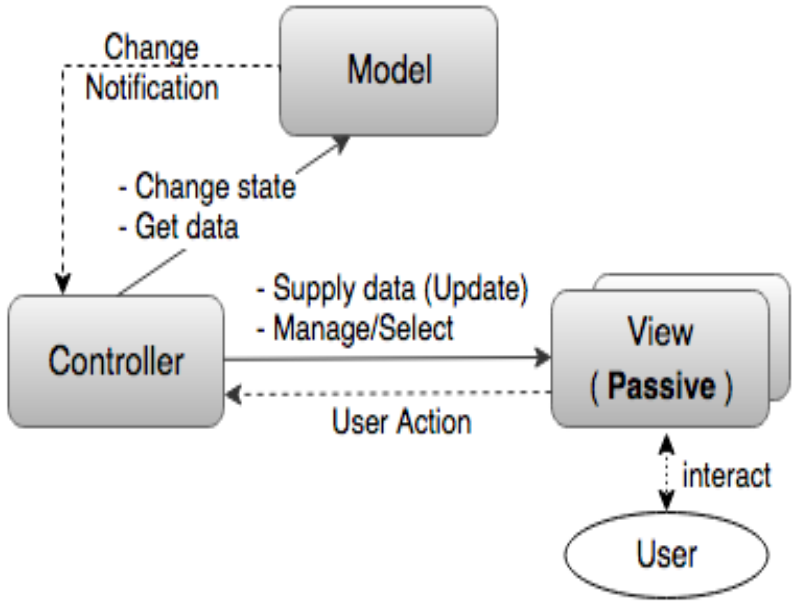
Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

ASP.NET Core MVC

© Марченко Антон Александрович marchenko@it.kfu.ru Казань, 2020 г.

Вспомним идеи MVC

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



https://habrahabr.ru/post/321050/

Слои приложения

1. Модель предметной области
2. Доступ к данным
3. Web-приложение (пользовательский интерфейс)

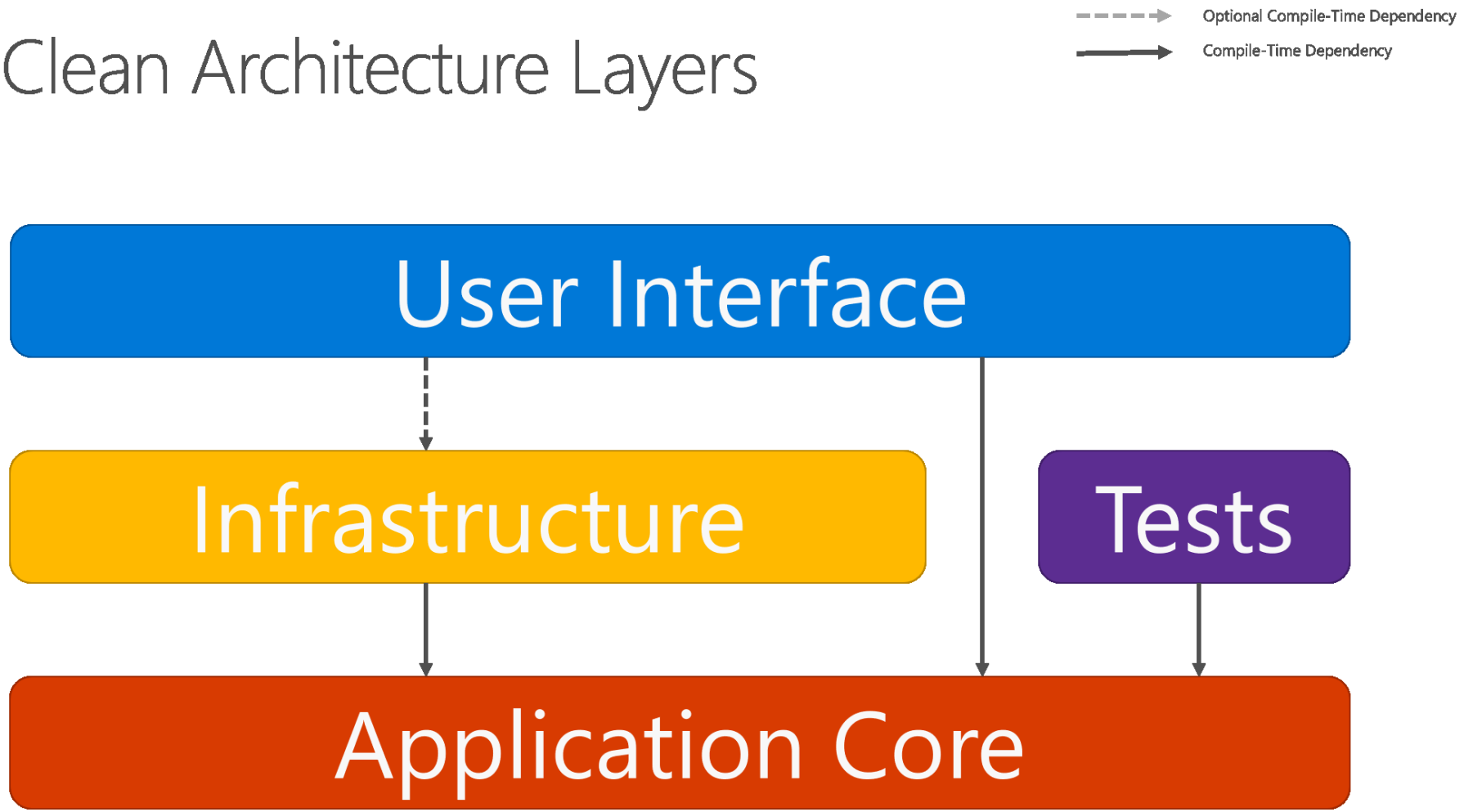
Для уменьшения связности слои выделены в отдельные проекты

https://en.wikipedia.org/wiki/Multitier_architecture

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

Слои приложения

Clean Architecture Layers



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
- Сокрытие 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

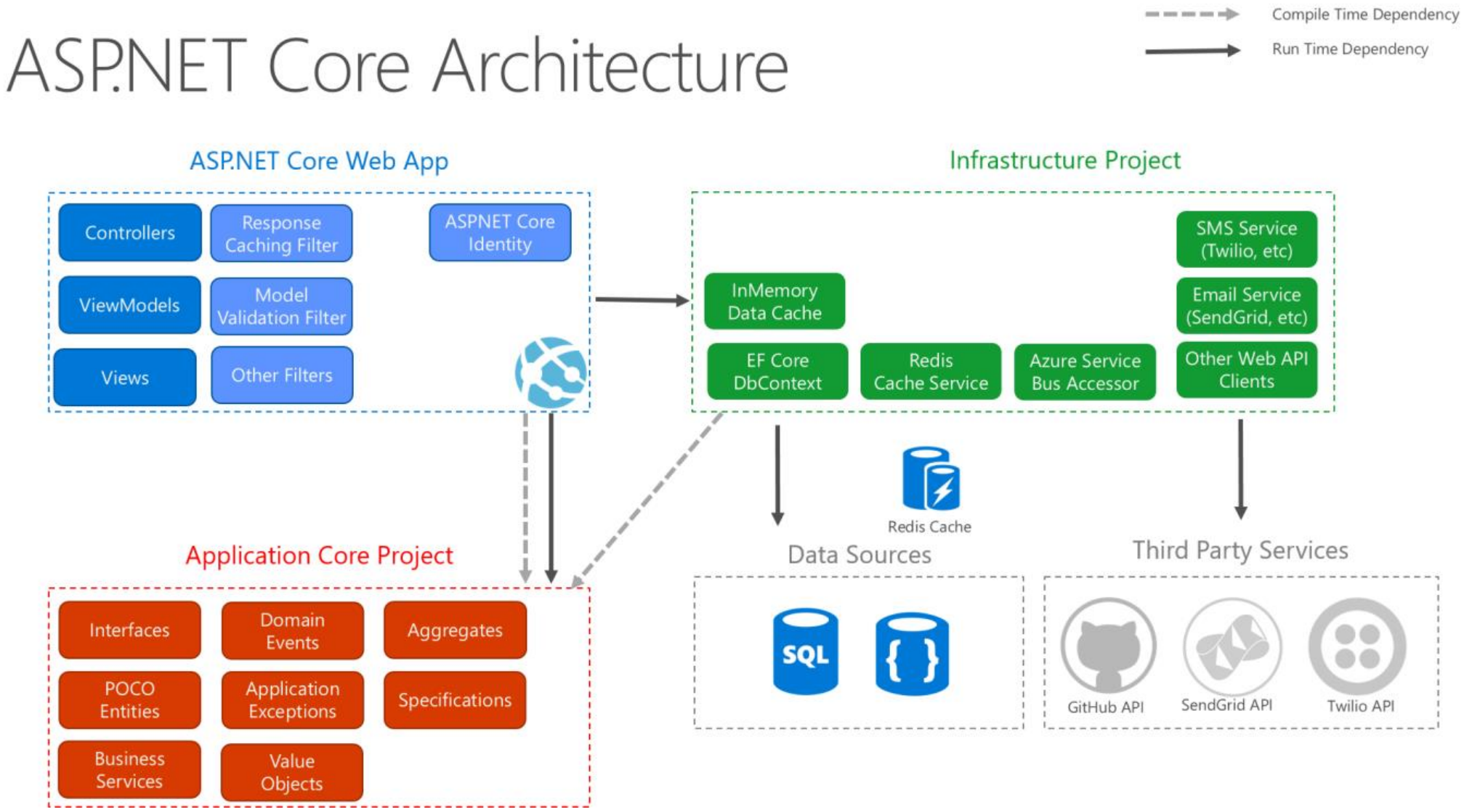
Инъекция Generic Repository

- 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
- Работа с ними осуществляется транзакционно
- Используем UoW для удобного внедрения зависимостей

Чистая архитектура ASP.NET Core



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

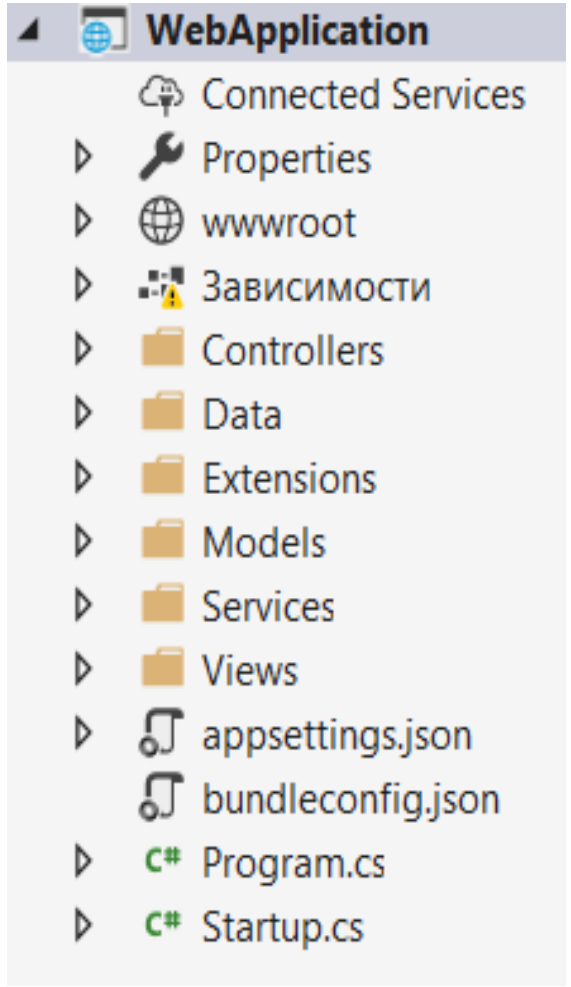
- Сервис и middleware для реализации архитектурного паттерна MVC в ASP.NET Core

```
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



M: Модель

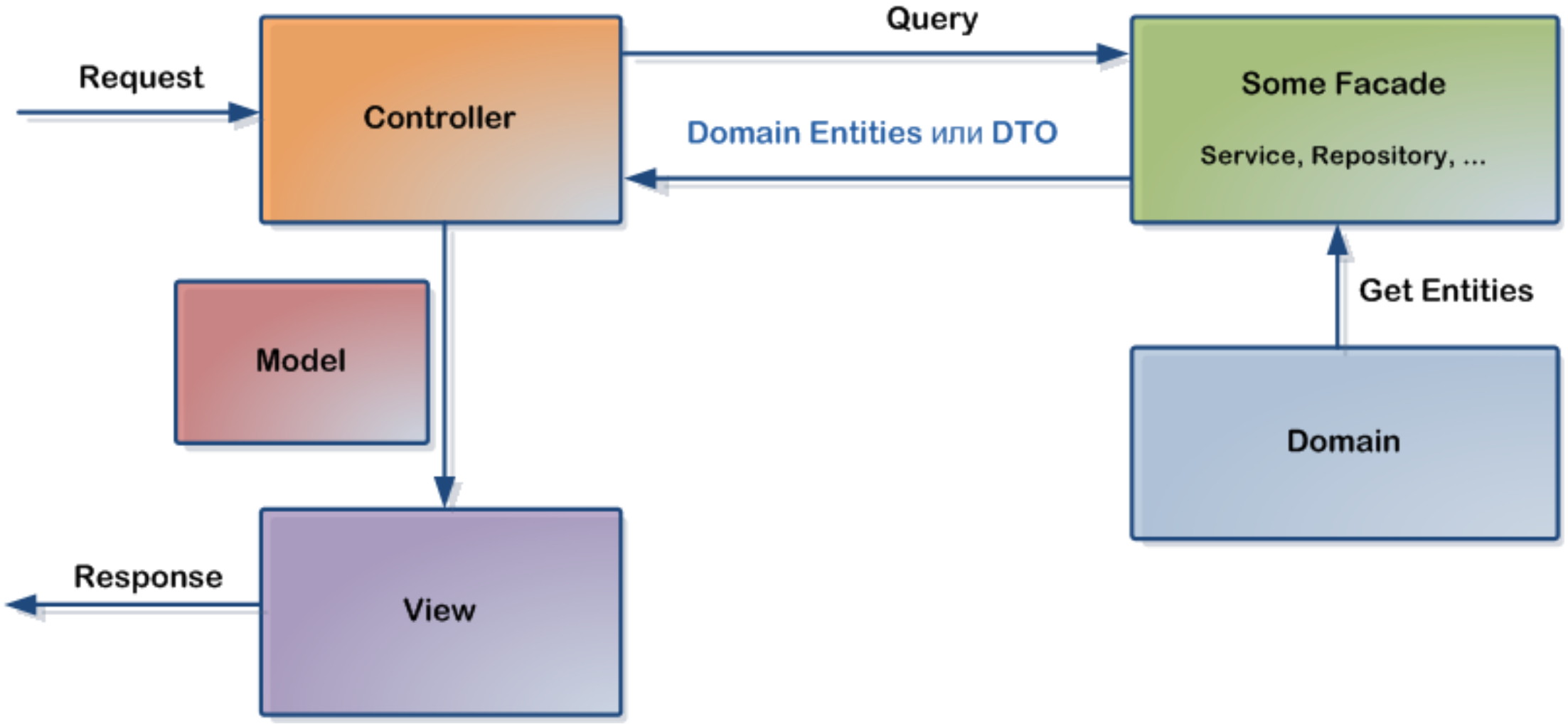
- Описывает данные, используемые в приложении
- Модель MVC не является моделью предметной области
- Описывает классы для отображения и передачи данных

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

ViewModel = Model



Валидация модели

```
using System.ComponentModel.DataAnnotations;
public class LoginViewModel
{
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        [Display(Name = "Remember me?")]
        public bool RememberMe { get; set; }
}
```

C: Контроллеры

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

Валидация модели

```
public async Task<IActionResult> Login(LoginViewModel model, string
returnUrl = null)
{
        if (ModelState.IsValid)
        {
                // работаем с моделью
        }
        // Здесь, что-то пошло не так, показываем форму ещё раз
        return View(model);
}
```

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

Вспомогательные функции тегов

- Позволяют серверному коду участвовать в создании и отрисовке HTML-элементов в файлах Razor
- Большая схожесть с HTML
- Полнофункциональная среда IntelliSense для создания разметки HTML и Razor

```
<label asp-for="Movie.Title"></label>
<label for="Movie_Title">Title</label>
```

Скаффолдинг

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

https://metanit.com/sharp/aspnet5/21.2.php

Маршрутизация

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

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

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

Несколько путей

```
app.UseMvc(routes =>
{
    routes.MapRoute("blog", "blog/{*article}",
        defaults: new { controller = "Blog", action = "Article" });
    routes.MapRoute("default",
                    "{controller=Home}/{action=Index}/{id?}");
});
```

Первый параметр служит для указания имени пути (в дальнейшем можно использовать для создания ссылок с помощью метода Url.RouteUrl).

Маршрутизация

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
        [HttpGet("{id}")]
        public IActionResult GetProduct(int id)
        { ... }
}
```

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

- Все открытые методы по умолчанию являются действиями
- 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
- ActionDescriptor – дескриптор действия - объект, который описывает вызываемое действие контроллера
- RouteData – данные маршрута

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

Загрузка файлов. Безопасность

- Передавать файлы в выделенную область для отправки файлов, желательно не на системный диск.
- Не сохраняйте переданные файлы в дереве каталогов, где находится приложение.
- Использовать безопасное имя файла, определяемое приложением. Не используйте имя файла, предоставленное пользователем, или ненадежное имя переданного файла
- Разрешать только утвержденные расширения файлов для спецификации на проектирование приложения.
- Убедиться, что проверки на стороне клиента выполняются и на сервере
- Проверять размер отправленного файла. Установить максимальный предельный размер
- Если файлы не должны перезаписываться переданным файлом с тем же именем, перед отправкой файла проверять его имя в базе данных или физическом хранилище.
- В идеале сканер для проверки отправляемого содержимого на наличие вирусов и вредоносных программ до сохранения файла

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

Привязка модели

Привязка модели попытается найти значения для следующих типов целевых объектов:

- Параметры метода действия контроллера, к которому направлен запрос.
- Параметры метода обработчика Razor Pages, к которому направлен запрос.
- Открытые свойства контроллера или класса PageModel, если задано атрибутами.

Представления

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

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/dependency-injection

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