**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,**

**СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Санкт-Петербургский колледж телекоммуникаций им. Э.Т.**

Курсовая работа

Выполнил студент: 4 курса,группы ЗФ-95

Чистяков Константин Вячеславович

Руководитель: Кривоносова Н.В.

Санкт-Петербург, 2022 г.

**Оглавление**

[Введение 3](#_Toc136906663)

[1 Формирование требований к программному продукту 4](#_Toc136906664)

[1.1 Задача 4](#_Toc136906665)

[1.2 Бизнес-требования 4](#_Toc136906666)

[1.3 Пользовательские требования 5](#_Toc136906667)

[1.4. Функциональные требования 6](#_Toc136906668)

[1.5. Нефункциональные требования 7](#_Toc136906669)

[1.6 Ограничения 8](#_Toc136906670)

[1.7 Программные средства разработки 9](#_Toc136906671)

[1.8 Аппаратные средства разработки 10](#_Toc136906672)

[2. Архитектура системы 11](#_Toc136906673)

[2.1 Общая структура проекта 11](#_Toc136906674)

[2.1.2 Диаграммы классов: 13](#_Toc136906675)

[2.2. Моделирование основных сценариев системы 19](#_Toc136906676)

[2.2.1 Система регистрации пользователя 19](#_Toc136906677)

[2.2.2 Система создания курса 20](#_Toc136906678)

[2.2.3 Система прохождения теста 22](#_Toc136906679)

[2.2.4 Диаграмма деятельности создания урока 23](#_Toc136906680)

[2.2.5 Диаграмма деятельности создания теста 24](#_Toc136906681)

[2.2.6 Диаграмма деятельности: регистрация пользователя 26](#_Toc136906682)

[2.2.7 Проектирование вариантов использования 27](#_Toc136906683)

[3 Проектирование программных модулей 28](#_Toc136906684)

[3.1 Уровень Domain 28](#_Toc136906685)

[3.2 Уровень DAL 32](#_Toc136906686)

[3.3 Уровень Service 37](#_Toc136906687)

[3.4 Уровень Controller 45](#_Toc136906688)

[3.5 Уровень view 47](#_Toc136906689)

[Заключение 53](#_Toc136906690)

[Список источников информации 54](#_Toc136906691)

# Введение

Современная сфера информационных технологий и программирования становится все более значимой и востребованной. Большой спрос на специалистов, владеющих языком программирования Python, приводит к необходимости разработки образовательных сервисов, способствующих эффективному изучению данного языка.

Глобальные тенденции в области цифровой трансформации требуют принципиальных решений в развитии образования и доступности обучения программированию. Национальные стратегии, такие как "Цифровая экономика" и "Цифровое образование", вносят значительный вклад в развитие информационных технологий в Российской Федерации.

Дипломный проект по разработке образовательного сервиса по изучению языка программирования Python является актуальным и стратегически важным.

Целью данного проекта является создание онлайн-платформы, которая предоставит пользователям удобный и эффективный инструмент для освоения языка Python.

Провести информационное исследование выбранной предметной области на выявление конкурентов-аналогов, систем со схожим функционалом;

Основные задачи проекта включают в себя:

1. сформулировать требования к проектируемому программному продукту, включающие в себя:
   1. быстрый доступ к информации;
   2. возможность ознакомления с наиболее значимыми работами каждого деятеля искусства;
   3. получение теоретической информации о языке программирования python;
   4. практическое выполнение заданий в браузере;
2. разработать модель программного продукта, включая:
   1. UML-диаграммы:
      1. вариантов использования (use-case);
      2. деятельности (activity);
      3. базы данных (erd);
      4. последовательностей (sequence);
   2. макеты пользовательского интерфейса приложения, выполненные в графическом редакторе;
3. разработать основной функционал программного продукта для организации виртуального пространства просмотра контента;
4. провести тестирование программного продукта;
5. выполнить технико-экономическое обоснование программного продукта;
6. подготовить документационный материал по процессу работы над проектом;
7. обосновать выпускную квалификационную работу.

## 1 Формирование требований к программному продукту

## 1.1 Задача

Необходимо разработать интуитивно понятный веб-интерфейс, с

возможностью ввода, чтения и редактирования информации. Интерфейс должен быть понятен минимально подготовленному пользователю.

Данные должны храниться централизованно в базе данных. Информация

заложенная в систему должна храниться в одном месте и предоставляться, добавляться и редактироваться пользователем в соответствии с его запросами.

Необходима возможность просмотра списка результатов повышения квалификации,

а также печати документов и отчетов.

Внешний интерфейс системы должен соответствовать современным стандартам

оформления web-приложений. Быть интуитивно понятным и не перегруженным излишней информацией. Компоновка пользовательских элементов на экране должна быть такой, чтобы пользователь мог легко воспринимать весь объем поступающей к нему информации, оценивать ситуацию и быстро предпринимать все необходимые действия.

Взаимодействие пользователя с программой должно осуществляться в браузерном

режиме. Функциональное назначение интуитивно непонятного элемента программы должно быть дополнительно пояснено посредством надписей там, где это возможно, и системы контекстных подсказок.

## 1.2 Бизнес-требования

Функциональность:

* Создание удобного и интуитивно понятного интерфейса пользователя для обучения языку программирования Python.
* Предоставление структурированного и качественного образовательного контента, включая учебные материалы, примеры кода, задания и тесты.
* Возможность практического применения полученных знаний через интерактивные задачи и проекты.
* Поддержка мониторинга и оценки прогресса учащихся, включая систему отслеживания выполненных заданий и предоставление обратной связи.
* Интеграция с популярными инструментами разработки Python, такими как IDE и онлайн-редакторы кода.

Безопасность:

* Защита персональных данных пользователей, соблюдение приватности и конфиденциальности информации.
* Применение мер безопасности для предотвращения несанкционированного доступа, хакерских атак и утечек данных.

Масштабируемость и производительность:

* Обеспечение возможности расширения функциональности сервиса и добавления нового образовательного контента.
* Оптимизация производительности и скорости работы сервиса при обработке большого количества пользователей и одновременных запросов.

Адаптивность и доступность:

* Обеспечение кросс-платформенности и поддержки различных устройств, включая компьютеры, планшеты и мобильные устройства.
* Удобство использования для людей с ограниченными возможностями и соблюдение принципов доступности контента.

## 1.3 Пользовательские требования

Регистрация и авторизация:

* Возможность создания учетной записи пользователя с указанием логина и пароля.
* Авторизация пользователя для доступа к персональным данным и сохраненному прогрессу обучения.

Обучение и учебные материалы:

* Предоставление структурированного образовательного материала по языку программирования Python.
* Понятные и интерактивные учебные модули, охватывающие основные концепции и темы программирования.
* Примеры кода, объяснения и демонстрации работы программ для лучшего понимания материала.
* Возможность прохождения практических.

Мониторинг и оценка прогресса:

* Отслеживание прогресса пользователя в изучении Python, включая выполненные задания и результаты тестов.
* Предоставление обратной связи и рекомендаций для улучшения навыков программирования.
* Интерактивные функции:
* Возможность задавать вопросы и получать поддержку от экспертов или сообщества.
* Интерактивные упражнения и игры для активного погружения в изучение языка программирования.
* Возможность общения и сотрудничества с другими учащимися, обмен опытом и решениями.

Доступность и адаптивность:

* Поддержка различных устройств и платформ, включая компьютеры, планшеты и мобильные устройства.
* Адаптивный дизайн и удобство использования на разных размерах экранов.
* Поддержка доступности для пользователей с ограниченными возможностями.
* Пользовательский опыт:
* Интуитивно понятный интерфейс и навигация, чтобы пользователи могли легко ориентироваться в сервисе.
* Приятный и привлекательный дизайн, способствующий мотивации и заинтересованности в обучении.
* Персонализация контента и настройки для адаптации под индивидуальные потребности пользователей.

## 1.4. **Функциональные** требования

* Регистрация и аутентификация пользователей: Разработка обязательной процедуры регистрации пользователей, включая создание аккаунтов и аутентификацию с помощью логина и пароля.
* Учебные материалы: Предоставление обучающих материалов, таких как учебные видео, интерактивные уроки, текстовые инструкции и примеры кода, для изучения основных концепций и возможностей языка программирования Python.
* Практические задания: Предоставление пользователю интерактивных практических заданий, которые помогут им применить изученные концепции и навыки на практике. Система должна быть способна оценивать и предоставлять обратную связь по выполненным заданиям.
* Прогресс и достижения: Отслеживание прогресса пользователей в изучении Python, включая сохранение и отображение достижений, завершенных уроков и выполненных заданий. Пользователи должны иметь возможность видеть свой прогресс и уровень своих навыков.
* Форум и поддержка: Создание форума или чата, где пользователи могут общаться друг с другом, задавать вопросы и получать поддержку от опытных пользователей или инструкторов.
* Адаптивный дизайн: Разработка пользовательского интерфейса, который адаптируется к различным устройствам и позволяет пользователям удобно пользоваться сервисом на разных платформах, включая компьютеры, планшеты и мобильные устройства.
* Интеграция среды разработки: Предоставление пользователю интегрированной среды разработки (IDE) или возможность подключения внешних IDE для написания и исполнения кода на Python.

## 1.5. Нефункциональные требования

* Производительность: Сервис должен быть способен обрабатывать большое количество одновременных пользователей и обеспечивать быструю загрузку учебных материалов и выполнение заданий.
* Надежность: Система должна быть стабильной и надежной, минимизируя возможность сбоев и ошибок. Резервное копирование данных и механизмы восстановления должны быть предусмотрены для предотвращения потери информации.
* Безопасность: Защита пользовательских данных и конфиденциальности является важным аспектом. Система должна обеспечивать безопасную передачу данных, использовать механизмы аутентификации и авторизации, а также защищать от потенциальных угроз безопасности, таких как взлом или внедрение злонамеренного кода.
* Масштабируемость: Сервис должен быть способен масштабироваться с ростом числа пользователей и обрабатываемых данных. Архитектура и инфраструктура должны быть гибкими и масштабируемыми.
* Удобство использования: Пользовательский интерфейс и навигация должны быть интуитивно понятными и удобными для использования. Система должна быть доступна для пользователей с разным уровнем опыта в программировании.
* Совместимость: Сервис должен быть совместимым с различными веб-браузерами и операционными системами, чтобы пользователи могли получить доступ к нему с любого устройства.
* Сопровождение и поддержка: Обеспечение постоянной поддержки и технического обслуживания сервиса для решения возникающих проблем и вопросов пользователей.

## 1.6 Ограничения

Бюджетные ограничения:

* Разработка программного продукта должна быть выполнена в пределах определенного бюджета, с учетом затрат на разработку, тестирование, поддержку и обновление.
* Расходы на оборудование, серверное обслуживание и инфраструктуру должны быть учтены в рамках бюджета проекта.

Ограничения функциональности:

* Система должна обеспечивать работу с несколькими пользователями одновременно и поддерживать потенциальный рост количества пользователей.
* Скорость обработки запросов и загрузки контента должна быть достаточно высокой, чтобы обеспечить плавное и удобное взаимодействие с сервисом.
* Объем хранимых данных должен быть масштабируемым и достаточным для хранения информации о пользователях, их прогрессе в обучении и других необходимых данных.

Ограничения безопасности:

* Сервис должен соблюдать требования по защите персональных данных пользователей в соответствии с применимыми законодательными нормами и политиками конфиденциальности.
* Необходимо обеспечить защиту от несанкционированного доступа, взломов и других угроз безопасности данных и системы.

Ограничения по платформе и технологиям:

* Сервис должен быть доступным и совместимым с различными операционными системами, веб-браузерами и устройствами, включая компьютеры, планшеты и мобильные телефоны.
* Выбранные технологии и инструменты для разработки и развертывания должны соответствовать требованиям проекта и быть эффективными в использовании ресурсов.
* Важно учесть масштабируемость и гибкость системы, чтобы она могла адаптироваться к будущим потребностям и изменениям в технологической среде.
* Ограничения помогают определить рамки и параметры проекта, чтобы обеспечить его успешное выполнение и соответствие требованиям заказчика и пользователей.

## 1.7 Программные средства разработки

Для разработки будут использоваться следующие программные средства:

Visual Studio: Интегрированная среда разработки (IDE) от Microsoft, которая обеспечивает широкие возможности для создания и отладки программного кода. Visual Studio предоставляет поддержку различных языков программирования, включая C#.

Visual Studio Code: Легковесный редактор кода с расширенными возможностями и широкой поддержкой языков программирования. Visual Studio Code обеспечивает удобную среду разработки и настройку проекта.

C#: Язык программирования, разработанный Microsoft, широко используемый для создания приложений на платформе .NET. C# обладает сильной типизацией и широкими возможностями объектно-ориентированного программирования.

HTML, CSS, JS: Технологии веб-разработки, которые используются для создания пользовательского интерфейса веб-сервиса. HTML отвечает за структуру страницы, CSS - за оформление, а JavaScript - за интерактивность и динамическое поведение.

jQuery: Библиотека JavaScript, которая упрощает манипуляцию с HTML-элементами, обработку событий и взаимодействие с сервером. jQuery упрощает разработку клиентской части веб-приложения.

Bootstrap: Фреймворк для разработки адаптивных и стильных веб-интерфейсов. Bootstrap предоставляет набор готовых компонентов, которые можно использовать для быстрой и согласованной разработки пользовательского интерфейса.

AJAX: Технология, позволяющая асинхронно обмениваться данными между веб-сервером и клиентской частью приложения без перезагрузки страницы. AJAX обеспечивает более плавное и отзывчивое взаимодействие с сервером.

MySQL: Система управления базами данных, которая будет использоваться для хранения данных приложения. MySQL является популярным выбором для веб-приложений благодаря своей производительности и простоте использования.

Выбор данных программных средств обусловлен их широкой популярностью, функциональностью и поддержкой, а также их соответствием требованиям проекта. Они обеспечат эффективную разработку, тестирование и отладку приложения, а также создадут удобный и современный пользовательский интерфейс.

## 1.8 Аппаратные средства разработки

Для разработки веб-приложения использовался ПК со следующими характеристиками:

1. процессор — intel core i9-9900k;
2. оперативная память — 32 ГБ;
3. объем накопителя (SSD) — 1024 ГБ;
4. операционная система —windows 11.

# 2. Архитектура системы

## 2.1 Общая структура проекта

Общая архитектура приложения основана на паттерне MVC (Model-View-Controller) и включает следующие уровни: Data Access Layer (DAL), Domain Layer, Service Layer и само приложение.

Взаимодействие пользователей с приложением начинается через веб-браузер. Пользователи открывают веб-страницы в браузере и взаимодействуют с интерфейсом приложения, например, заполняют формы, нажимают кнопки и отправляют запросы на сервер.

Когда пользователь отправляет запрос на сервер, браузер использует протокол HTTP для передачи этого запроса на ASP.NET приложение. ASP.NET является серверной платформой для разработки веб-приложений и принимает запрос от браузера.

ASP.NET взаимодействует с Razor страницами, которые представляют собой шаблоны для генерации динамического HTML-кода. Razor страницы содержат код на языке программирования C# внутри HTML-разметки. Они позволяют динамически формировать веб-страницы, используя данные из моделей и вью моделей.

Entity Framework является ORM (Object-Relational Mapping) инструментом, который обеспечивает связь между базой данных и объектами моделей приложения. Entity Framework позволяет работать с базой данных через объектно-ориентированный подход, предоставляя средства для создания, чтения, обновления и удаления данных.

Модели представляют собой классы, которые описывают структуру данных в приложении. Они представляют сущности, с которыми работает приложение, например, пользователи, курсы, задания и т.д. Модели определяют свойства и методы, необходимые для работы с данными.

Вью модели (View Models) используются для передачи данных между контроллерами и представлениями (Razor страницами). Они представляют собой специальные классы, которые содержат только те данные, которые необходимы для отображения на странице. Вью модели позволяют разделять логику представления и бизнес-логику, упрощая управление данными в приложении.

Сущности Entity Framework взаимодействуют с моделями и базой данных. Они представляют таблицы и связи в базе данных и обеспечивают выполнение операций над данными, таких как создание, чтение, обновление и удаление записей в базе данных.

Общая архитектура представлена на рисунке 2.1

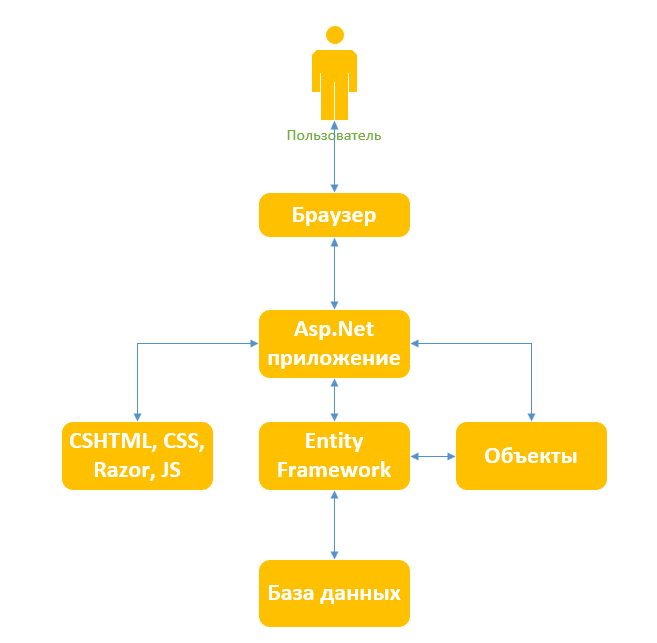


Рисунок 2.1 Общая схема взаимодействия объектов

Архитектура приложения также включает уровни DAL, Domain и Service. Data Access Layer (DAL) отвечает за взаимодействие с базой данных, выполняет запросы и обеспечивает доступ к данным. Domain Layer содержит бизнес-логику приложения, включая правила валидации данных, обработку операций и другие бизнес-процессы. Service Layer предоставляет интерфейс для взаимодействия с бизнес-логикой и управления приложением.

Таким образом, вся архитектура приложения строится вокруг паттерна MVC, где браузер взаимодействует с ASP.NET приложением, которое использует Razor страницы, Entity Framework, модели и вью модели для обработки запросов, доступа к данным и отображения информации пользователю. Различные уровни, такие как DAL, Domain и Service, обеспечивают разделение ответственности и организацию бизнес-логики приложения.

## 2.1.2 Диаграммы классов:

Весь проект состоит из 4 уровней, 3 библиотек классов: DAL, Domain, Service, а также самого приложения. Слой Domain в архитектуре приложения выполняет роль бизнес-логики и содержит модели данных, которые представляют сущности и объекты, с которыми работает приложение. Он описывает основные правила и операции, связанные с бизнес-логикой, а также определяет связи и отношения между моделями. На рисунке 2.2 представлены классы и интерфейсы библиотеки Domain.

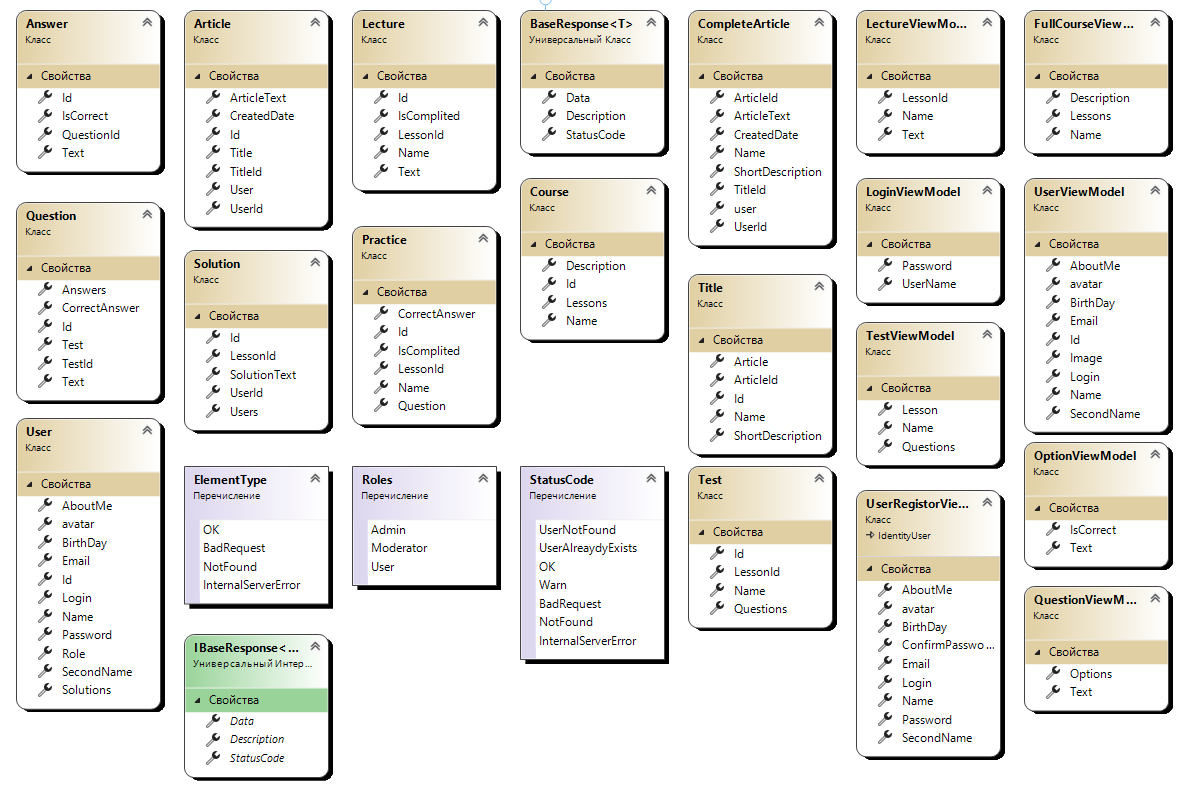


Рисунок 2.2 Диаграмма классов библиотеки Domain

В библиотеке Domain содержатся следующие модели:

Answer (Ответ): Модель, представляющая ответы на вопросы в рамках тестов и практических заданий. Она содержит свойства, такие как текст ответа, информацию о правильности ответа и связи с соответствующими вопросами.

Article (Статья): Модель, представляющая статьи или информационные материалы. Она содержит свойства, такие как, автора, краткое описание текст статьи и дату публикации.

Lecture (Лекция): Модель, представляющая лекции. Она содержит свойства, такие как название лекции, описание, содержание и связи с соответствующими курсами.

BaseResponse (Базовый ответ): Модель, представляющая базовый ответ API-запросов. Она содержит свойства, такие как статус код, сообщение и дополнительные данные.

Questions (Вопросы): Модель, представляющая вопросы, используемые в тестах и практических заданиях. Она содержит следующие свойства, текст вопроса, и правильные ответ.

Solution (Решение): Модель, представляющая решения практических заданий. Она содержит свойства, такие как текст решения и связь с соответствующим заданием.

Practice (Практика): Модель, представляющая практические задания. Она содержит свойства, такие как описание задания, условия и связь с соответствующими курсами.

Course (Курс): Модель, представляющая курсы по изучению языка программирования Python. Она содержит свойства, такие как название курса, описание, длительность и связи с соответствующими лекциями, тестами и практическими заданиями.

Title (Название): Модель, представляющая названия различных элементов приложения. Она использоваться для представления заголовков лекций, статей и других сущностей.

User (Пользователь): Модель, представляющая пользователей приложения. Она содержит свойства, такие как имя пользователя, адрес электронной почты, пароль, аватар и роли пользователя.

Test (Тест): Модель, представляющая собой тесты. Она содержит свойства, такие как название теста, описание, вопросы и связь с соответствующими курсами.

В библиотеке имеются вью модели, они представляют данные, которые используются для отображения информации на пользовательском интерфейсе и передачи данных между контроллерами и представлениями в ASP.NET MVC приложении. Они служат промежуточным слоем между моделями предметной области и представлениями.

Вот описания вью моделей, используемых в приложении:

LectureViewModel (Модель представления лекции): представляет данные, необходимые для отображения информации о лекции на пользовательском интерфейсе. Включает следующие свойства: название лекции, описание и связи с соответствующими курсами.

FullCourseViewModel (Модель представления полного курса): представляет данные, необходимые для отображения полной информации о курсе на пользовательском интерфейсе. Модель содержит следующие свойства, такие как название курса, описание, лекции, тесты и практические задания.

LoginViewModel (Модель представления для входа в систему): представляет данные, необходимые для аутентификации пользователя. Включает свойства, такие как адрес электронной почты, пароль и другую информацию, необходимую для успешного входа в систему.

UserViewModel (Модель представления пользователя): представляет данные, необходимые для отображения информации о пользователе на пользовательском интерфейсе. Включает свойства, такие как имя пользователя, адрес электронной почты, роли и другую информацию.

TestViewModel (Модель представления теста): представляет данные, необходимые для отображения информации о тесте на пользовательском интерфейсе. Включает в себя такие следующие свойства, название теста, описание, вопросы и варианты ответов.

OptionViewModel (Модель представления варианта ответа): представляет данные, необходимые для отображения варианта ответа на пользовательском интерфейсе. Модель имеет следующие свойства, текст варианта ответа и информацию о его правильности.

UserRegisterViewModel (Модель представления регистрации пользователя): представляет данные, необходимые для регистрации нового пользователя в системе. В данной модели имеются следующие свойства, имя пользователя, адрес электронной почты, пароль и другую информацию, необходимую для успешной регистрации.

QuestionViewModel (Модель представления вопроса): представляет данные, необходимые для отображения информации о вопросе на пользовательском интерфейсе. Включает свойства, такие как текст вопроса и варианты ответов.

В библиотеке имеются enum-типы, они представляют набор значений, которые могут быть использованы в различных частях приложения.

ElementType (Тип элемента): представляет типы элементов, которые могут быть использованы в системе, такие как статья, лекция, курс и т.д. Это может быть полезно для классификации и идентификации различных типов элементов.

Roles (Роли): представляет роли пользователей в системе, такие как администратор, преподаватель, студент и т.д. Это может использоваться для определения различных прав доступа и ограничений в зависимости от роли пользователя.

StatusCode (Статус код): представляет различные HTTP-статус коды, которые могут быть возвращены при обработке запросов в приложении. Это может быть полезно для определения успешности операций и обработки ошибок.

Эти классы являются важными компонентами приложения, позволяющими представлять данные, выполнять бизнес-логику и взаимодействовать с пользовательским интерфейсом, базой данных и другими частями системы. Они обеспечивают структуру и организацию данных в приложении, позволяя ему функционировать согласно требованиям проекта и бизнес-логике приложения.

Уровень DAL (Data Access Layer) представляет собой слой доступа к данным в приложении. В нем содержатся классы, которые обеспечивают взаимодействие с базой данных и предоставляют операции для работы с моделями данных. На рисунке 2.3 представлены классы и интерфейсы библиотеки DAL.

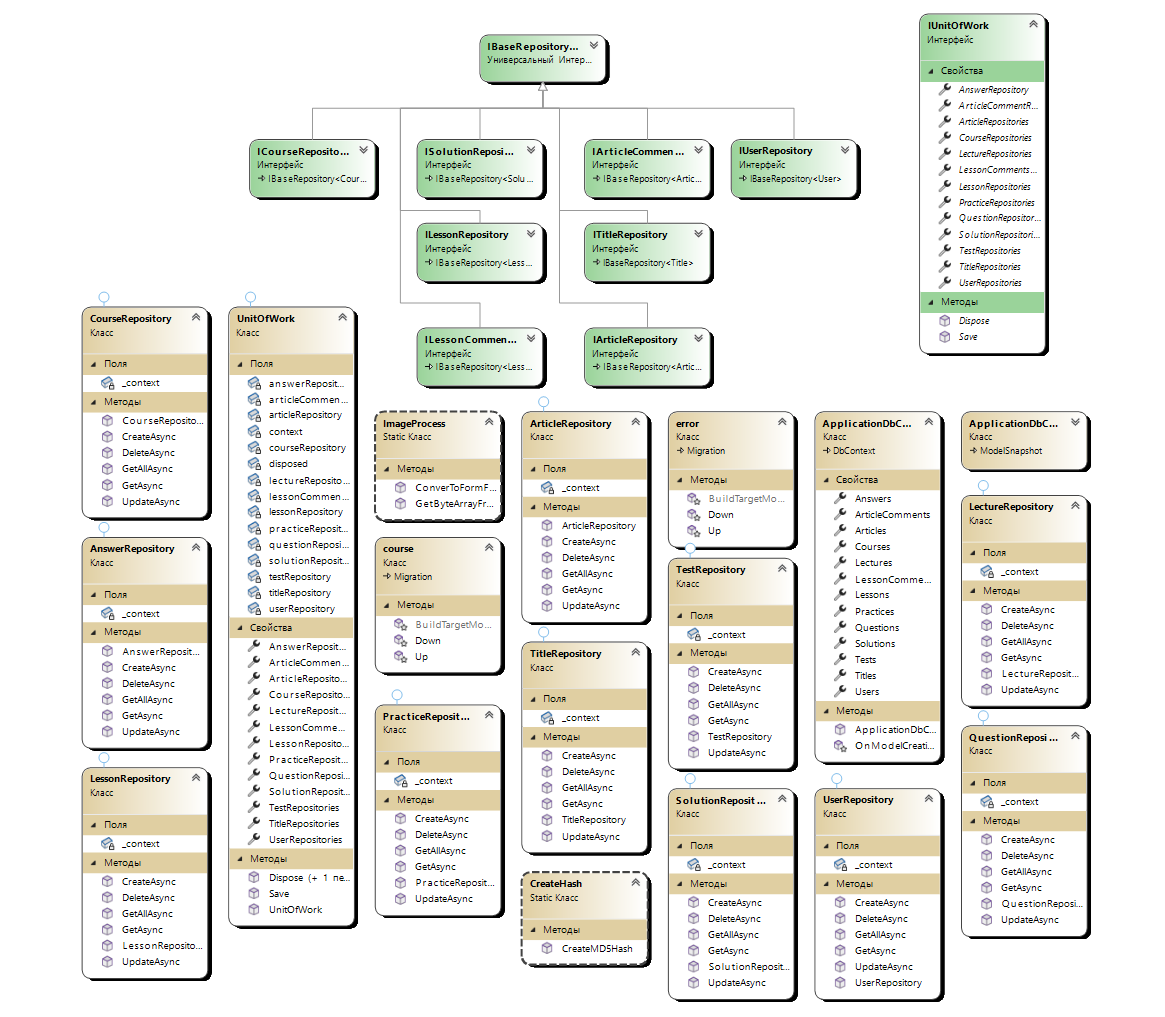


Рисунок 2.3 Диаграмма классов библиотеки Domain

Репозитории (Repositories): для каждой модели: Answer, Article, Lecture, BaseResponse, Questions, Solution, Practice, Course, Title и User, в уровне DAL имеется соответствующий класс репозитория. Репозитории предоставляют методы для выполнения операций чтения, записи, обновления и удаления данных в базе данных, связанных с определенной моделью. Каждый репозиторий реализует свой собственный интерфейс, который наследуется от интерфейса IBASERepository.

Интерфейс IBASERepository: это интерфейс, который определяет общие методы для всех репозиториев. Он включает методы, такие как Get, GetAll, Add, Update и Delete, которые позволяют выполнять соответствующие операции с данными.

UnitOfWork: Класс UnitOfWork представляет шаблон проектирования, который объединяет различные репозитории и координирует работу с ними. Он обеспечивает целостность транзакций и позволяет выполнять операции над несколькими репозиториями как единое действие.

ImageProcess: Класс ImageProcess отвечает за обработку изображений в приложении. Он содержит методы для загрузки, изменения размеров, обрезки и других операций с изображениями.

ApplicationDbContext: это класс, представляющий контекст базы данных приложения. Он наследуется от класса DbContext в Entity Framework и предоставляет доступ к базе данных, таблицам и отношениям между ними. ApplicationDbContext используется репозиториями для выполнения операций с базой данных.

CreateHash: Класс CreateHash содержит методы для создания хеш-значений и шифрования данных, таких как пароли пользователей. Он обеспечивает безопасность и защиту данных в приложении.

Уровень DAL играет важную роль в обеспечении доступа к данным, выполнении операций базы данных и управлении моделями приложения. Он работает в тесной связи с остальными уровнями архитектуры и позволяет приложению эффективно работать с данными.

Уровень Service представляет собой слой бизнес-логики приложения. В нем содержатся классы, которые предоставляют сервисные функции для работы с данными и выполнения операций, связанных с определенными моделями. На рисунке 2.4 представлены классы и интерфейсы библиотеки Service. Класс AccountService предоставляет сервисные функции для работы с аккаунтами пользователей. Он включает методы для регистрации новых пользователей, аутентификации, управления профилем и других операций, связанных с учетными записями.

Класс ArticleService предоставляет сервисные функции для работы с статьями. Он содержит методы для создания, чтения, обновления и удаления статей. ArticleService также включает операции, связанные с комментариями к статьям и взаимодействием с другими сервисами.

Класс UserService предоставляет сервисные функции для работы с пользователями. Он включает методы для получения информации о пользователях, управления правами доступа, изменения паролей и других операций, связанных с пользователями.

CourseService: Класс CourseService предоставляет сервисные функции для работы с курсами. Он содержит методы для создания, чтения, обновления и удаления курсов. CourseService также включает операции, связанные с добавлением лекций, тестов или управлением прогрессом студентов.

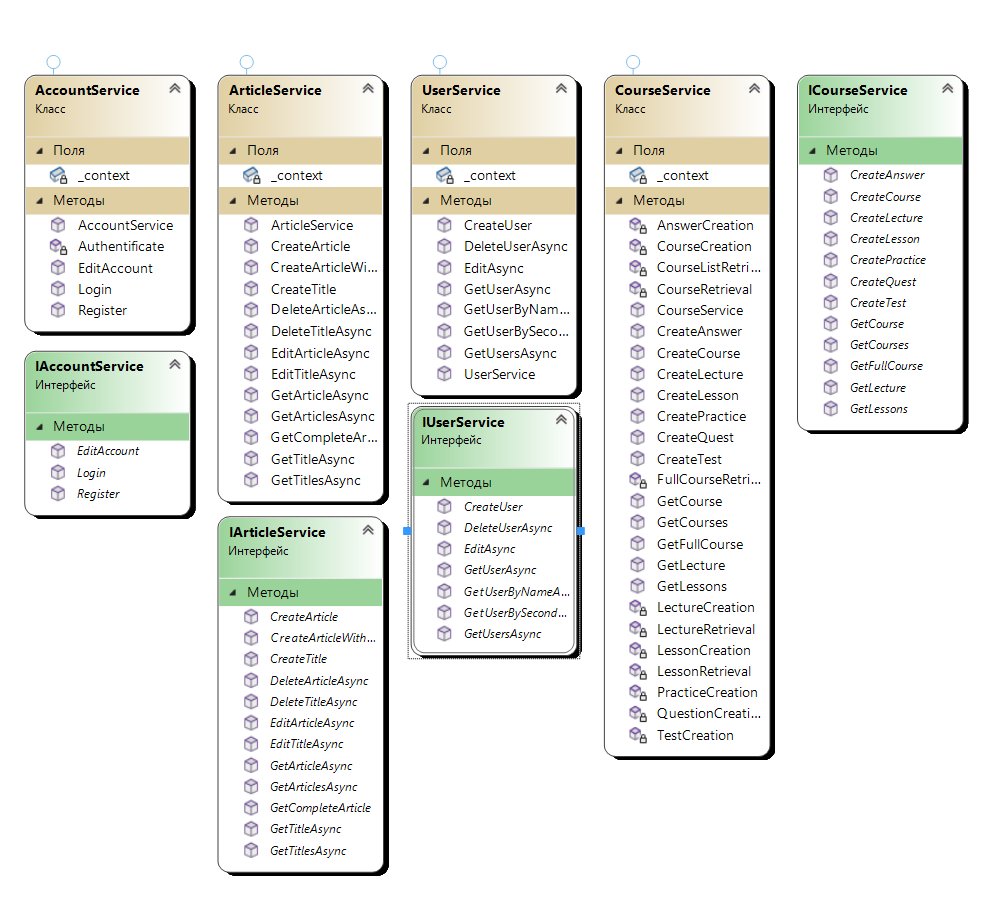


Рисунок 2.4 Диаграмма классов библиотеки Service

Каждый класс Service также имеет соответствующий интерфейс, определяющий контракты для методов, которые должны быть реализованы. Интерфейсы для классов Service могут называться так же, как сами классы, но с добавлением префикса "I" (например, IAccountService, IArticleService, IUserService, ICourseService). Это позволяет легко заменять реализацию сервисов и использовать инверсию управления (Inversion of Control), что полезно для модульного тестирования и разделения ответственности.

Уровень Service отвечает за обработку бизнес-логики, координирование работы различных сервисов и предоставление удобного интерфейса для взаимодействия с данными внутри приложения. Он служит прослойкой между уровнем DAL и слоем представления, обеспечивая централизованное управление бизнес-правилами и операциями над данными.

## 2.2. Моделирование основных сценариев системы

## 2.2.1 Система регистрации пользователя

Регистрация пользователей является одной из ключевых функциональности веб-приложений, позволяющей пользователям создавать учетные записи и получать доступ к персонализированным функциям и ресурсам. Для многих онлайн-сервисов, включая образовательные платформы, социальные сети, электронные коммерции и другие, регистрация пользователей является первым шагом в привлечении и удержании пользователей. Процесс регистрации пользователя показан на рисунке 2.5.

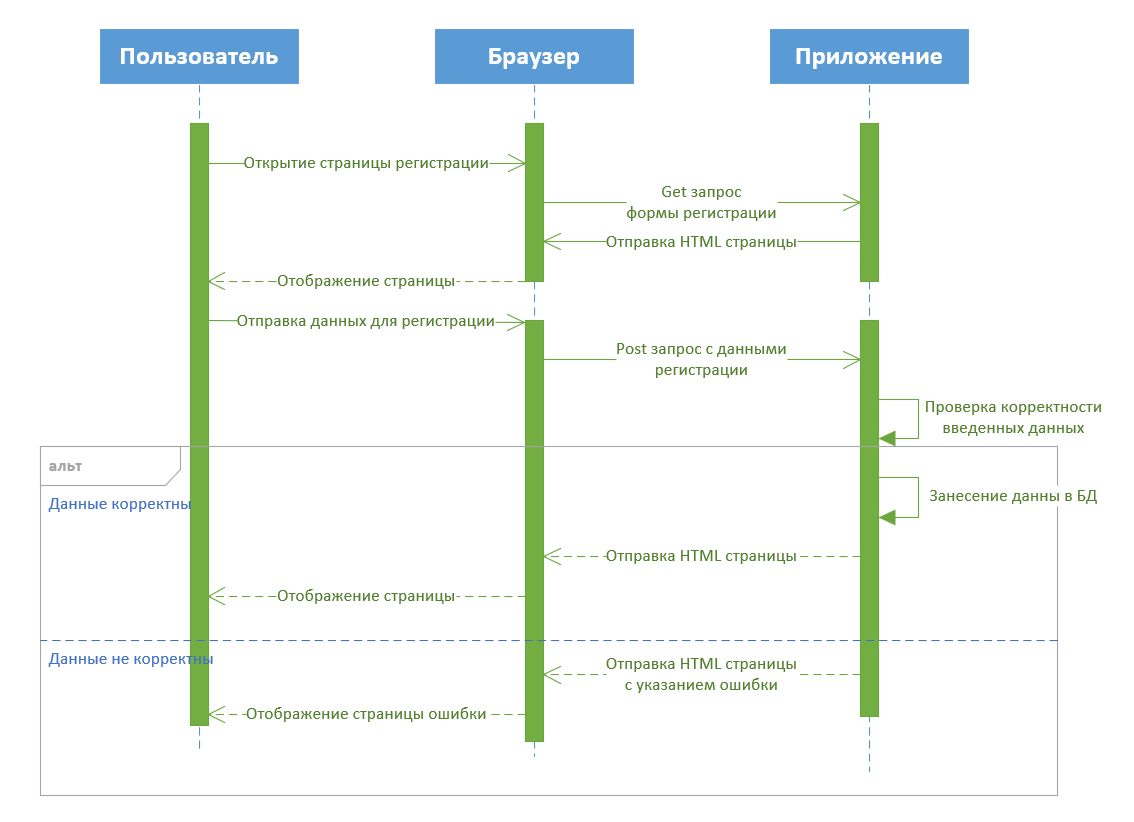


Рисунок 2.5 Диаграмма последовательности регистрации

Данная диаграмма последовательности иллюстрирует последовательность действий и взаимодействий между различными компонентами системы в процессе регистрации нового пользователя. Она отображает шаги и коммуникации, которые происходят между частями системы, такими как пользователь и веб-приложение, в целях успешной регистрации.

Actors:

Пользователь: Человек, который желает зарегистрироваться в системе.

Браузер: Веб-браузер, через который пользователь взаимодействует с веб-приложением.

Сервер: Веб-сервер, на котором расположено ASP.NET приложение.

Шаги и взаимодействия:

1. Пользователь открывает веб-браузер и переходит на страницу регистрации.
2. Браузер отправляет запрос на сервер, запрашивая страницу регистрации.
3. Сервер формирует и возвращает страницу регистрации с полями для ввода необходимой информации.
4. Пользователь вводит свои данные в поля формы регистрации, такие как имя, адрес электронной почты и пароль.
5. После заполнения полей, пользователь нажимает кнопку "Зарегистрироваться".
6. Браузер отправляет запрос на сервер, содержащий введенные пользователем данные регистрации.
7. Сервер получает запрос с данными регистрации и проводит их валидацию, проверяя наличие обязательных полей и корректность введенной информации.
8. Если данные регистрации проходят проверку, сервер создает новую запись пользователя в базе данных, сохраняя введенную информацию, включая хэшированный пароль для безопасности.
9. Сервер формирует ответ на запрос и отправляет его браузеру, сообщая о успешной регистрации.
10. Браузер отображает страницу с подтверждением успешной регистрации и предоставляет пользователю возможность войти в систему.

Взаимодействие между браузером и сервером осуществляется по протоколу HTTP. Валидация данных включает проверку формата адреса электронной почты, сравнение паролей для подтверждения, а также проверку уникальности адреса электронной почты в системе. Создание записи пользователя в базе данных осуществляется с помощью ORM-фреймворка, Entity Framework.

## 2.2.2 Система создания курса

Создание курса является одной из ключевых задач в разработке образовательного сервиса. Курсы представляют собой структурированные учебные программы, которые предоставляют пользователям доступ к материалам, урокам, заданиям и другим обучающим ресурсам.

Диаграмма на рисунке 2.6 показывает последовательность действий необходимых для создания курса.

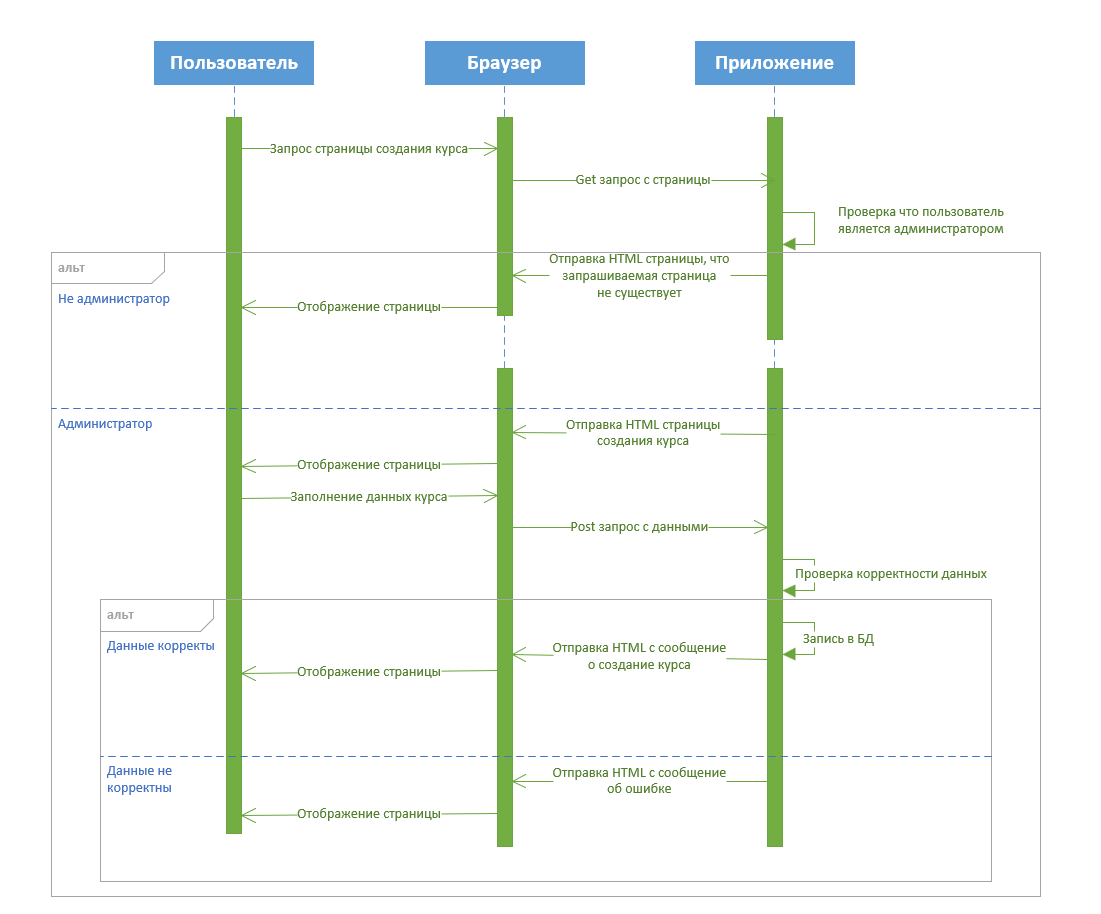


Рисунок 2.6 Диаграмма последовательности создания курс

Actors:

1. Администратор: Пользователь с привилегиями администратора, который создает курс в системе.
2. Браузер: Веб-браузер, через который администратор взаимодействует с веб-приложением.
3. Сервер: Веб-сервер, на котором расположено ASP.NET приложение.

Шаги и взаимодействия:

1. Администратор открывает веб-браузер и входит в систему как администратор.
2. Браузер отправляет запрос на сервер, запрашивая страницу создания курса.
3. Сервер формирует и возвращает страницу создания курса с полями для ввода необходимой информации о курсе.
4. Администратор вводит данные о курсе, такие как название, описание, длительность и прочие параметры.
5. После заполнения полей, администратор нажимает кнопку "Создать курс".
6. Браузер отправляет запрос на сервер, содержащий введенные администратором данные о курсе.
7. Сервер получает запрос с данными о курсе и проводит их валидацию, проверяя наличие обязательных полей и корректность введенной информации.
8. Если данные курса проходят проверку, сервер создает новую запись о курсе в базе данных, сохраняя введенную информацию.
9. Сервер формирует ответ на запрос и отправляет его браузеру, сообщая об успешном создании курса.
10. Браузер отображает страницу с подтверждением успешного создания курса и предоставляет администратору возможность продолжить работу с курсом.

Взаимодействие между браузером и сервером осуществляется по протоколу HTTP. Валидация данных включает проверку наличия обязательных полей, ограничения по длине текста и другие правила валидации, определенные для каждого поля курса.

## 2.2.3 Система прохождения теста

Ещё одной из ключевых возможностей, данного сервиса, является возможность проходить тестирование по изученному материалу. Процесс прохождения теста показан на диаграмме (Рисунок 2.7).

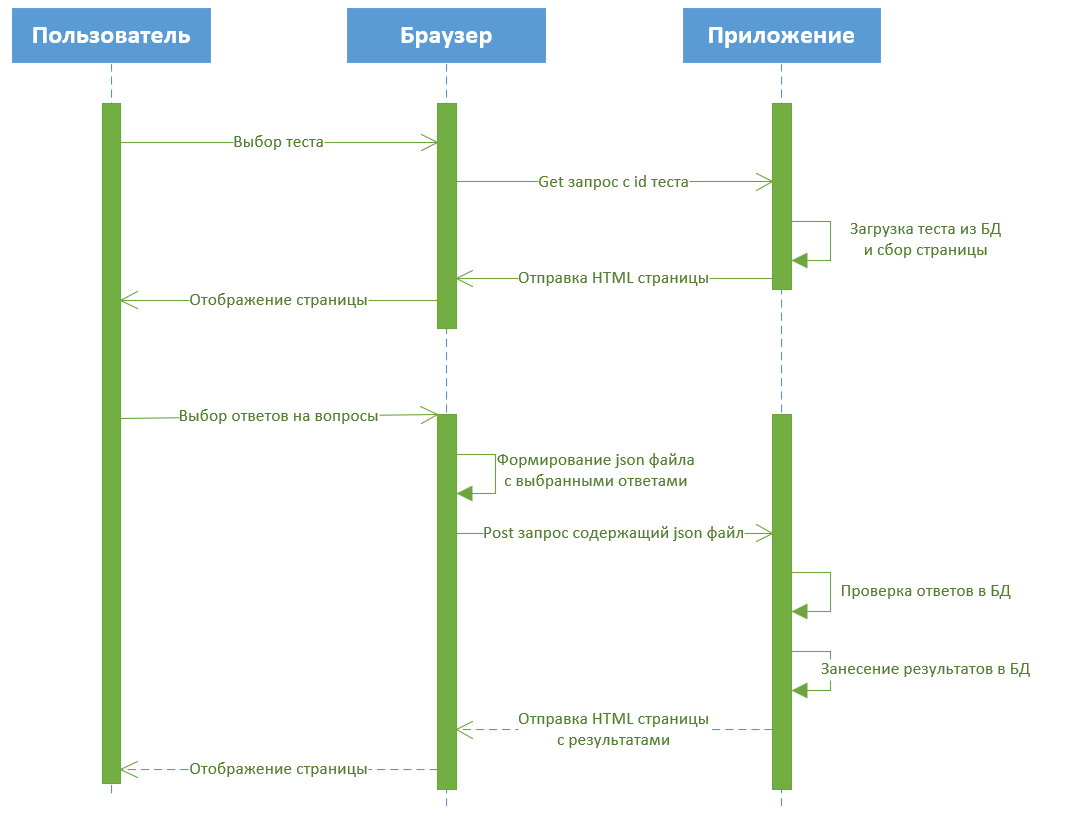


Рисунок 2.6 Диаграмма последовательности прохождение теста

Actors:

1. Пользователь: Человек, который проходит тест;
2. браузер: Интернет-браузер, через который пользователь взаимодействует с веб-приложением;
3. сервер: Веб-сервер, на котором расположено ASP.NET приложение.

Шаги и взаимодействия:

1. Пользователь выбирает тест, который хочет пройти;
2. браузер отправляет запрос на сервер с информацией о выбранном тесте;
3. сервер загружает вопросы и варианты ответов из базы данных, связанные с выбранным тестом;
4. сервер формирует страницу с вопросом и вариантами ответов и отправляет ее браузеру;
5. браузер отображает страницу с вопросами и вариантами ответов;
6. пользователь выбирает один или несколько ответов;
7. браузер формирует json файл содержащий вопросы и ответы выбранные пользователем;
8. браузер отправляет post запрос на сервер с файлом;
9. сервер формирует страницу с результатами теста и отправляет ее браузеру;
10. браузер отображает страницу с результатами теста пользователю.

## 2.2.4 Диаграмма деятельности создания урока

В процессе создания урока пользователь выполняет ряд действий, таких как определение названия урока, выбора курса к которому относится урок. Диаграмма также отражает решение ветвления, позволяя пользователю вернуться к определенным действиям, если требуется внесение изменений или исправление ошибок.

Ниже приведено описание элементов, которые представлены на диаграмме:

1. Пользователь определяет для какого курса этот урок;
2. пользователь определяет название урока, который будет создан;
3. пользователь добавляет описание урока
4. пользователь просматривает и проверяет созданный урок перед его публикацией или сохранением;
5. пользователь отправляет данные в приложение;
6. если урок содержит ошибки или требуется внесение изменений, сервер формирует сообщение об ошибке и отправляет его пользователю;
7. пользователь может внести корректировки в урок;
8. если ошибок нет урок сохраняется в БД;
9. Пользователю отправляется сообщение об успешном создание урока.

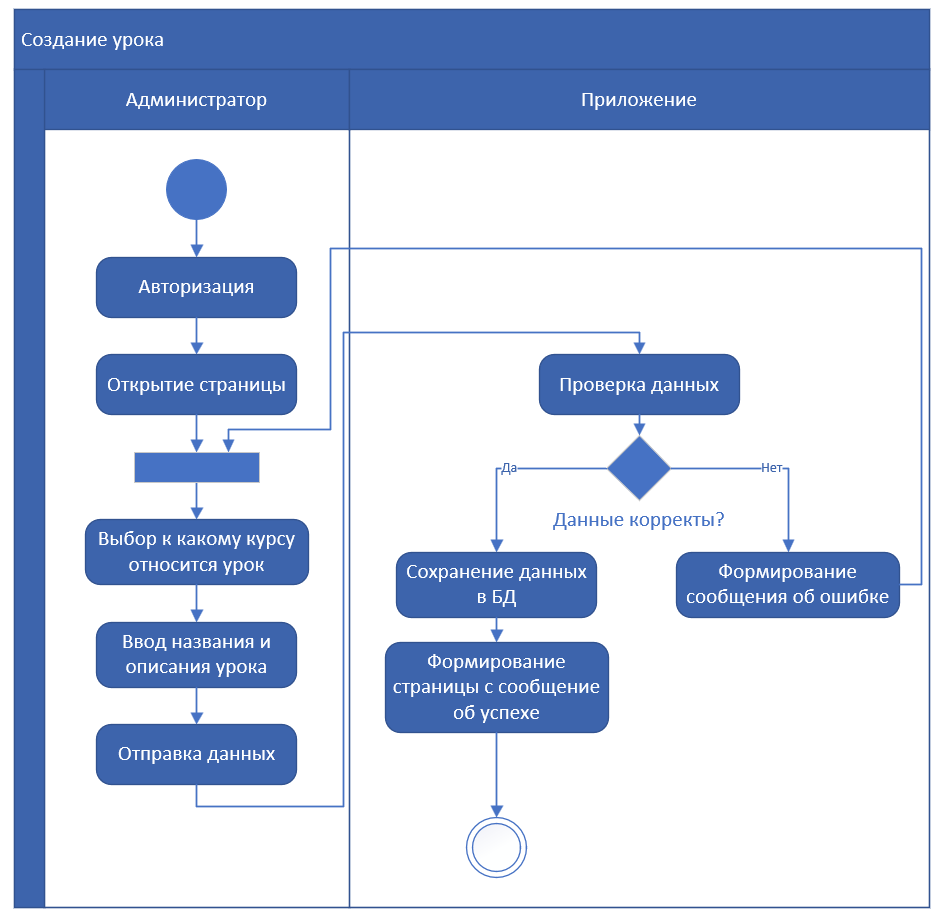


Рисунок 2.7 создание урока

Процесс создания урока представлен на рисунке 2.7.

## 2.2.5 Диаграмма деятельности создания теста

Данная диаграмма предоставляет наглядное представление последовательности действий и взаимодействий, необходимых для создания теста.

В процессе создания теста пользователь выполняет ряд действий, таких как определение названия теста, добавление вопросов и ответов, установка правильных ответов для прохождения теста. Диаграмма деятельности также отражает возможность добавления новых вопросов сохранение и публикацию теста. Диаграмма изображена на рисунке 2.8

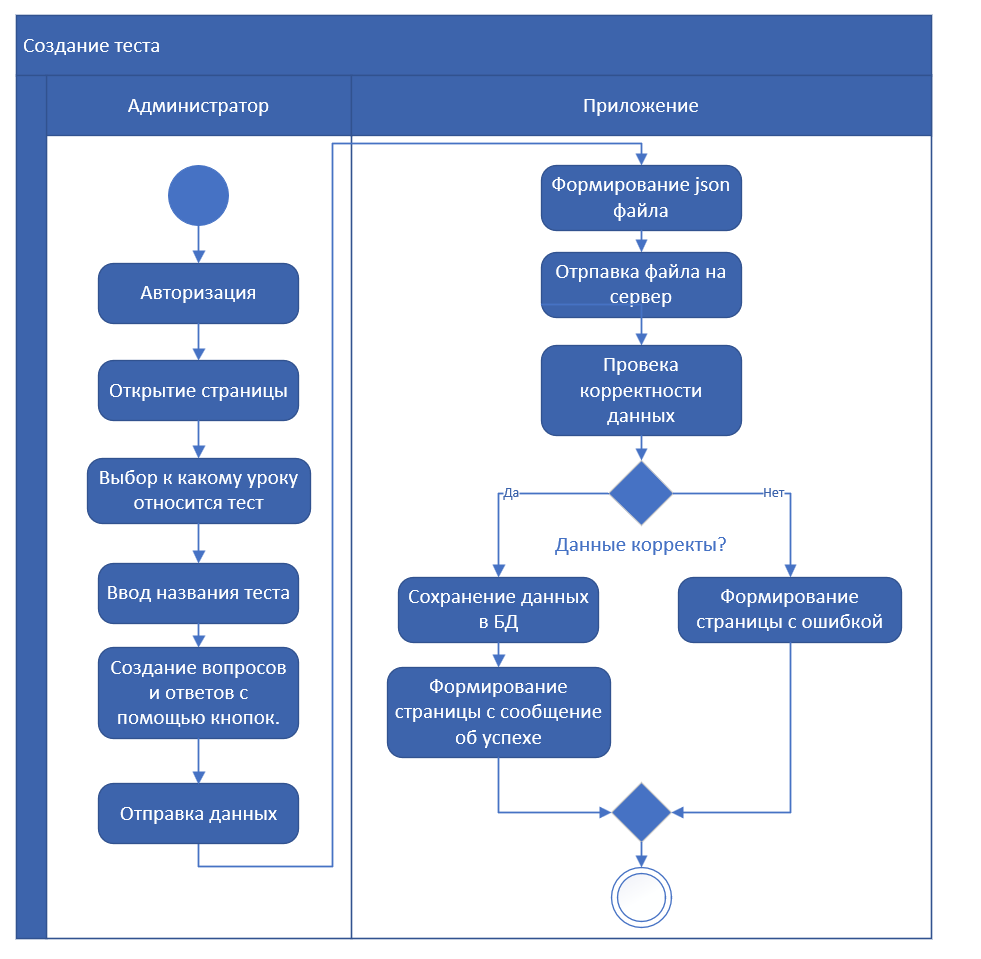


Рисунок 2.8 создание теста

Пользователь начинает процесс создания теста следующим образом:

1. пользователь входит в систему, используя свои учетные данные для авторизации;
2. пользователь выбирает урок, к которому относится создаваемый тест.
3. пользователь вводит название теста;
4. пользователь нажимает кнопку "Добавить вопрос" и вводит вопрос, а затем добавляет связанные с ним варианты ответов, указывая правильный ответ;
5. браузер собирает информацию о вопросах и ответах и формирует JSON-файл, содержащий данные теста;
6. браузер загружает JSON-файл, отправляя его на сервер;
7. сервер получает загруженный файл и производит проверку на корректность данных, включая формат JSON-файла, наличие всех необходимых полей и правильность указания правильных ответов;
8. если данные проходят проверку, сервер сохраняет информацию о созданном тесте в базе данных, связывая его с выбранным уроком;
9. если данные не проходят проверку, сервер формирует страницу с сообщением об ошибке, указывая конкретные проблемы, которые необходимо исправить;
10. если данные успешно сохранены, сервер формирует страницу с сообщением об успешном создании теста, подтверждая его добавление в систему.

## 2.2.6 Диаграмма деятельности: регистрация пользователя

Данная диаграмма (рисунок 2.9) иллюстрирует последовательность действий, которые необходимо выполнить для успешной регистрации пользователя в системе. Она включает в себя взаимодействие между пользователем и системой, а также внутренние операции, выполняемые системой для обработки регистрационных данных.

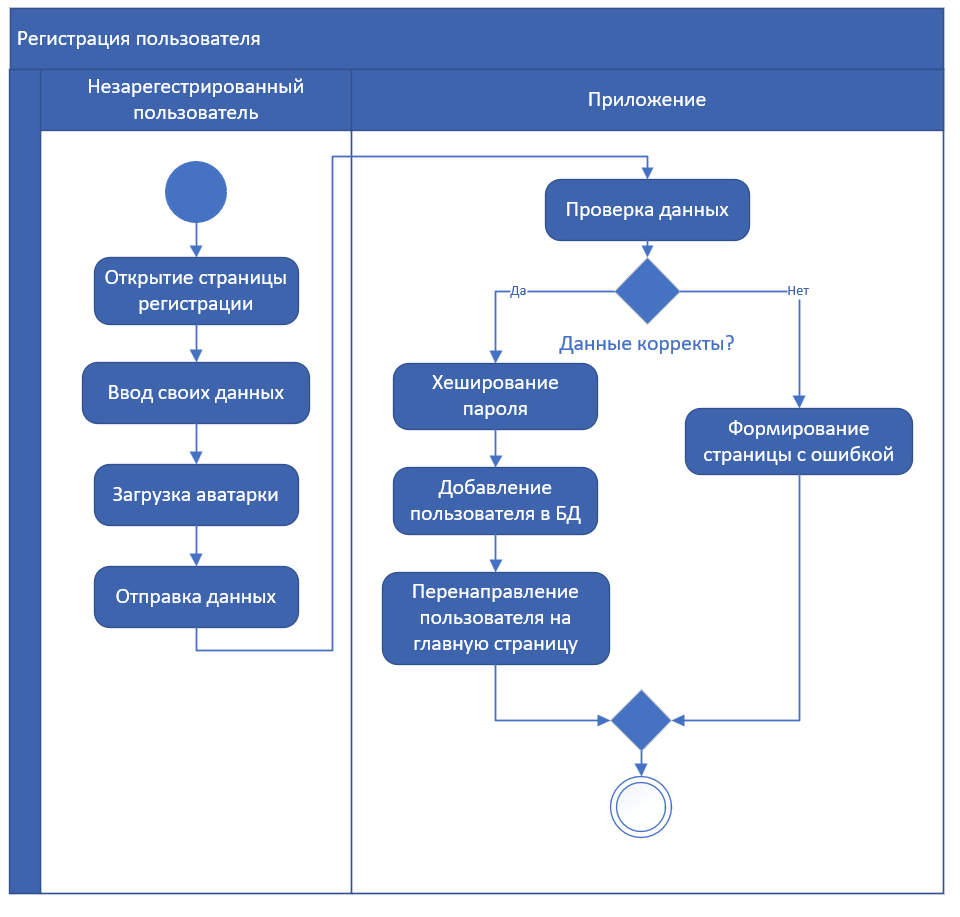


Рисунок 2.9 Регистрация пользователя

Шаги для регистрации пользователя:

1. Пользователь открывает веб-браузер и переходит на страницу регистрации;
2. система отображает страницу регистрации с необходимыми полями для ввода информации, такими как имя, адрес электронной почты и пароль;
3. пользователь вводит свои данные в соответствующие поля на странице регистрации;
4. после заполнения полей, пользователь нажимает кнопку "Зарегистрироваться" для отправки данных на сервер;
5. система получает данные, введенные пользователем, и проводит их валидацию. Валидация включает проверку обязательных полей, правильности формата адреса электронной почты и пароля, а также проверку уникальности адреса электронной почты в системе;
6. если данные проходят валидацию, система создает новую запись пользователя в базе данных, сохраняя информацию, включая хешированный пароль для обеспечения безопасности;
7. после успешного создания записи пользователя, система отображает страницу с подтверждением успешной регистрации и предоставляет пользователю возможность войти в систему;
8. в случае возникновения ошибок в процессе регистрации (например, неверный формат электронной почты или неправильный пароль), система формирует страницу с сообщением об ошибке, указывая конкретные проблемы, которые необходимо исправить.

## 2.2.7 Проектирование вариантов использования

Для проектирования вариантов использования был выбран инструмент Microsoft visio.

Не зарегистрированный пользователь может выполнять следующие действия:

* просмотр основной страницы сайта;
* просмотр информации о курсе;
* просмотр статей;
* возможность зарегистрироваться.

Зарегистрированный пользователь, к дополнению возможностей незарегистрированного пользователя, имеет следующие возможности:

* аутентификация, которая включает в себя ввод логина и пароля;
* выполнять действия с акантом, изменять свои данные;
* работать с курсом: проходить тесты, изучать теорию.

Администрация в дополнение ко всему предыдущему, может выполнять следующие действия:

* редактировать курс, что в себя включает: добавление курсов, добавление уроков, создание лекций, создание тестов;
* работа с пользователями: создание и удаление.

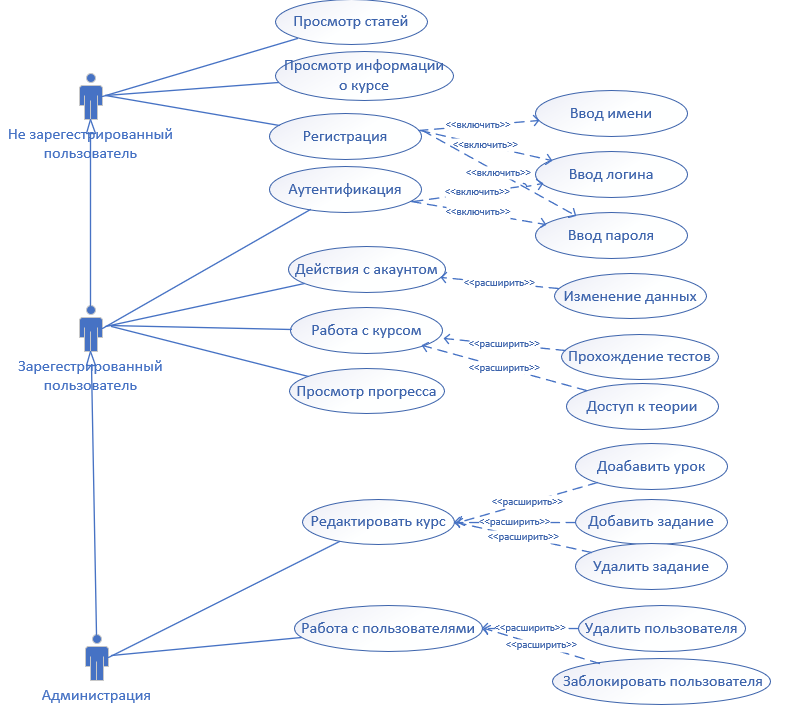


Рисунок 2.10 варианты использования

Приведенная диаграмма (рисунке 2.10) показывает взаимодействие актёров с прецедентами в рамках проекта

# 3 Проектирование программных модулей

## 3.1 Уровень Domain

На этом слое находятся классы моделей, перечислений, ответов от сервера, и вью модели. Далее представлен код и описание модели пользователя:

public class User

{

public int Id { get; set; }

[Required]

[MaxLength(30)]

public string Name { get; set; }

[Required]

[MaxLength(30)]

public string SecondName { get; set; }

[Required]

public DateTime? BirthDay { get; set; }

[Required]

[MaxLength(30)]

public string Login { get; set; }

[Required]

[MaxLength(30)]

public string Email { get; set; }

[Required]

[MaxLength(300)]

public string Password { get; set; }

[MaxLength(400)]

public string AboutMe { get; set; }

public byte[] avatar { get; set; }

[Required]

public Roles Role { get; set; }

[ValidateNever]

public List<Solution> Solutions { get; set; }

}

* Id: поле типа int, представляющее уникальный идентификатор пользователя.
* Name: поле типа string, представляющее имя пользователя. Это поле обязательно для заполнения и имеет максимальную длину в 30 символов.
* SecondName: поле типа string, представляющее фамилию пользователя. Это поле обязательно для заполнения и имеет максимальную длину в 30 символов.
* BirthDay: поле типа DateTime?, представляющее дату рождения пользователя. Это поле обязательно для заполнения.
* Login: поле типа string, представляющее логин пользователя. Это поле обязательно для заполнения и имеет максимальную длину в 30 символов.
* Email: поле типа string, представляющее электронную почту пользователя. Это поле обязательно для заполнения и имеет максимальную длину в 30 символов.
* Password: поле типа string, представляющее пароль пользователя. Это поле обязательно для заполнения и имеет максимальную длину в 300 символов.
* AboutMe: поле типа string, представляющее информацию о пользователе. Это поле имеет максимальную длину в 400 символов.
* avatar: поле типа byte[], представляющее изображение аватара пользователя.
* Role: поле типа Roles, представляющее роль пользователя. Это поле обязательно для заполнения.
* Solutions: поле типа List<Solution>, представляющее список решений, связанных с пользователем.

Класс модели "User" содержит свойства, которые описывают основную информацию о пользователе, такую как имя, фамилия, дата рождения, логин, электронная почта, пароль, информация о пользователе, аватар и роль. Кроме того, у пользователя может быть связь с решениями, которые он предоставляет в системе.

Также на этом уровне имеются различные перечисления: статус код, роли, тип элемента. Ниже представлен один из элементов перечисления (статус код):

public enum StatusCode

{

//users Codes 0-20

[Display(Name = "Пользователь не найден")]

UserNotFound = 0,

[Display(Name = "Пользователь уже существует")]

UserAlreaydyExists = 1,

//common codes

[Display(Name = "Успех")]

OK = 200,

[Display(Name = "Предупреждение")]

Warn = 201,

[Display(Name = "Ошибка запроса")]

BadRequest = 400,

[Display(Name = "Страница не найдена")]

NotFound = 404,

[Display(Name = "Ошибка на стороне сервера")]

InternalServerError = 500

}

Перечисление представляет различные статусы, которые могут быть возвращены в ответ на запрос в системе.

* UserNotFound: Значение 0. Пользователь не найден.
* UserAlreadyExists: Значение 1. Пользователь уже существует.

Далее перечислены общие статусы:

* OK: Значение 200. Успешный статус. Запрос выполнен успешно.
* Warn: Значение 201. Предупреждение. Запрос выполнен успешно, но с некоторыми замечаниями или предупреждениями.
* BadRequest: Значение 400. Ошибка запроса. Запрос некорректный или неполный.
* NotFound: Значение 404. Страница не найдена. Запрашиваемый ресурс или страница не существует.
* InternalServerError: Значение 500. Ошибка на стороне сервера. Внутренняя ошибка сервера при обработке запроса.

Перечисление используется для определения статусов ответов на запросы в системе и может быть использовано для обработки различных ситуаций при взаимодействии с API или другими частями приложения. Каждое значение имеет ассоциированное отображаемое имя, которое может быть использовано для отображения пользователю или для других целей визуализации.

Ещё одним элементом данного уровня, являются шаблоны ответов серверу от службы:

public class BaseResponse<T> : IBaseResponse<T>

{

public string Description { get; set; }

public StatusCode StatusCode { get; set; }

public T Data { get; set; }

}

Класс BaseResponse<T> представляет общий формат ответа, который может быть возвращен в системе. Он реализует интерфейс IBaseResponse<T>.

Вот описание его свойств:

* Description: Описание ответа. Это строковое свойство, которое содержит описание результата или дополнительную информацию об ответе.
* StatusCode: Статус код. Это свойство типа StatusCode (перечисление), которое указывает на статус ответа, описывая его с помощью одного из значений из перечисления StatusCode.
* Data: Данные ответа. Это свойство типа T, которое представляет собой обобщенный тип данных и содержит конкретные данные, связанные с ответом. Тип T будет определен в момент использования класса BaseResponse<T>, в зависимости от контекста и требуемого типа данных для ответа.

Класс BaseResponse<T> позволяет упаковывать результаты и данные ответов в стандартный формат, включающий описание, статус код и соответствующие данные. Это удобно для передачи и обработки ответов в системе, а также для стандартизации формата ответов в рамках проекта.

Последней частью данного уровня являются вью модели, которые служат для представления данных в нужном формате для различных мест.

public class CompleteArticle

{

public int TitleId { get; set; }

public int ArticleId { get; set; }

[Required]

[StringLength(20)]

public string Name { get; set; }

[Required]

[StringLength(300)]

public string ShortDescription { get; set; }

[Required]

[ValidateNever]

public string ArticleText { get; set; }

[Required]

[ValidateNever]

public DateTime CreatedDate { get; set; }

[Required]

[ValidateNever]

public PythonLearn.Domain.Entity.User user { get; set; }

[Required]

public int UserId { get; set; }

}

Вью-модель CompleteArticle представляет данные, необходимые для полного отображения статьи. Вот описание его свойств:

* TitleId: Идентификатор заголовка статьи. Это целочисленное свойство, которое содержит идентификатор заголовка, связанного с данной статьей.
* ArticleId: Идентификатор статьи. Это целочисленное свойство, которое содержит идентификатор статьи.
* Name: Название статьи. Это строковое свойство, которое содержит название статьи. Оно обязательно для заполнения и должно быть не длиннее 20 символов.
* ShortDescription: Краткое описание статьи. Это строковое свойство, которое содержит краткое описание статьи. Оно обязательно для заполнения и должно быть не длиннее 300 символов.
* ArticleText: Текст статьи. Это строковое свойство, которое содержит полный текст статьи. Оно помечено атрибутом ValidateNever, что означает, что его значение не будет проверяться на валидность.
* CreatedDate: Дата создания статьи. Это свойство типа DateTime, которое указывает на дату создания статьи. Оно обязательно для заполнения.
* user: Пользователь, связанный со статьей. Это свойство типа User из пространства имен PythonLearn.Domain.Entity. Оно представляет собой связь со связанной сущностью пользователя.
* UserId: Идентификатор пользователя. Это целочисленное свойство, которое содержит идентификатор пользователя, связанного со статьей.

Вью-модель CompleteArticle используется для передачи данных о статье и связанном пользователе между контроллером и представлением, обеспечивая полное её отображение с основными атрибутами и связанными данными пользователя.

## 3.2 Уровень DAL

DAL (Data Access Layer) представляет собой уровень доступа к данным и взаимодействия с базой данных. В этом слое находятся компоненты, отвечающие за выполнение операций с базой данных, такие как Entity Framework и модели данных. DAL обеспечивает абстракцию базы данных и предоставляет возможность взаимодействия с данными через объектно-ориентированный подход.

На данном уровне располагаются: классы репозитории, миграции, классы для маппинга моделей и служебные классы.

Ниже представлен один из классов репозиториев:

public class TestRepository : ITestRepository

{

private readonly ApplicationDbContext \_context;

public TestRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(Test entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var test = await \_context.Tests.FirstOrDefaultAsync(x => x.Id == id);

if (test == null)

{

throw new ArgumentNullException("[TestRepository] DeleteAsync(int id): the test isn't found");

}

\_context.Tests.Remove(test);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Test entity)

{

var test = await \_context.Tests.FirstOrDefaultAsync(x => x.Id == entity.Id);

if (test == null)

{

throw new ArgumentNullException("[TestRepository] UpdateAsync(int id): the test isn't found");

}

if (entity.Questions != null)

{

test.Questions = entity.Questions;

}

await \_context.SaveChangesAsync();

}

public async Task<Test> GetAsync(int id)

{

var result = await \_context.Tests.Include(c=>c.Questions).FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

throw new ArgumentNullException("[TestRepository] GetAsync(int id): the test isn't found");

}

else

{

return result;

}

}

public IQueryable<Test> GetAllAsync()

{

var result = \_context.Tests.Include(c=>c.Questions);

if (result == null)

{

throw new ArgumentNullException("[TestRepository] GetAllAsync(): the tests aren't found");

}

else

{

return result;

}

}

}

Класс TestRepository представляет репозиторий для работы с сущностями типа Test. Вот описание его методов и свойств:

* private readonly ApplicationDbContext \_context: Приватное только для чтения поле \_context представляет контекст базы данных ApplicationDbContext, который используется для взаимодействия с базой данных.
* public TestRepository(ApplicationDbContext context): Конструктор класса, который принимает объект контекста базы данных и инициализирует поле \_context.
* public async Task CreateAsync(Test entity): Метод для создания новой сущности Test. Принимает объект entity в качестве параметра, добавляет его в контекст базы данных с помощью метода AddAsync, а затем сохраняет изменения с помощью метода SaveChangesAsync.
* public async Task DeleteAsync(int id): Метод для удаления сущности Test по заданному идентификатору id. Ищет сущность с указанным идентификатором в контексте базы данных с помощью метода FirstOrDefaultAsync, а затем удаляет ее из контекста и сохраняет изменения с помощью метода SaveChangesAsync. Если сущность не найдена, генерируется исключение ArgumentNullException.
* public async Task UpdateAsync(Test entity): Метод для обновления сущности Test. Ищет сущность с указанным идентификатором в контексте базы данных с помощью метода FirstOrDefaultAsync. Если сущность найдена, обновляет ее свойства на основе переданного объекта entity. Затем сохраняет изменения с помощью метода SaveChangesAsync. Если сущность не найдена, генерируется исключение ArgumentNullException.
* public async Task<Test> GetAsync(int id): Метод для получения сущности Test по заданному идентификатору id. Ищет сущность с указанным идентификатором в контексте базы данных с помощью метода FirstOrDefaultAsync. Если сущность не найдена, генерируется исключение ArgumentNullException. Если сущность найдена, возвращает ее.
* public IQueryable<Test> GetAllAsync(): Метод для получения всех сущностей Test из базы данных. Возвращает объект IQueryable<Test>, который представляет набор всех сущностей Test в базе данных. Если набор сущностей пуст, генерируется исключение ArgumentNullException.

Класс TestRepository предоставляет основные операции создания, удаления, обновления и получения сущностей Test в базе данных с использованием контекста ApplicationDbContext.

Также на этом уровне располагается класс для доступа к базе данных через entity framework

public class ApplicationDbContext: DbContext

{

public ApplicationDbContext()

{

}

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)

{

Database.EnsureCreated();

}

public DbSet<Answer> Answers { get; set; }

public DbSet<Article> Articles { get; set; }

public DbSet<ArticleComment> ArticleComments { get; set; }

public DbSet<Course> Courses { get; set; }

public DbSet<Lecture> Lectures { get; set; }

public DbSet<Lesson> Lessons { get; set; }

public DbSet<LessonComment> LessonComments { get; set; }

public DbSet<Practice> Practices { get; set; }

public DbSet<Solution> Solutions { get; set; }

public DbSet<Title> Titles{ get; set; }

public DbSet<Test> Tests { get; set; }

public DbSet<Question> Questions { get; set; }

public DbSet<User> Users { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.ApplyConfiguration(new UserConfiguration());

}

}

Класс ApplicationDbContext является производным от класса DbContext и представляет контекст базы данных для взаимодействия с таблицами и сущностями в рамках приложения. Вот описание его свойств, методов и конструкторов:

* public ApplicationDbContext(): Пустой конструктор класса ApplicationDbContext.
* public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options): Конструктор класса ApplicationDbContext, который принимает объект DbContextOptions<ApplicationDbContext> в качестве параметра. Он вызывает базовый конструктор класса DbContext с переданными опциями и также вызывает метод EnsureCreated(), который создает базу данных, если она еще не создана.
* public DbSet<Answer> Answers { get; set; }: Свойство Answers представляет набор сущностей типа Answer в базе данных.
* public DbSet<Article> Articles { get; set; }: Свойство Articles представляет набор сущностей типа Article в базе данных.
* public DbSet<ArticleComment> ArticleComments { get; set; }: Свойство ArticleComments представляет набор сущностей типа ArticleComment в базе данных.
* public DbSet<Course> Courses { get; set; }: Свойство Courses представляет набор сущностей типа Course в базе данных.
* public DbSet<Lecture> Lectures { get; set; }: Свойство Lectures представляет набор сущностей типа Lecture в базе данных.
* public DbSet<Lesson> Lessons { get; set; }: Свойство Lessons представляет набор сущностей типа Lesson в базе данных.
* public DbSet<LessonComment> LessonComments { get; set; }: Свойство LessonComments представляет набор сущностей типа LessonComment в базе данных.
* public DbSet<Practice> Practices { get; set; }: Свойство Practices представляет набор сущностей типа Practice в базе данных.
* public DbSet<Solution> Solutions { get; set; }: Свойство Solutions представляет набор сущностей типа Solution в базе данных.
* public DbSet<Title> Titles { get; set; }: Свойство Titles представляет набор сущностей типа Title в базе данных.
* public DbSet<Test> Tests { get; set; }: Свойство Tests представляет набор сущностей типа Test в базе данных.
* public DbSet<Question> Questions { get; set; }: Свойство Questions представляет набор сущностей типа Question в базе данных.
* public DbSet<User> Users { get; set; }: Свойство Users представляет набор сущностей типа User в базе данных.
* protected override void OnModelCreating(ModelBuilder modelBuilder): Переопределенный метод OnModelCreating, который принимает объект ModelBuilder в качестве параметра. В этом методе можно настраивать модель данных с помощью Fluent API. В данном случае, он применяет конфигурацию UserConfiguration к модели данных с помощью метода ApplyConfiguration(new UserConfiguration()).

Класс ApplicationDbContext является основным классом для работы с базой данных в приложении и предоставляет доступ к различным таблицам и сущностям через свойства типа DbSet<T>.

## 3.3 Уровень Service

Уровень сервиса (Service) в архитектуре приложения является промежуточным уровнем между уровнем представления (Presentation) и уровнем доступа к данным (Data Access). Задачей уровня сервиса является предоставление бизнес-логики и функциональности, связанной с обработкой данных и выполнением операций.

Назначение уровня сервиса в классе ArticleService заключается в предоставлении методов для управления статьями и заголовками. Он обеспечивает основную функциональность, связанную с созданием, чтением, обновлением и удалением (CRUD) статей и заголовков.

public class ArticleService : IArticleService

{

private readonly IUnitOfWork \_context;

/// <summary>

/// Конструктор для dependence injactive

/// </summary>

/// <param name="context">Объект реализующий интерфейс IUnitOfWork</param>

public ArticleService(IUnitOfWork context)

{

\_context = context;

}

//CREATE

/// <summary>

/// Создание статьи

/// </summary>

/// <param name="article">Статья</param>

/// <returns>true в случае успешного создания</returns>

public async Task<IBaseResponse<bool>> CreateArticle(Article article)

{

var response = new BaseResponse<bool>();

try

{

var newArticle = new Article()

{

TitleId = article.TitleId,

ArticleText = article.ArticleText,

UserId = article.UserId

};

await \_context.ArticleRepositories.CreateAsync(newArticle);

response.Description = $"Статья успешно создана";

response.StatusCode = StatusCode.OK;

response.Data = true;

return response;

}

catch (Exception ex)

{

return new BaseResponse<bool>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] CreateArticle(Article article): {ex.Message}"

};

}

}

/// <summary>

/// Создание заголовка

/// </summary>

/// <param name="title">Заголовок</param>

/// <returns>true в случае успешного создания</returns>

public async Task<IBaseResponse<bool>> CreateTitle(Title title)

{

var response = new BaseResponse<bool>();

try

{

var newTitle = new Title()

{

Name = title.Name,

ShortDescription = title.ShortDescription

};

await \_context.TitleRepositories.CreateAsync(newTitle);

response.Description = $"Заголовок успешно создана";

response.StatusCode = StatusCode.OK;

response.Data = true;

return response;

}

catch (Exception ex)

{

return new BaseResponse<bool>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] CreateTitle(Title title): {ex.Message}"

};

}

}

public async Task<IBaseResponse<bool>> CreateArticleWithTitle(CompleteArticle article)

{

var response = new BaseResponse<bool>();

try

{

var newTitle = new Title()

{

Name = article.Name,

ShortDescription = article.ShortDescription

};

await \_context.TitleRepositories.CreateAsync(newTitle);

article.TitleId = \_context.TitleRepositories.GetAllAsync().FirstOrDefault(x => x.Name == article.Name).Id;

var newArticle = new Article()

{

TitleId = article.TitleId,

ArticleText = article.ArticleText,

UserId = article.UserId,

CreatedDate = DateTime.Now,

};

await \_context.ArticleRepositories.CreateAsync(newArticle);

response.Description = $"Статья успешно создана";

response.StatusCode = StatusCode.OK;

response.Data = true;

return response;

}

catch (Exception ex)

{

return new BaseResponse<bool>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] CreateArticleWithTitle(CompleteArticle article): {ex.Message}"

};

}

}

//READ

/// <summary>

/// Возвращает статью по id

/// </summary>

/// <param name="id">id статьи</param>

/// <returns>Статью</returns>

public async Task<IBaseResponse<Article>> GetArticleAsync(int id)

{

var response = new BaseResponse<Article>();

try

{

var article = await \_context.ArticleRepositories.GetAsync(id);

if (article == null)

{

response.StatusCode = StatusCode.NotFound;

response.Description = $"Статья с id {id} не найден в БД";

return response;

}

else

{

response.StatusCode = StatusCode.OK;

response.Description = $"Статья {article.Title.Name} с id {id} найдена";

response.Data = article;

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<Article>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] GetArticleAsync(int id): {ex.Message}"

};

}

}

/// <summary>

/// Получить полную статью с заголовком и текстом статьи

/// </summary>

/// <param name="id"> Id статьи </param>

/// <returns></returns>

public async Task<IBaseResponse<CompleteArticle>> GetCompleteArticle(int id)

{

var response = new BaseResponse<CompleteArticle>();

try

{

var article = await \_context.ArticleRepositories.GetAsync(id);

if (article == null)

{

response.StatusCode = StatusCode.NotFound;

response.Description = $"Статья с id {id} не найден в БД";

return response;

}

else

{

var title = await \_context.TitleRepositories.GetAsync(article.TitleId);

var completeArticle = new CompleteArticle()

{

TitleId = title.Id,

ArticleId = article.Id,

Name = title.Name,

ShortDescription = title.ShortDescription,

ArticleText = article.ArticleText,

CreatedDate = article.CreatedDate,

UserId = article.UserId,

user = article.User

};

response.StatusCode = StatusCode.OK;

response.Description = $"Статья {article.Title.Name} с id {id} найдена";

response.Data = completeArticle;

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<CompleteArticle>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] GetCompleteArticle(int id): {ex.Message}"

};

}

}

/// <summary>

/// Получить все статьи

/// </summary>

/// <returns>Список статей</returns>

public async Task<IBaseResponse<IEnumerable<Article>>> GetArticlesAsync()

{

var baseResponse = new BaseResponse<IEnumerable<Article>>();

try

{

var articles = await \_context.ArticleRepositories.GetAllAsync().ToListAsync();

if (articles.Count == 0)

{

baseResponse.StatusCode = StatusCode.Warn;

baseResponse.Description = $"Найдено 0 элементов";

return baseResponse;

}

else

{

baseResponse.StatusCode = StatusCode.OK;

baseResponse.Description = $"Найдено {articles.Count} статей";

baseResponse.Data = articles;

return baseResponse;

}

}

catch (Exception ex)

{

return new BaseResponse<IEnumerable<Article>>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] GetArticlesAsync(): {ex.Message}"

};

}

}

/// <summary>

/// Получить заголовок по его id

/// </summary>

/// <param name="id">id заголовка</param>

/// <returns>Заголовок</returns>

public async Task<IBaseResponse<Title>> GetTitleAsync(int id)

{

var response = new BaseResponse<Title>();

try

{

var title = await \_context.TitleRepositories.GetAsync(id);

if (title == null)

{

response.StatusCode = StatusCode.NotFound;

response.Description = $"Заголовок с id {id} не найден в БД";

return response;

}

else

{

response.StatusCode = StatusCode.OK;

response.Description = $"Заголовок {title.Name} с id {id} найден";

response.Data = title;

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<Title>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] GetTitleAsync(int id): {ex.Message}"

};

}

}

/// <summary>

/// Получить список всех заголовков

/// </summary>

/// <returns>Список заголовков</returns>

public async Task<IBaseResponse<IEnumerable<Title>>> GetTitlesAsync()

{

var baseResponse = new BaseResponse<IEnumerable<Title>>();

try

{

var titles = await \_context.TitleRepositories.GetAllAsync().ToListAsync();

if (titles.Count == 0)

{

baseResponse.StatusCode = StatusCode.Warn;

baseResponse.Description = $"Найдено 0 элементов";

return baseResponse;

}

else

{

baseResponse.StatusCode = StatusCode.OK;

baseResponse.Description = $"Найдено {titles.Count} заголовков";

baseResponse.Data = titles;

return baseResponse;

}

}

catch (Exception ex)

{

return new BaseResponse<IEnumerable<Title>>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] GetTitlesAsync(): {ex.Message}"

};

}

}

/// <summary>

/// Изменить статью

/// </summary>

/// <param name="id">id статьи</param>

/// <param name="article">новая статья</param>

public async Task<IBaseResponse<Article>> EditArticleAsync(int id, Article newArticle)

{

var response = new BaseResponse<Article>();

try

{

var article = await \_context.ArticleRepositories.GetAsync(id);

if (article == null)

{

response.StatusCode = StatusCode.NotFound;

response.Description = $"статья с id {id} не найден в БД";

return response;

}

else

{

if (newArticle.ArticleText != null)

{

article.ArticleText = article.ArticleText;

}

if (newArticle.TitleId != 0)

{

article.TitleId = newArticle.TitleId;

}

if (newArticle.UserId != 0)

{

article.UserId = newArticle.UserId;

}

await \_context.ArticleRepositories.UpdateAsync(article);

response.StatusCode = StatusCode.OK;

response.Description = $"Статья изменена";

response.Data = article;

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<Article>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] EditArticleAsync(int id, Article newArticle): {ex.Message}"

};

}

}

/// <summary>

/// Обновить заголовок

/// </summary>

/// <param name="id">id заголовка</param>

/// <param name="newTitle">новый заголовок</param>

public async Task<IBaseResponse<Title>> EditTitleAsync(int id, Title newTitle)

{

var response = new BaseResponse<Title>();

try

{

var title = await \_context.TitleRepositories.GetAsync(id);

if (title == null)

{

response.StatusCode = StatusCode.NotFound;

response.Description = $"Заголовок с id {id} не найден в БД";

return response;

}

else

{

if (title.Name != null)

{

title.Name = newTitle.Name;

}

if (title.ShortDescription != null)

{

title.ShortDescription= newTitle.ShortDescription;

}

await \_context.TitleRepositories.UpdateAsync(title);

response.StatusCode = StatusCode.OK;

response.Description = $"Статья изменена";

response.Data = title;

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<Title>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] EditTitleAsync(int id, Title newTitle): {ex.Message}"

};

}

}

//DELETE

/// <summary>

/// Удалить статью по её id

/// </summary>

/// <param name="id"> id статьи</param>

/// <returns>true в случае успеха</returns>

public async Task<IBaseResponse<bool>> DeleteArticleAsync(int id)

{

var response = new BaseResponse<bool>();

try

{

var article = await \_context.ArticleRepositories.GetAsync(id);

if (article == null)

{

response.StatusCode = StatusCode.NotFound;

response.Description = $"Статья с id {id} не найден в БД";

response.Data = false;

return response;

}

else

{

response.StatusCode = StatusCode.OK;

response.Description = $"Статья с именем {article.Title.Name} с id {id} удалена";

response.Data = true;

await \_context.ArticleRepositories.DeleteAsync(id);

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<bool>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] DeleteArticleAsync(int id): {ex.Message}",

Data = false

};

}

}

/// <summary>

/// Удалить заголовок статьи

/// </summary>

/// <param name="id">id заголовка</param>

/// <returns>true в случае успеха</returns>

public async Task<IBaseResponse<bool>> DeleteTitleAsync(int id)

{

var response = new BaseResponse<bool>();

try

{

var title = await \_context.TitleRepositories.GetAsync(id);

if (title == null)

{

response.StatusCode = StatusCode.NotFound;

response.Description = $"Заголовок с id {id} не найден в БД";

response.Data = false;

return response;

}

else

{

response.StatusCode = StatusCode.OK;

response.Description = $"Заголовок {title.Name} с id {id} удален";

response.Data = true;

await \_context.TitleRepositories.DeleteAsync(id);

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<bool>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[ArticleService] DeleteTitleAsync(int id): {ex.Message}",

Data = false

};

}

}

}

Класс ArticleService является службой, предоставляющей операции для работы со статьями и заголовками. Он реализует интерфейс IArticleService и зависит от интерфейса IUnitOfWork для доступа к базе данных.

Конструктор ArticleService принимает объект, реализующий интерфейс IUnitOfWork, в качестве параметра. Это позволяет внедрять зависимость и обеспечивает возможность работы с базой данных.

Класс ArticleService содержит методы для создания, чтения, обновления и удаления статей и заголовков. Каждый метод выполняет соответствующую операцию с использованием объектов Article и Title и вызывает соответствующие методы репозиториев, доступных через объект IUnitOfWork.

Методы CreateArticle, CreateTitle и CreateArticleWithTitle выполняют создание новых статей и заголовков, а методы GetArticleAsync, GetCompleteArticle, GetArticlesAsync, GetTitleAsync и GetTitlesAsync выполняют чтение статей и заголовков из базы данных. Методы EditArticleAsync и EditTitleAsync обновляют информацию о статье и заголовке, а методы DeleteArticleAsync и DeleteTitleAsync удаляют статью и заголовок соответственно.

Каждый метод возвращает объект, реализующий интерфейс IBaseResponse, который содержит информацию о результате операции, статусе операции и, при необходимости, данные, связанные с операцией.

Класс ArticleService отвечает за бизнес-логику, связанную со статьями и заголовками, и обеспечивает абстракцию над доступом к данным через репозитории.

## 3.4 Уровень Controller

Уровень контроллера (Controller) в архитектуре приложения отвечает за обработку входящих запросов от клиентского кода и управление выполнением операций, связанных с бизнес-логикой приложения. Он служит как посредник между пользовательским интерфейсом и уровнем сервиса, принимая входные данные от клиента, вызывая соответствующие методы сервиса и возвращая результаты клиенту.

public class ArticleController : Controller

{

private readonly ILogger<HomeController> \_logger;

private readonly IArticleService \_service;

private readonly IUserService \_userService;

public ArticleController(ILogger<HomeController> logger, IArticleService service, IUserService userService)

{

\_logger = logger;

\_service = service;

\_userService= userService;

}

/\*

public async Task<IActionResult> Show()

{

var article = await \_service.GetArticleAsync(4);

return View(article.Data);

}\*/

[HttpGet]

public async Task<IActionResult> Articles()

{

var articles = await \_service.GetArticlesAsync();

return View(articles.Data);

}

[HttpGet]

public async Task<IActionResult> Article(int id)

{

var fullArticle = await \_service.GetCompleteArticle(id);

return View(fullArticle.Data);

}

[HttpGet]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Create()

{

return View();

}

[HttpPost]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Create(CompleteArticle article)

{

var TxtInfo = HttpContext.Request.Form["content"].ToString();

var user = \_userService.GetUsersAsync().Result.Data.FirstOrDefault(x => x.Login == User.Identity.Name);

ModelState.ClearValidationState(nameof(CompleteArticle));

article.ArticleText = TxtInfo;

article.UserId = user.Id;

var isValid = TryValidateModel(article, nameof(CompleteArticle));

var test = ModelState.IsValid;

if (isValid)

{

var response = await \_service.CreateArticleWithTitle(article);

if (response.StatusCode == Domain.Enum.StatusCode.OK)

{

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError("", response.Description);

}

return View(article);

}

[HttpPost]

public ActionResult UplodImage(List<IFormFile> files, string folder)

{

if (folder == null)

{

folder = Path.GetRandomFileName();

}

string folderPath = Path.Combine("wwwroot", "images/articles/", folder);

if (!Directory.Exists(folderPath))

{

Directory.CreateDirectory(folderPath);

}

var filePath = "";

foreach (IFormFile photo in Request.Form.Files)

{

string serverMapPath = Path.Combine("wwwroot", "images/articles/", folder, photo.FileName);

using (var stream = new FileStream(serverMapPath, FileMode.Create))

{

photo.CopyToAsync(stream);

}

filePath = "https://localhost:7265/images/articles/" + folder +"/" + photo.FileName;

}

return Json(new { url = filePath });

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

Контроллер ArticleController позволяет взаимодействовать с операциями, связанными со статьями, взаимодействуя с соответствующим уровнем сервиса и предоставляя необходимые данные и результаты клиентскому коду.

## 3.5 Уровень view

Уровень View в паттерне проектирования MVC (Model-View-Controller) представляет собой пользовательский интерфейс (UI) и отображает данные модели пользователю. В данном случае, уровень View отвечает за отображение информации пользователю и взаимодействие с ним.

Razor-страницы являются одним из способов реализации уровня View в ASP.NET Core приложениях. Они представляют собой шаблоны, написанные на языке Razor, который является комбинацией HTML и C#. Razor-страницы обладают динамическим характером и позволяют встраивать C#-код в HTML для удобного отображения данных.

Вот некоторые назначения уровня View и Razor-страниц:

Отображение данных: Уровень View и Razor-страницы служат для отображения данных модели пользователю. Они определяют, как данные должны быть представлены на веб-странице. Различные элементы HTML, такие как заголовки, параграфы, таблицы и формы, используются для структурирования и отображения данных.

Взаимодействие с пользователем: Уровень View и Razor-страницы обрабатывают пользовательские действия и события, такие как нажатие кнопок, ввод текста и выбор элементов из списка. Они могут содержать JavaScript-код для обработки клиентской стороны и AJAX-запросов для обновления данных без перезагрузки страницы.

Представление состояния: Уровень View и Razor-страницы отображают текущее состояние модели и позволяют пользователям взаимодействовать с ней. Например, они могут отображать детали статьи, список статей, формы для создания или редактирования статей и т. д.

Поддержка шаблонов: Razor-страницы поддерживают использование шаблонов для повторного использования кода и структуры страниц. Это позволяет сократить дублирование кода и упростить обслуживание и разработку приложения.

Валидация данных: Уровень View и Razor-страницы могут выполнять проверку и валидацию данных, введенных пользователем. Они могут содержать атрибуты и правила валидации, которые позволяют проверять данные на соответствие определенным условиям и выводить сообщения об ошибках при неверных значениях.

В целом, уровень View и Razor-страницы играют важную роль в представлении данных модели и обеспечивают пользовательский интерфейс для взаимодействия с приложением. Они помогают создавать удобный и интуитивно понятный пользовательский опыт.

Пример страницы создания теста:

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<**script** src="~/js/ckeditor5/ckeditor.js"></**script**>

<script src="https://kit.fontawesome.com/c2c8ebeca3.js" crossorigin="anonymous"></script>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet"

integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD" crossorigin="anonymous">

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"

integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN"

crossorigin="anonymous"></script>

<link rel="stylesheet" href="css/style.css">

<link rel="stylesheet" type="text/css" href="js/slick-1.8.1/slick/slick.css" />

<title>Обучение python</title>

</head>

<div class="container mt-4">

<h1>Cоздание тестирования</h1>

<label for="testName">Название теста</label>

<input type="text" class="form-control" id="testName" />

<div class="form-group mt-4">

<label for="lessonSelect">Выберите урок:</label>

<**select** class="form-control" id="lessonSelect" **asp-items**="@ViewBag.Lessons"></**select**>

</div>

<div class="form-group mt-4">

<button id="addSingleChoiceQuestion" class="btn btn-primary">Добавить вопрос с одним вариантом ответа</button>

<button id="addMultipleChoiceQuestion" class="btn btn-primary">Добавить вопрос с несколькими вариантами ответа</button>

<button id="addTextQuestion" class="btn btn-primary">Добавить вопрос с текстовым ответом</button>

</div>

<div id="questionsContainer" class="mt-4"></div>

<button id="saveTest" class="btn btn-success mt-4">Сохранить тест</button>

</div>

<script src="https://cdn.jsdelivr.net/npm/@@popperjs/core@2.11.6/dist/umd/popper.min.js"

integrity="sha384-oBqDVmMz9ATKxIep9tiCxS/Z9fNfEXiDAYTujMAeBAsjFuCZSmKbSSUnQlmh/jp3"

crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.min.js"

integrity="sha384-mQ93GR66B00ZXjt0YO5KlohRA5SY2XofN4zfuZxLkoj1gXtW8ANNCe9d5Y3eG5eD"

crossorigin="anonymous"></script>

<script src="js/jquery-3.6.3.js"></script>

<script type="text/javascript" src="js/slick-1.8.1/slick/slick.min.js"></script>

<script src="js/main.js"></script>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script>

$(document).ready(function () {

var questionCounter = 0;

$('#addSingleChoiceQuestion').click(function () {

questionCounter++;

var questionId = 'question' + questionCounter;

var questionHtml = '<div id="' + questionId + '" class="card mt-4">' +

'<div class="card-body">' +

'<input type="text" class="form-control questionInput" placeholder="Введите вопрос">' +

'<button class="btn btn-primary mt-2 addOption">Добавить вариант ответа</button>' +

'<button class="btn btn-danger mt-2 removeQuestion">Удалить вопрос</button>' +

'<div class="optionsContainer mt-2"></div>' +

'</div>' +

'</div>';

$('#questionsContainer').append(questionHtml);

$('#' + questionId).on('click', '.addOption', function () {

var optionHtml = '<div class="input-group">' +

'<div class="input-group-text">' +

'<input type="radio" name="' + questionId + '" class="optionRadio">' +

'</div>' +

'<input type="text" class="form-control optionInput" placeholder="Введите вариант ответа">' +

'<button class="btn btn-danger removeOption">Удалить вариант ответа</button>' +

'</div>';

$(this).siblings('.optionsContainer').append(optionHtml);

});

$('#' + questionId).on('click', '.removeOption', function () {

$(this).parent().remove();

});

$('#' + questionId).on('click', '.removeQuestion', function () {

$(this).parent().parent().remove();

});

});

$('#addMultipleChoiceQuestion').click(function () {

questionCounter++;

var questionId = 'question' + questionCounter;

var questionHtml = '<div id="' + questionId + '" class="card mt-4">' +

'<div class="card-body">' +

'<input type="text" class="form-control questionInput" placeholder="Введите вопрос">' +

'<button class="btn btn-primary mt-2 addOption">Добавить вариант ответа</button>' +

'<button class="btn btn-danger mt-2 removeQuestion">Удалить вопрос</button>' +

'<div class="optionsContainer mt-2"></div>' +

'</div>' +

'</div>';

$('#questionsContainer').append(questionHtml);

$('#' + questionId).on('click', '.addOption', function () {

var optionHtml = '<div class="input-group">' +

'<div class="input-group-text">' +

'<input type="checkbox" name="' + questionId + '" class="optionCheckbox">' +

'</div>' +

'<input type="text" class="form-control optionInput" placeholder="Введите вариант ответа">' +

'<button class="btn btn-danger removeOption">Удалить вариант ответа</button>' +

'</div>';

$(this).siblings('.optionsContainer').append(optionHtml);

});

$('#' + questionId).on('click', '.removeOption', function () {

$(this).parent().remove();

});

$('#' + questionId).on('click', '.removeQuestion', function () {

$(this).parent().parent().remove();

});

});

$('#addTextQuestion').click(function () {

questionCounter++;

var questionId = 'question' + questionCounter;

var questionHtml = '<div id="' + questionId + '" class="card mt-4">' +

'<div class="card-body">' +

'<input type="text" class="form-control questionInput" placeholder="Введите вопрос">' +

'<button class="btn btn-danger mt-2 removeQuestion">Удалить вопрос</button>' +

'<div class="textAnswerContainer mt-2">' +

'<label for="textAnswer">Текстовый ответ</label>' +

'<input type="text" id="textAnswer" class="form-control" placeholder="Введите ответ">' +

'</div>' +

'</div>' +

'</div>';

$('#questionsContainer').append(questionHtml);

$('#' + questionId).on('click', '.removeQuestion', function () {

$(this).parent().parent().remove();

});

});

$('#saveTest').click(function () {

var test = {

name: $('#testName').val(),

lesson: $('#lessonSelect').val(),

questions: []

};

$('.card').each(function () {

var question = {

text: $(this).find('.questionInput').val(),

options: []

};

var questionId = $(this).attr('id');

var isSingleChoice = $(this).find('.optionRadio').length > 0;

if (isSingleChoice) {

$(this).find('.optionInput').each(function () {

question.options.push({

text: $(this).val(),

isCorrect: $(this).siblings('.input-group-text').find('.optionRadio').is(':checked')

});

});

} else {

$(this).find('.optionInput').each(function () {

question.options.push({

text: $(this).val(),

isCorrect: $(this).siblings('.input-group-text').find('.optionCheckbox').is(':checked')

});

});

}

test.questions.push(question);

});

$('.textAnswerContainer').each(function () {

var question = {

text: $(this).siblings('.questionInput').val(),

answer: $(this).find('#textAnswer').val()

};

test.questions.push(question);

});

var jsonData = JSON.stringify(test);

console.log(jsonData);

$.ajax({

url: "/Course/CreateTest",

type: "POST",

dataType: "json",

contentType: "application/json",

data: jsonData,

success: function (response) {

},

error: function (xhr, status, error) {

}

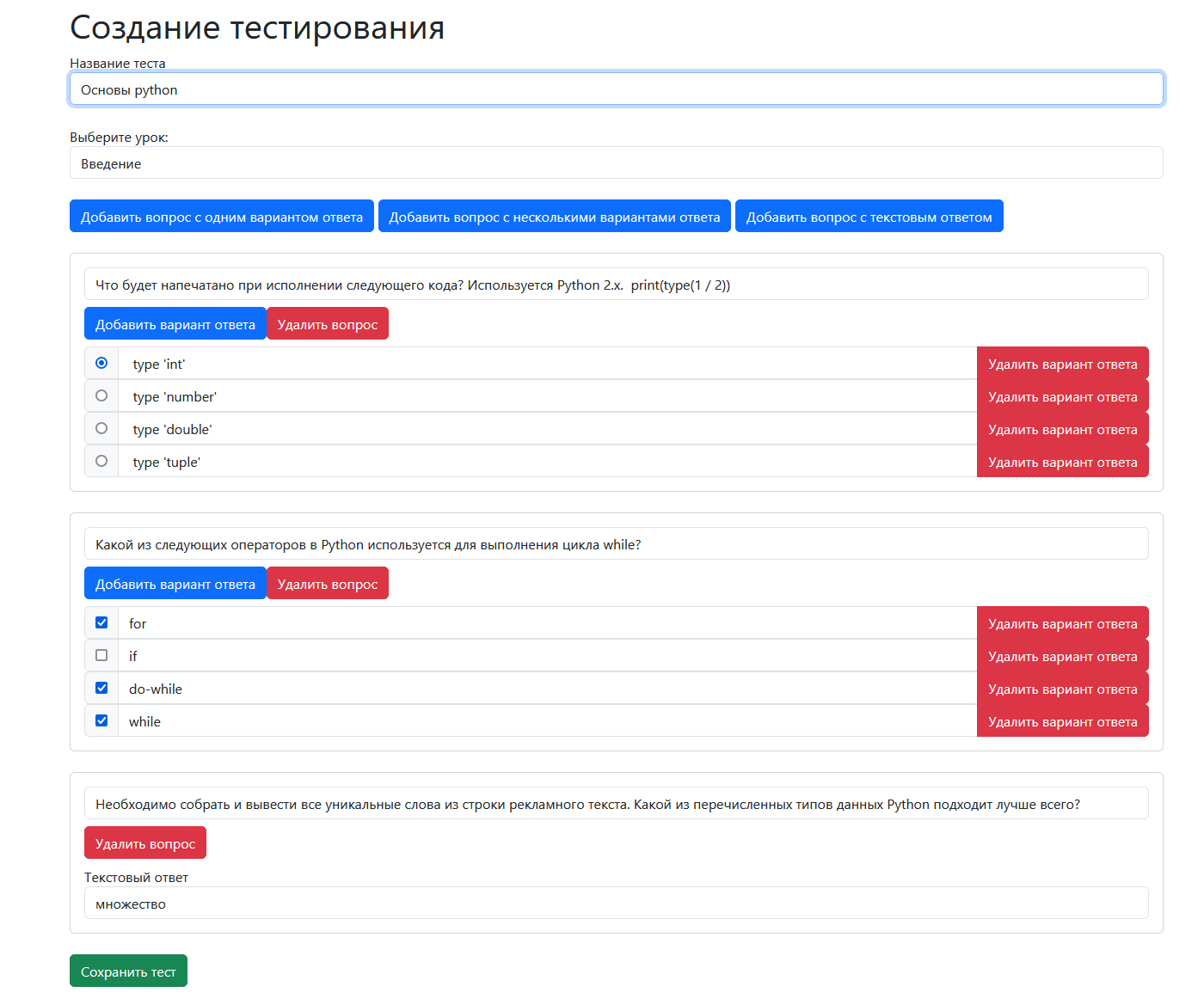
});

});

});

</script>

В результате получается такая страница



# Заключение

В заключение, проект создания сервиса на платформе ASP.NET является важным и актуальным направлением развития сферы онлайн образования. Он предоставляет возможность пользователям изучать язык программирования Python в удобном и доступном формате, объединяя в себе преимущества ASP.NET и Python.

В ходе работы над проектом были рассмотрены основные аспекты разработки образовательных сервисов, включая архитектуру MVC, создание моделей данных, разработку уровней контроллера, сервиса и представления с использованием ASP.NET и Razor-страниц.

Была разработана функциональность, позволяющая пользователям просматривать статьи, создавать новые статьи с возможностью загрузки изображений, а также просматривать полные версии статей. Важным аспектом проекта является поддержка авторизации и ролевой модели, позволяющей ограничить доступ к определенным функциям только для администраторов.

Также была реализована валидация данных, обработка пользовательских действий и взаимодействие с базой данных для сохранения и извлечения информации.

В целом, проект является успешной реализацией идей и концепций, объединяющих две важные платформы - ASP.NET и Python, и предоставляющих пользователям возможность изучать и развиваться в сфере программирования с помощью современных технологий онлайн обучения.

# Список источников информации

1. https://habr.com/ru/company/nix/blog/321686/
2. https://metanit.com/?ysclid=lfw1a917lx563693748
3. https://metanit.com/sharp/tutorial/
4. https://metanit.com/sharp/tutorial/2.8.php
5. https://metanit.com/sharp/tutorial/3.37.php
6. https://learn.microsoft.com/ru-ru/dotnet/csharp/
7. https://learn.microsoft.com/ru-ru/users/dotnet/collections/yz26f8y64n7k07?WT.mc\_id=dotnet-35129-website
8. https://professorweb.ru/my/ASP\_NET/base/level1/base\_aspnet\_index.php
9. https://professorweb.ru/my/ASP\_NET/base/level5/5\_3.php
10. https://professorweb.ru/my/ASP\_NET/gamestore/level1/
11. https://professorweb.ru/my/ASP\_NET/gamestore/level2/2\_2.php
12. https://professorweb.ru/my/ASP\_NET/gamestore/level2/2\_3.php
13. https://learn.microsoft.com/ru-ru/dotnet/api/system.stackoverflowexception?view=netcore-2.2
14. https://learn.microsoft.com/ru-ru/dotnet/api/system.tuple?view=netcore-2.2
15. https://learn.microsoft.com/ru-ru/dotnet/api/system.typeloadexception?view=netcore-2.2
16. https://learn.microsoft.com/ru-ru/dotnet/api/system.uriformat?view=netcore-2.2
17. https://learn.microsoft.com/ru-ru/dotnet/api/system.urikind?view=netcore-2.2
18. https://learn.microsoft.com/ru-ru/dotnet/csharp/