**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное автономное образовательное учреждение высшего образования

**«Национальный исследовательский Томский политехнический университет»**

Инженерная школа информационных технологий и робототехники

Направление «Информационные системы и технологии»

**Описание технологии разработки портала**

**«Wiki-словарь»**

Выполнил:

Студент гр. 8И6А Толпикин Н.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

Доцент ОИТ Цапко С.Г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Томск 2021

Оглавление

[1. Аннотация 3](#_Toc68574474)

[2. Паттерн MVC 3](#_Toc68574475)

[2.1 Общее описание 3](#_Toc68574478)

[2.2 Модель 4](#_Toc68574479)

[2.3 Представление 4](#_Toc68574480)

[2.4 Контроллер 4](#_Toc68574481)

[3. Фреймворк ASP.Net Core 5](#_Toc68574482)

[3.1 Роутинг 5](#_Toc68574484)

[3.2 Связывание моделей 6](#_Toc68574485)

[3.3 Валидация моделей 7](#_Toc68574486)

[3.4 Фильтры 8](#_Toc68574487)

[3.5 Движок представления Razor 8](#_Toc68574488)

[3.6 Строго типизированные представления 9](#_Toc68574489)

[3.7 Тег-хелперы 9](#_Toc68574490)

[3.8 Работа с сессией 10](#_Toc68574491)

[3.9 Защита от межсайтовой подделки запросов 11](#_Toc68574492)

[3.10 Файловое хранилище 11](#_Toc68574493)

[3.11 Обработка событий 12](#_Toc68574494)

[3.12 ORM 12](#_Toc68574495)

[3.13 Использование миграций 12](#_Toc68574496)

[3.14 Аутентификация и авторизация 13](#_Toc68574497)

[3.15 CSS и Bootstrap, JQuery 14](#_Toc68574498)

[3.16 Панель навигации 14](#_Toc68574499)

[3.17 Система сеток 14](#_Toc68574500)

[3.18 Панель администратора 14](#_Toc68574501)

# Аннотация

Для написания серверной части использовался ASP.NET Core MVC - фреймворк для создания веб приложений и API с помощью паттерна Model-View-Controller. Фреймворк предлагает основанный на паттерне способ создания динамических веб сайтов, где включено чистое разделение ответственности. Также в его функции входит полный контроль над разметкой, поддержка TDD разработки и последних веб стандартов.

# Паттерн MVC



## Общее описание

Архитектурный паттерн Model-View-Controller (MVC) делит приложение на три основные группы компонентов: Models (модели), Views (представления) и Controllers (контроллеры). Этот паттерн помогает достичь разделения ответственности. С помощью этого паттерна пользовательские запросы передаются контроллеру, который отвечает за работу с моделью для получения результата запроса. Затем контроллер выбирает представление, чтобы отобразить результат, и передает ему из модели необходимые данные.

На Рисунке 1 показаны три основных компонента и существующие между ними связи.

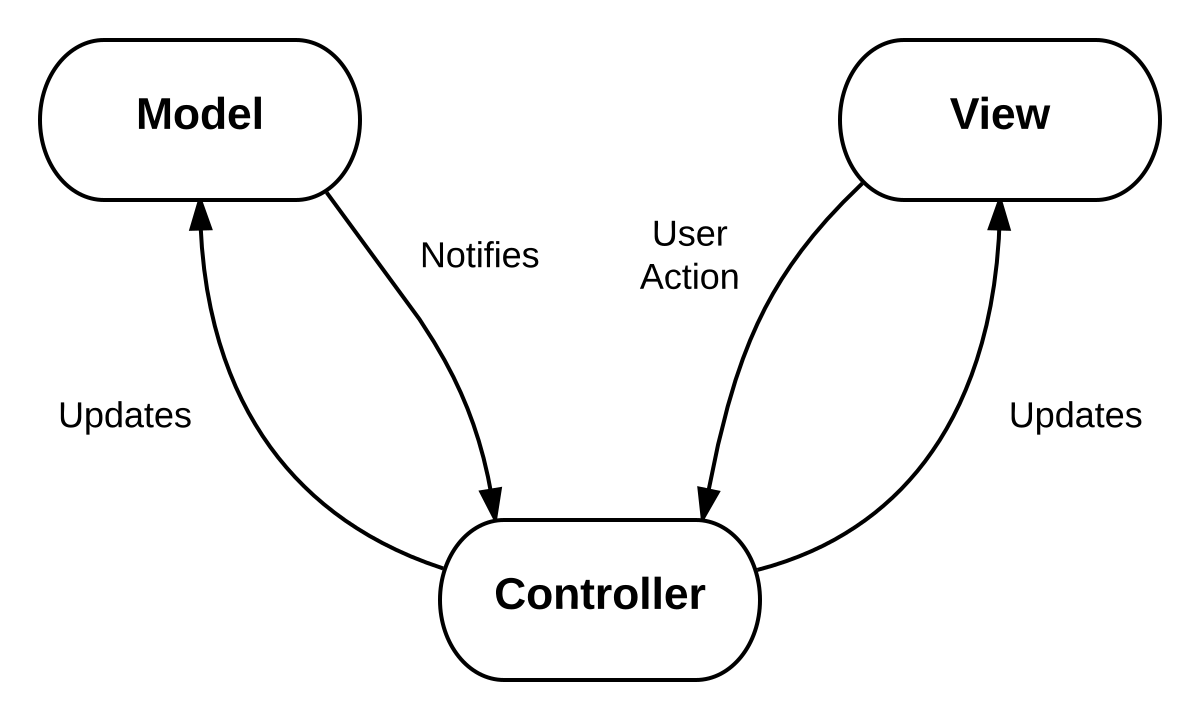


Рисунок – Паттерн MVC

## Модель

Модель в приложении MVC представляет состояние приложения и бизнес-логику или операций, которые должны в нем выполняться. Бизнес-логика должна быть включена в состав модели вместе с логикой реализации для сохранения состояния приложения. Как правило, строго типизированные представления используют типы ViewModel, предназначенные для хранения данных, отображаемых в этом представлении. Контроллер создает и заполняет эти экземпляры ViewModel из модели.

## Представление

Представления отвечают за отображение контекста в пользовательском интерфейсе. Они используют движок представления Razor, чтобы включить .NET код в HTML разметку. В представлениях должно быть по минимуму логики, и эта логика должна касаться отображаемого контента. Если вам нужно внести большой кусок логики в файлы представлений, чтобы отобразить данные из сложной модели, используйте компонент представления, ViewModel или шаблон, чтобы упростить представление.

## Контроллер

Контроллеры – это компоненты, которые обрабатывают пользовательские запросы, работают с моделью и выбирают представление для отображения контента. В MVC приложении представление только отображает информацию, а контроллер обрабатывает пользовательские запросы. В MVC паттерне контроллер является начальной точкой входа и отвечает за выбор типов моделей, с которыми он будет работать, и представлений.

# Фреймворк ASP.Net Core

ASP.Net Core – это веб-инфраструктура с открытым исходным кодом, оптимизированная для облачных вычислений, для разработки современных веб-приложений, которые можно разрабатывать и запускать на Windows, Linux и Mac. Фреймворк обладает следующими особенностями:

* Приложения ASP.Net Core могут работать на .Net Core или на полной .Net Framework.
* Спроектирован для обеспечения оптимизированной среды разработки для приложений, которые развертываются в облаке или запускаются локально.
* Состоит из модульных компонентов с минимальными накладными расходами, поэтому сохраняется гибкость при построении ваших решений.
* Возможна разработка и запуск кроссплатформенные приложений ASP.Net Core в Windows, Mac и Linux.

Рассмотрим составные компоненты этого фреймворка.



## Роутинг

В ASP.Net Core MVC активно используется ASP.Net Core роутинг, мощный компонент URL-маппинга, позволяющий создавать приложения с понятными и четкими URL. Вы определяете паттерны именования URL, которые хороши для SEO и генерирования ссылок, не принимая во внимание то, как организованы файлы на сервере. Вы можете определять роуты, используя удобный шаблонный синтаксис, при котором поддерживаются ограничения, значения по умолчанию и дополнительные значения.

Роутинг, основанный на соглашении позволяет глобально определить те форматы URL, которые принимает ваше приложение, и определяет, как эти форматы согласуются с конкретными методами действия контроллера. После получения входящего запроса механизм роутинга парсит URL, связывает его с одним из определенных URL форматов, а затем вызывает подходящий метод действия контроллера.

Атрибутивный роутинг позволяет указать роутовую информацию, передав контроллерам и методам действия атрибуты, которые определяют роуты приложения. Это обозначает, что определения располагаются рядом с контроллером и действиями, с которыми они связаны.

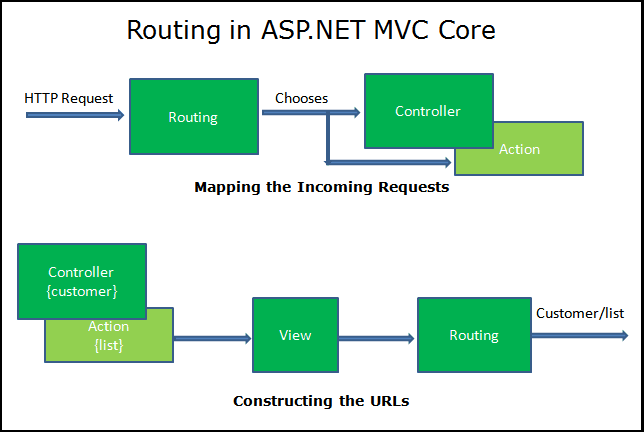


Рисунок – Роутинг в ASP.Net Core MVC

## Связывание моделей

Связывание моделей ASP.Net Core MVC конвертирует данные клиентского запроса (значения из форм, роутовые данные, параметры строки запроса, HTTP заголовки) в объекты, которые может обработать контроллер. В результате этого контроллер не должен выяснять, что за входящие данные ему пришли; данные просто передаются его методам действия в виде параметров.

## Валидация моделей

ASP.Net Core MVC поддерживает валидацию, присваивая объекту модели атрибуты валидации DataAnnotations. С помощью атрибутов валидации проверка происходит со стороны клиента, прежде чем значения будут отправлены на сервер, а затем проверка происходит со стороны сервера, прежде чем будет вызван метод действия контроллера.

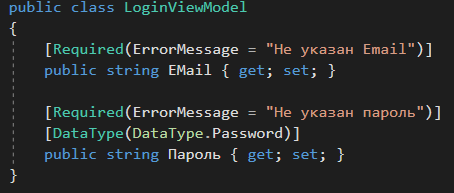


Рисунок – Валидация модели

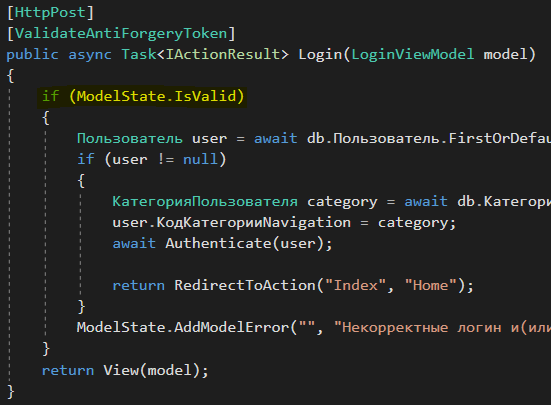


Рисунок – Проверка валидации на контроллере

Фреймворк обработает валидируемые данные запроса и со стороны клиента, и со стороны сервера. Логика валидации добавляется отображаемым представлениям в виде аннотаций и реализуется в браузере с помощью валидации jQuery.

## Фильтры

Фильтры помогают разработчикам инкапсулировать такие вещи как обработка ошибок или авторизация. Фильтры запускают для методов действий пользовательскую “до” и “после” обработку, и их можно настроить так, чтобы они запускались в конкретной точке в потоке выполнения запроса. Фильтры можно применять к контроллерам или методам действий в виде атрибутов (либо их можно запускать глобально). Некоторые фильтры (например, Authorize) включены во фреймворк.

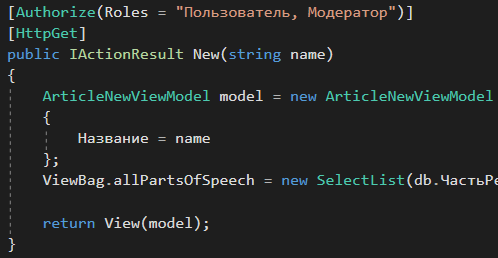


Рисунок – Применение фильтра Authorize для метода контроллера

## Движок представления Razor

В ASP.NET Core MVC представлениях для отображения используется движок представления Razor. Razor - это компактный, выразительный язык разметки шаблонов для определения представлений, использующий встроенный C# код. Razor используется для динамического генерирования веб контента на сервере. Razor позволяет смешивать серверный и клиентский код, не загрязняя его. Также он поддерживает HTML5.

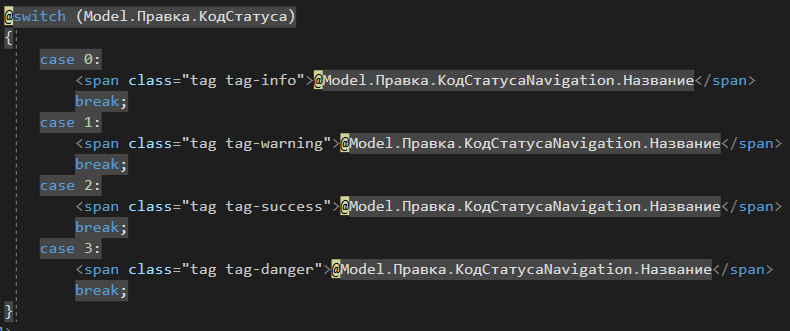


Рисунок – Применение движка Razor в представлении

## Строго типизированные представления

Razor представления в MVC могут быть строго типизированными. Контроллер может передать представлению строго типизированную модель, и тогда будет проверяться тип представления, а также включаться поддержка IntelliSense.

Например, следующее представление определяет модель типа «Wiki.СловарнаяСтатья» (Рисунок 7).

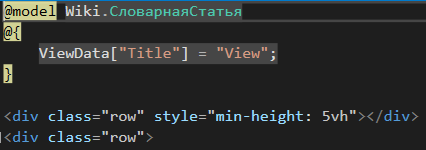


Рисунок – Привязка представления к модели

## Тег-хелперы

Тег-хелперы позволяют серверному коду принимать участие в создании и отображении HTML элементов в Razor файлах. Тег-хелперы можно использовать для определения пользовательских тегов или для изменения поведения существующих тегов. Тег-хелперы привязываются к конкретным элементам, основываясь на имени элемента и его атрибутах. Они дают преимущества отображения со стороны сервера, сохраняя возможности HTML редактирования.

Для общих задач существует множество встроенных тег-хелперов - для создания форм, ссылок, загрузки активов и так далее - и еще больше доступно в открытых GitHub репозиториях и в виде NuGet пакетов. Тег-хелперы работают с HTML элементами, основываясь на имени элемента, имени атрибута или родительском теге. Например, встроенный LinkTagHelper можно использовать для создания ссылки к методам действия Login и Register контроллера AccountController (Рисунок 8):

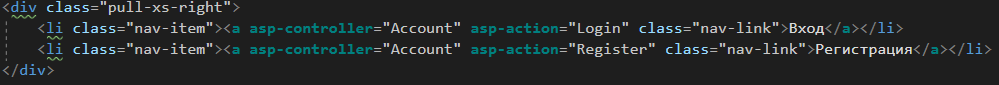


Рисунок – Тег-хелперы к методам AccountController

## Работа с сессией

В ASP.Net Core MVC существует специальный объект для работы с сессией – HttpContext.Session. Абсолютно для каждого нового запроса на сервер (неважно, разные это клиенты или один) фреймворк генерирует уникальный идентификатор сессии. Для того, чтобы ASP.Net Core держал сессию, необходимо две вещи:

1. Разрешить сохранение куки – для хранения автоматически генерируемого идентификатора сессии в них;
2. Записать в хранилище сессии любое значение (пример на Рисунке 9);

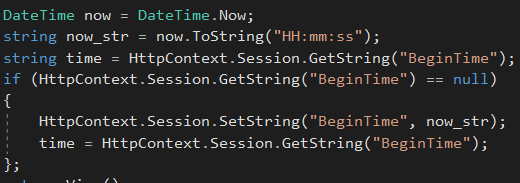


Рисунок – Пример записи в хранилище сессии значения текущего времени

## Защита от межсайтовой подделки запросов

В ASP NET фильтр ValidateAntiforgeryToken предназначен для противодействия подделке межсайтовых запросов, производя верификацию токенов при обращении к методу действия. Эти токены генерируются для каждого запроса, защищенного этим фильтром и хранятся в куках. Наиболее частым случаем является применение данного фильтра к методам, отвечающим за авторизацию.

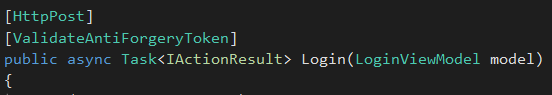


Рисунок – Пример использование фильтра ValidateAntiforgeryToken

## Файловое хранилище

Для хранения файлов различного рода используется каталог wwwroot проекта. Для получения абсолютного пути до этого каталога используется метод HostingEnvironment.WebRootPath(). Пример сохранения файла на Рисунке 11.

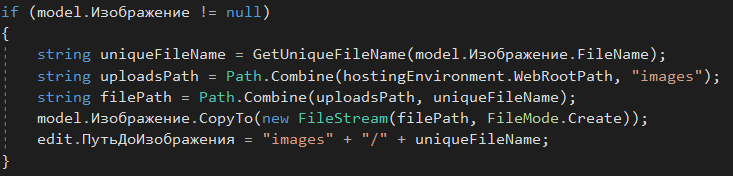


Рисунок – Пример сохранения изображения

## Обработка событий

В ASP.Net Core не реализован шаблон порождения событий, как например в Laravel. Обработка событий ограничивается только событиями элементов представлений. Такую обработку можно реализовать, например, при помощи скриптов на JQuery (Рисунок 12).

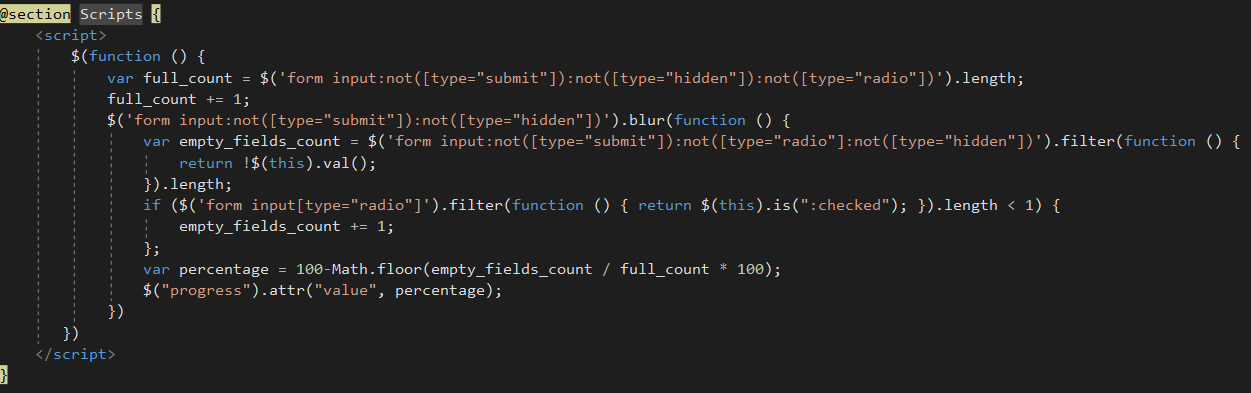


Рисунок – Обработка события смены фокуса элемента

## ORM

Для создания БД применяется паттерн Code-First. Это означает, что структура БД задается в коде приложения (классе контекста и моделях), а потом динамически создается в подключенной к веб-приложению БД средствами ORM. ORM Entity Framework Core «из коробки» поставляется с фреймворком ASP.Net Core MVC.

## Использование миграций

Для внесения изменений в структуру таблицы использовался модуль миграций Entity Framework Core. Для создания миграции достаточно внести изменения в класс контекста и применить в командной строке (в папке проекта) следующие команды:

• dotnet ef migration add <название\_миграции>

• dotnet ef database update

После этого будет создан класс с данными миграции и таблица в БД с информацией о проведенных миграциях. В качестве СУБД используется MS SQL 2019 EXPRESS.

## Аутентификация и авторизация

В ASP.Net Core MVC существует несколько механизмов аутентификации:

* Куки - ASP.NET Core имеет встроенную поддержку аутентификации на основе куки. Для этого в ASP.NET определен специальный компонент middleware, который сериализует данные пользователя в зашифрованные аутентификационные куки и передает их на сторону клиента. При получении запроса от клиента, в котором содержатся аутентификационные куки, происходит их валидация, десериализация и инициализация свойства User объекта HttpContext.
* Claim – представляют собой объекты типа «ключ-значение», которые привязываются к HttpContext. Несколько объектов Claim можно объединить в класс ClaimIdentity, который представляет собой набор полей для пользователя, которые могут быть потом использованы для авторизации.

Авторизация происходит в основном при помощи фильтра [Authorize], в котором можно уточнить любое из полей Claim. Пример на Рисунке 13.

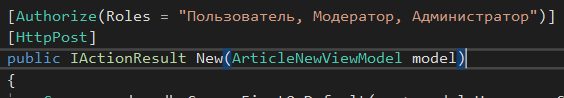


Рисунок – Авторизация при помощи фильтра

## CSS и Bootstrap, JQuery

При создании представлений с помощью движка Razor можно подключать таблицы стилей (например, Bootstrap), как из локального хранилища, так и из CDN (через внешние ссылки). Таким же образом можно подключать JS-скрипты (например, JQuery).

## Панель навигации

Панель навигации реализована при помощи мастер-шаблона и класса navbar из Bootstrap.

## Система сеток

В проекте используется стандартная система сеток Bootstrap (классы row и col).

## Панель администратора

В ASP.Net Core MVC не предоставляется «из коробки» панель администратора. Для ручного ее создания была использована автоматическая генерация контроллеров и представлений с CRUD-операциями для классов моделей.