Оглавление

[1. ПОСТАНОВКА ЗАДАЧИ 10](#_Toc137758446)

[1.1 Анализ предметной области 10](#_Toc137758447)

[1.2 SWOT анализ 13](#_Toc137758448)

[1.3 PEST-анализ 15](#_Toc137758449)

[1.4 Формирование требований к программному продукту 16](#_Toc137758450)

[2. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ 22](#_Toc137758451)

[2.1. Архитектура системы 22](#_Toc137758452)

[2.2. Моделирование основных сценариев системы 30](#_Toc137758453)

[2.2. Проектирование графического интерфейса пользователя 39](#_Toc137758454)

[2.3. Проектирование и разработка модели данных 48](#_Toc137758455)

[3. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА 51](#_Toc137758456)

[3.1 Разработка уровня Domain 51](#_Toc137758457)

[3.2 Разработка уровня DAL (Domain access level) 58](#_Toc137758458)

[3.2 Разработка уровня Service 65](#_Toc137758459)

[3.3 Разработка уровня Controller 74](#_Toc137758460)

[4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 80](#_Toc137758461)

[4.1. История изменений 80](#_Toc137758462)

[4. 2. Терминология тестирования программного обеспечения 81](#_Toc137758463)

[4.3. Стратегия тестирования программного обеспечения 81](#_Toc137758464)

[4.4 Определение объектов тестирования 82](#_Toc137758465)

[4.5 Архитектура тестируемой системы 84](#_Toc137758466)

[4. 6. Описание процесса тестирования 84](#_Toc137758467)

[5. ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ 87](#_Toc137758468)

[6. ТЕХНИЧЕСКАЯ ДОКУМЕНТАЦИЯ 92](#_Toc137758469)

[6.3 Определение качества программного продукта по метрикам 95](#_Toc137758470)

[7. МЕРОПРИЯТИЯ ПО ТЕХНИКЕ БЕЗОПАСНОСТИ 96](#_Toc137758471)

[7.1. Характеристика объекта проектирования 96](#_Toc137758472)

[8. ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА 99](#_Toc137758473)

[ЗАКЛЮЧЕНИЕ 107](#_Toc137758474)

[СПИСОК ИСТОЧНИКОВ ИНФОРМАЦИИ 108](#_Toc137758475)

[ПРИЛОЖЕНИЕ 1. ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ 109](#_Toc137758476)

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

БД — База Данных.

UML (Unified Modeling Language) — Язык универсального проектирования.

URL (Uniform Resource Locator) — Унифицированный указатель ресурса.

JSON (JavaScript Object Notation) — Нотация Объектов JavaScript.

API (Application programming interface) — Программный интерфейс приложения.

SQL — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

HTML (HyperText Markup Language) — язык гипертекстовой разметки.

CSS (Cascading Style Sheets) — Каскадные таблицы стилей.

HTTP (HyperText Transfer Protocol) — протокол передачи гипертекста.

GET — метод, используемый для запроса содержимого указанного ресурса.

POST — метод, применяемый для передачи пользовательских данных заданному ресурсу.

DELETE — метод, удаляющий заданный ресурс.

Cookie — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя.

JS (JavaScript) — мультипарадигменный язык программирования.

JSX – расширение языка JavaScript.

DTO (Data Transfer Object) — один из шаблонов проектирования, используется для

CRUD (Create Read Update Delete) — акроним, обозначающий четыре базовые функции, используемые при работе с базами данных.

Enum (Enumeration) — тип данных, чье множество значений представляет собой ограниченный список идентификаторов.

Header — блок в верхней части сайта, который виден на всех страницах виртуального пространства.

Footer — блок в нижней части сайта, видимый на всех страницах и дублирующий меню, а также содержащий ссылки на сторонние ресурсы.

**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.
8. ПОСТАНОВКА ЗАДАЧИ

## 1.1 Анализ предметной области

Обучение программирования, использую различные интернет-ресурсы, довольно распространенно в век информационных технологий. Существуют некоторые интернет-ресурсы, способных удовлетворить потребности пользователя в получении как теоретических знаний, так и практических навыков:

1. **«stepik.org»** — интернет-ресурс, содержащий большое количество различных курсов (рис 1.1).

В «Stepik» имеется большое количество курсов связанных с Python

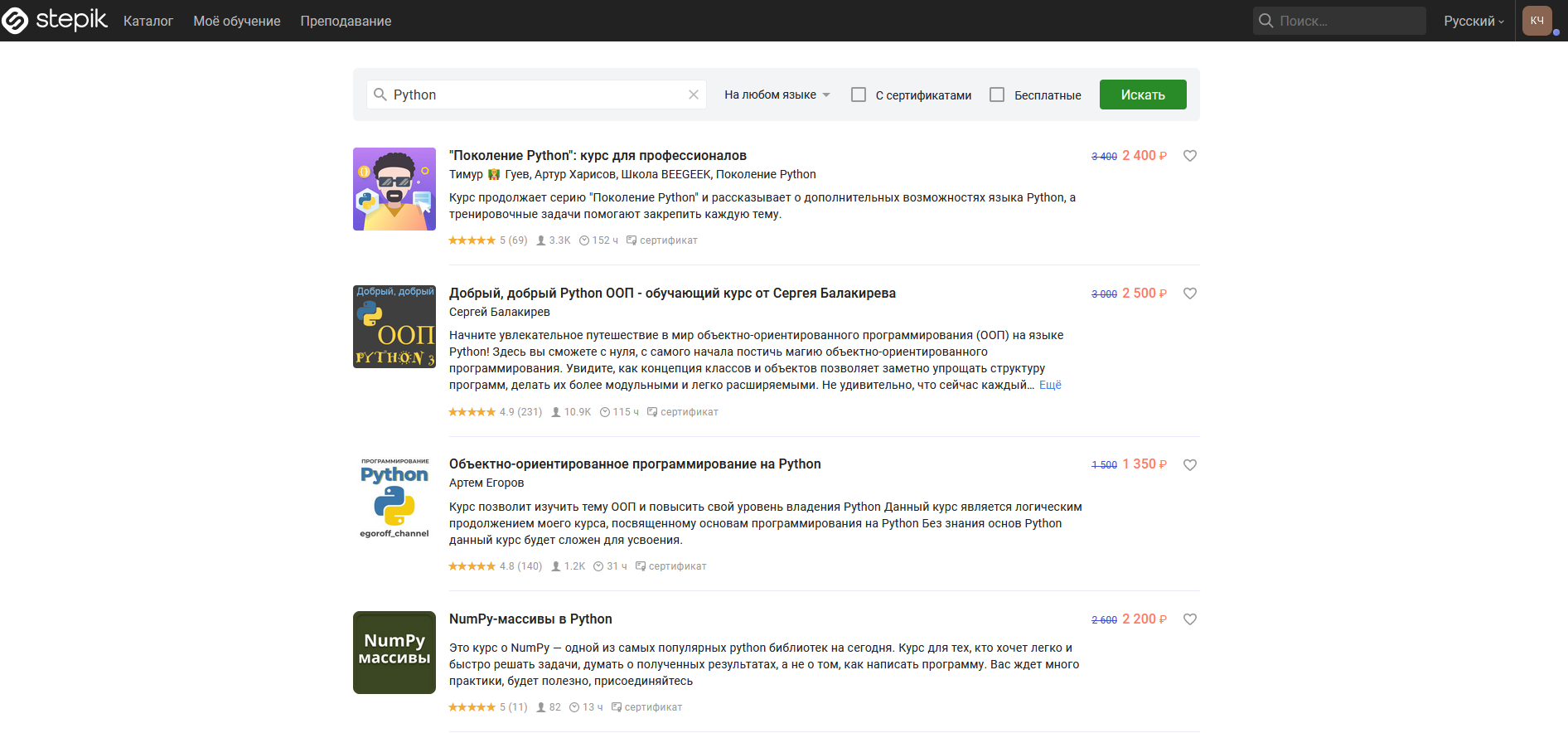


Рис. 1.1 курсы python на stepik

Данный ресурс предоставляет функционал по:

1. изучению теории;
2. просмотра как текстовых, так и видео уроков;
3. решение различных задач;
4. отслеживания процесса обучения (рис 1.2).

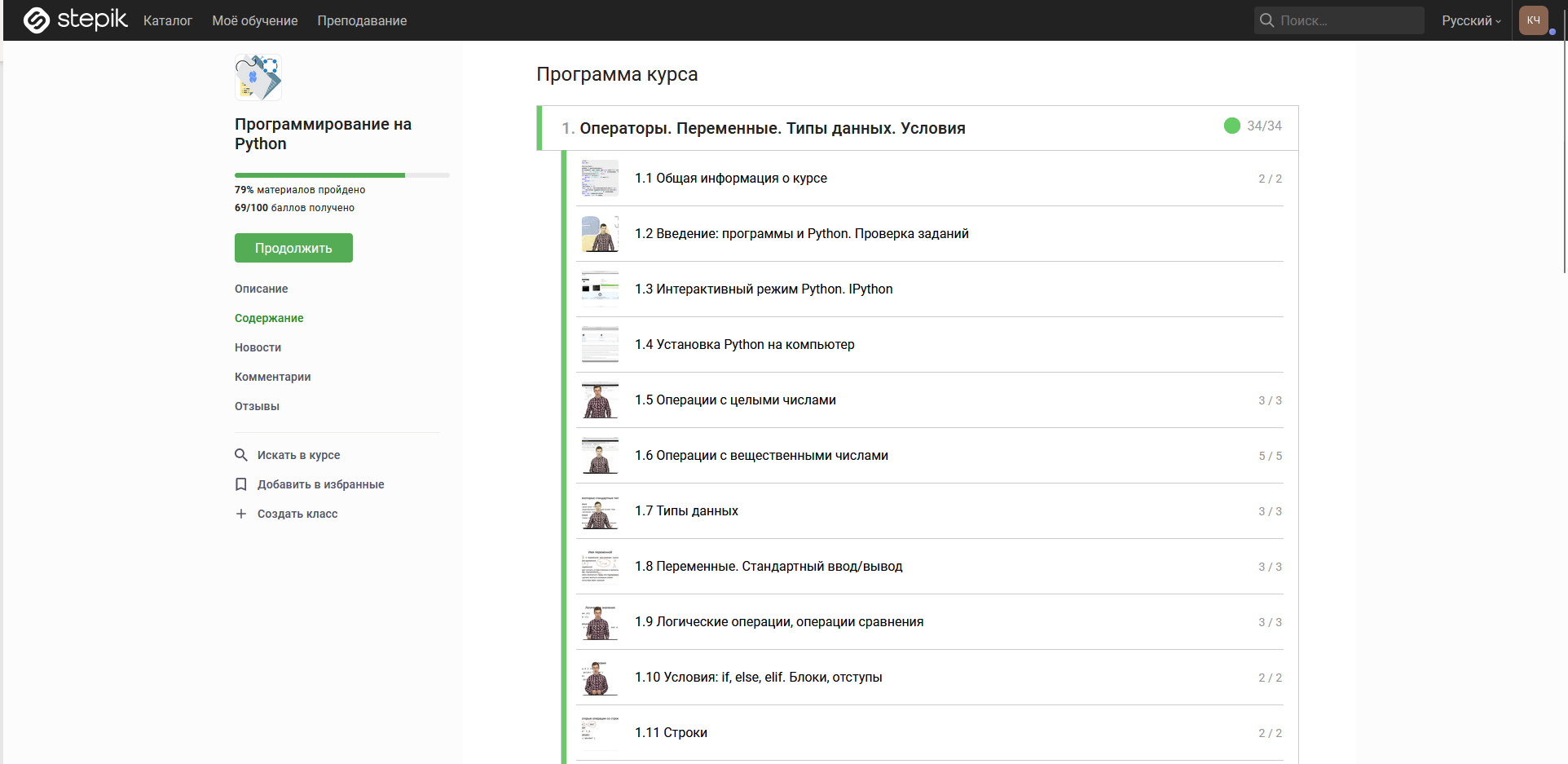


Рис. 1.2 Просмотр прогресса

1. **«Udemy.com»** (рис 1.3) — ресурс с аналогичным функционалом.

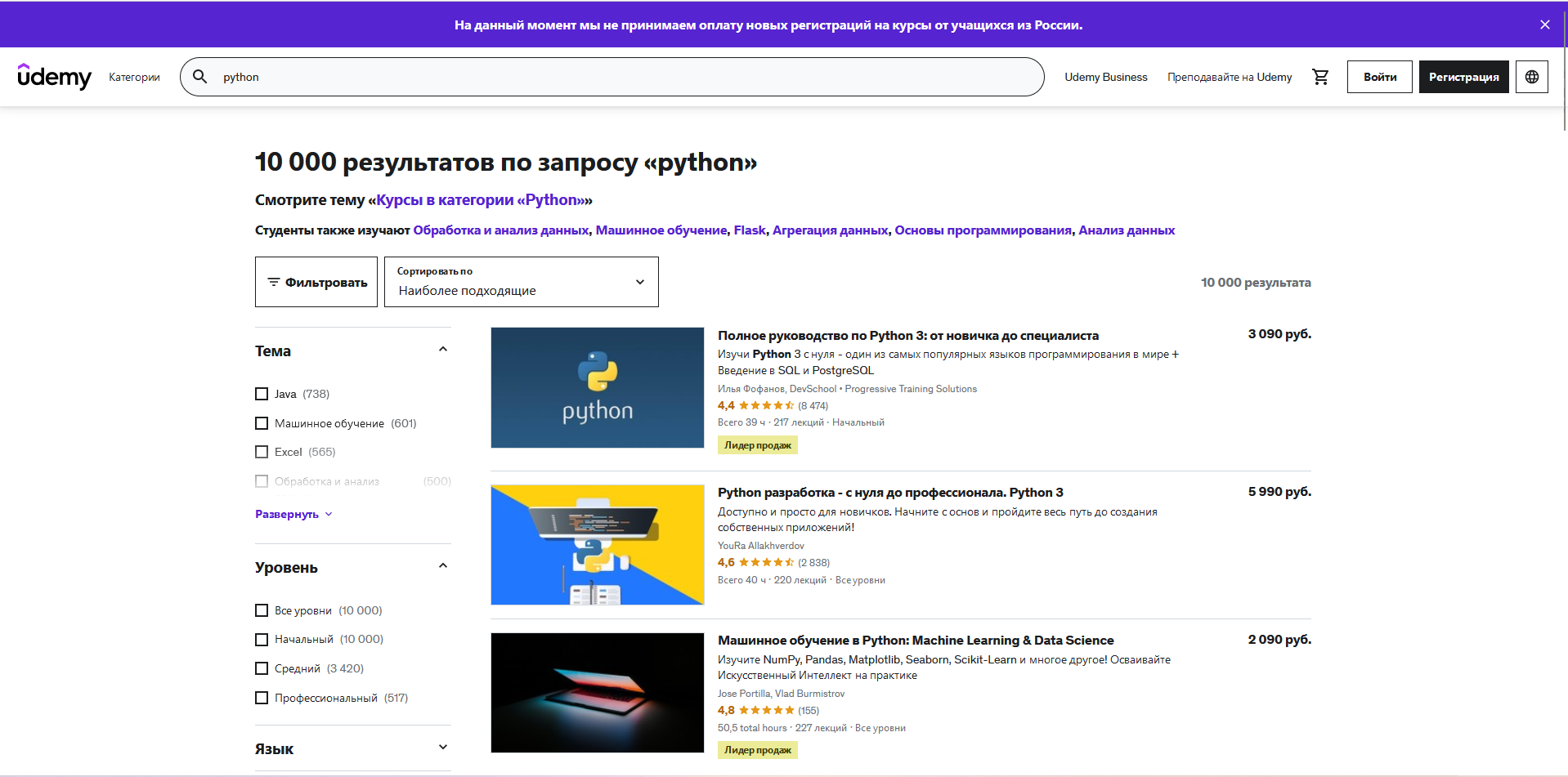


Рис. 1.3 курсы python на Undemy

1. Есть отдельные ресурсы, предоставляющие бесплатное обучение, но как правило с меньшим функционалом. Примером такого ресурса может служить «**code-basics.com**» (рис 1.4)

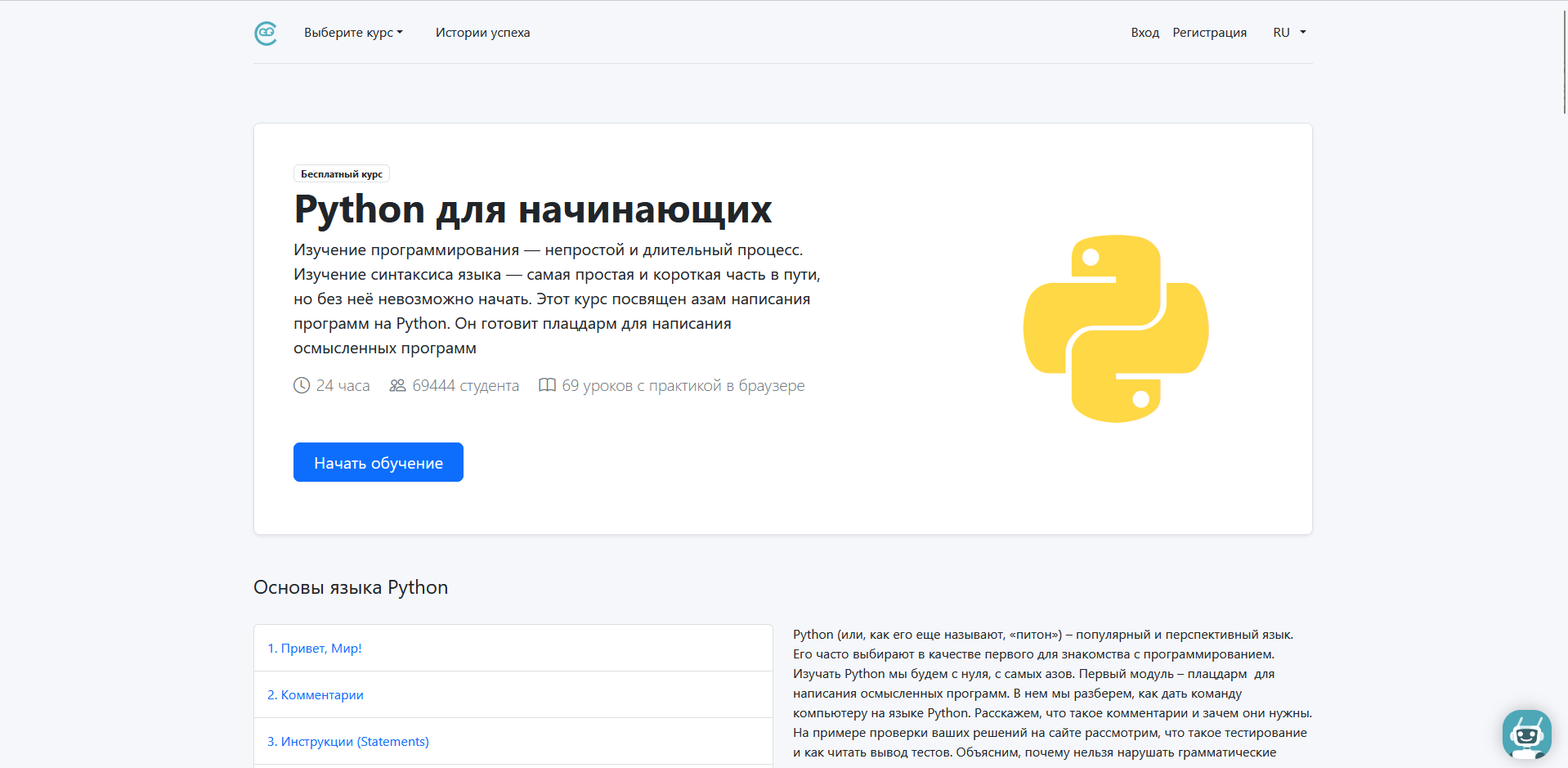


Рис. 1.4 курс python на code-basics.com

Функционал:

1. Просмотр теории;
2. выполнение практических задач;
3. регистрация и отслеживания прогресса обучения

В отличие от предыдущих ресурсов здесь нет выбора курсов, а также отсутствует возможность создания собственных.

Подводя итог можно выделить следующие общераспространенные недостатки систем-аналогов:

1. Большинство курсов платные;
2. В бесплатных курсах ограниченное количество информации;
3. функциональные ограничения.

## 1.2 SWOT анализ

Для данного проекта был проведен SWOT анализ, в результате чего были сделанны следующие выводы:

Сильные стороны:

1. Растущий спрос на изучение языка программирования Python, как одного из наиболее популярных языков в сфере разработки программного обеспечения.
2. Уникальная целевая аудитория - студенты, программисты, IT-специалисты, которым нужны качественные образовательные ресурсы по Python.
3. Использование современных технологий и инструментов разработки, таких как ASP.NET, jQuery, AJAX, CSS, HTML, Bootstrap, что позволяет создать современный и интерактивный пользовательский интерфейс.

Слабые стороны:

1. Высокая конкуренция на рынке образовательных сервисов по изучению языка программирования Python.
2. Необходимость постоянного обновления и добавления нового обучающего контента для поддержания интереса и удовлетворения потребностей пользователей.
3. Возможные технические сложности в разработке и поддержке сложной инфраструктуры, такой как серверная часть на Windows Server и использование базы данных MySQL.
4. Зависимость от доступности и стабильности интернет-соединения, так как сервис будет предоставлять онлайн-образовательные материалы.

Возможности:

1. Расширение функциональности сервиса, добавление новых образовательных материалов, упражнений, тестов и интерактивных заданий.
2. Партнерство с учебными заведениями, компаниями и организациями для предоставления дополнительных образовательных ресурсов и возможностей практического обучения.
3. Масштабирование сервиса для привлечения более широкой аудитории и расширения географического охвата.
4. Возможность получения обратной связи и рекомендаций от пользователей для улучшения и развития сервиса.

Угрозы:

1. Быстрое развитие и изменение технологий программирования, что требует постоянного обновления образовательных материалов и соответствия последним трендам.
2. Возможность появления новых конкурентов с более привлекательными предложениями и большим опытом в области образовательных сервисов.
3. Проблемы безопасности данных и угрозы хакерских атак, требующие внимания к аспектам безопасности и защите пользовательской информации.
4. Возможные ограничения и регулятивные требования в области онлайн-образования и доступности контента для разных стран и регионов.

## PEST-анализ

Также был проведен PEST анализ результаты которого представлены ниже

Политические факторы:

* Поддержка правительством и органами власти в области развития цифровой культуры и образования.
* Изменения в законодательстве и регулировании, которые могут повлиять на предоставление онлайн-образовательных услуг и защиту данных пользователей.
* Политическая стабильность и уровень государственной поддержки в сфере образования и развития IT-индустрии.

Экономические факторы:

* Финансовые возможности для инвестирования в разработку и масштабирование образовательного сервиса.
* Экономическое развитие и спрос на IT-специалистов, что может способствовать росту интереса к изучению языка программирования Python.

Социальные факторы:

* Рост популярности программирования и цифровой грамотности в обществе.
* Изменение предпочтений пользователей в области образования и предпочтение онлайн-формата обучения.
* Различия в культурных предпочтениях и образовательных потребностях разных социальных групп.

Технологические факторы:

* Быстрый темп развития технологий и инноваций в области IT, которые могут предоставить новые возможности для создания образовательных ресурсов.
* Доступность и стабильность интернет-соединения, необходимость использования передовых технологий для обеспечения качественного пользовательского опыта.
* Технические ограничения и проблемы безопасности, связанные с хранением и передачей данных пользователей.

## 1.4 Формирование требований к программному продукту

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.

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.

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.

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.

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.

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.

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.
5. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

## 2.1. Архитектура системы

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

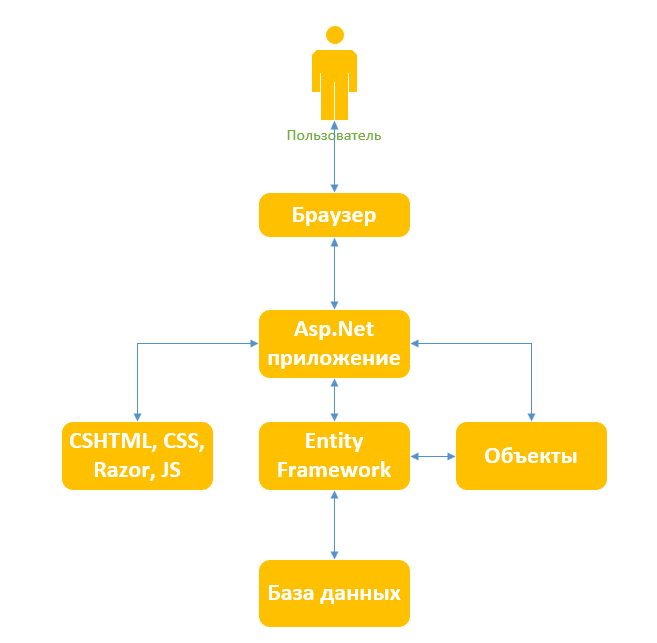


Рисунок 2.1 Общая схема взаимодействия объектов

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.

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.

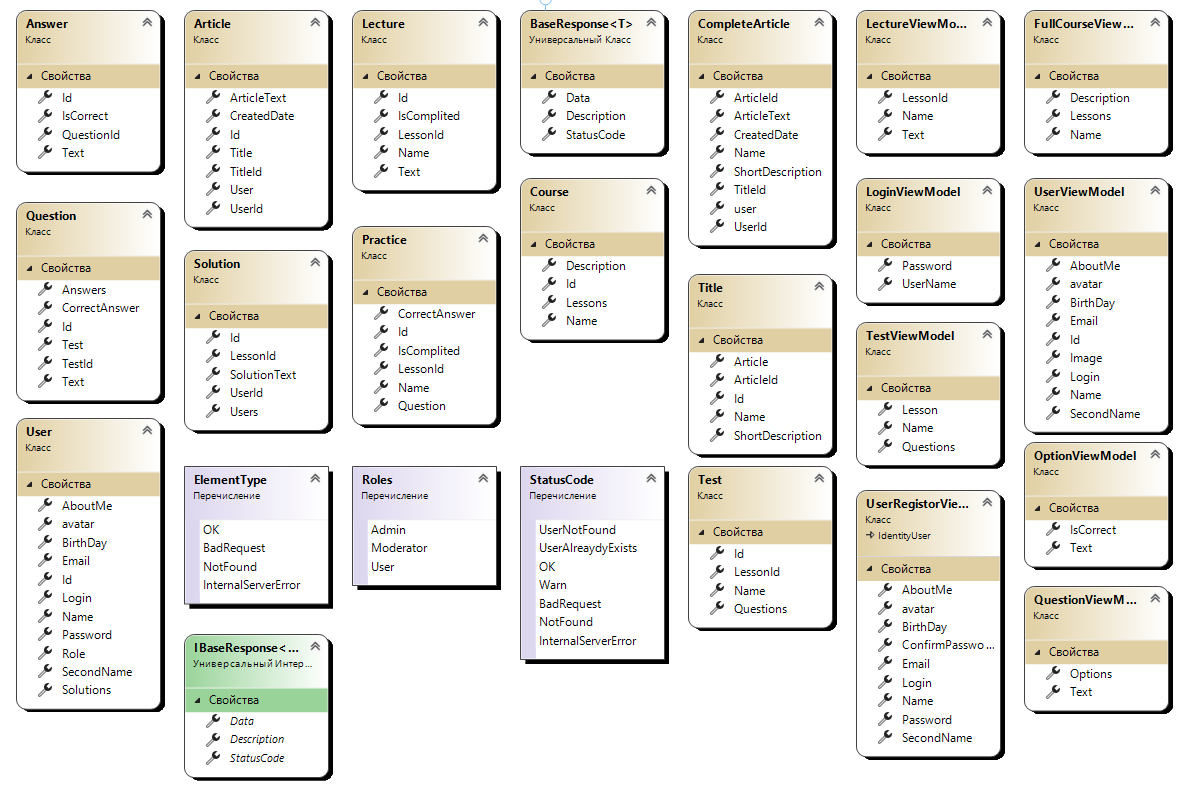


Рисунок 2.2 Диаграмма классов библиотеки Domain

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.

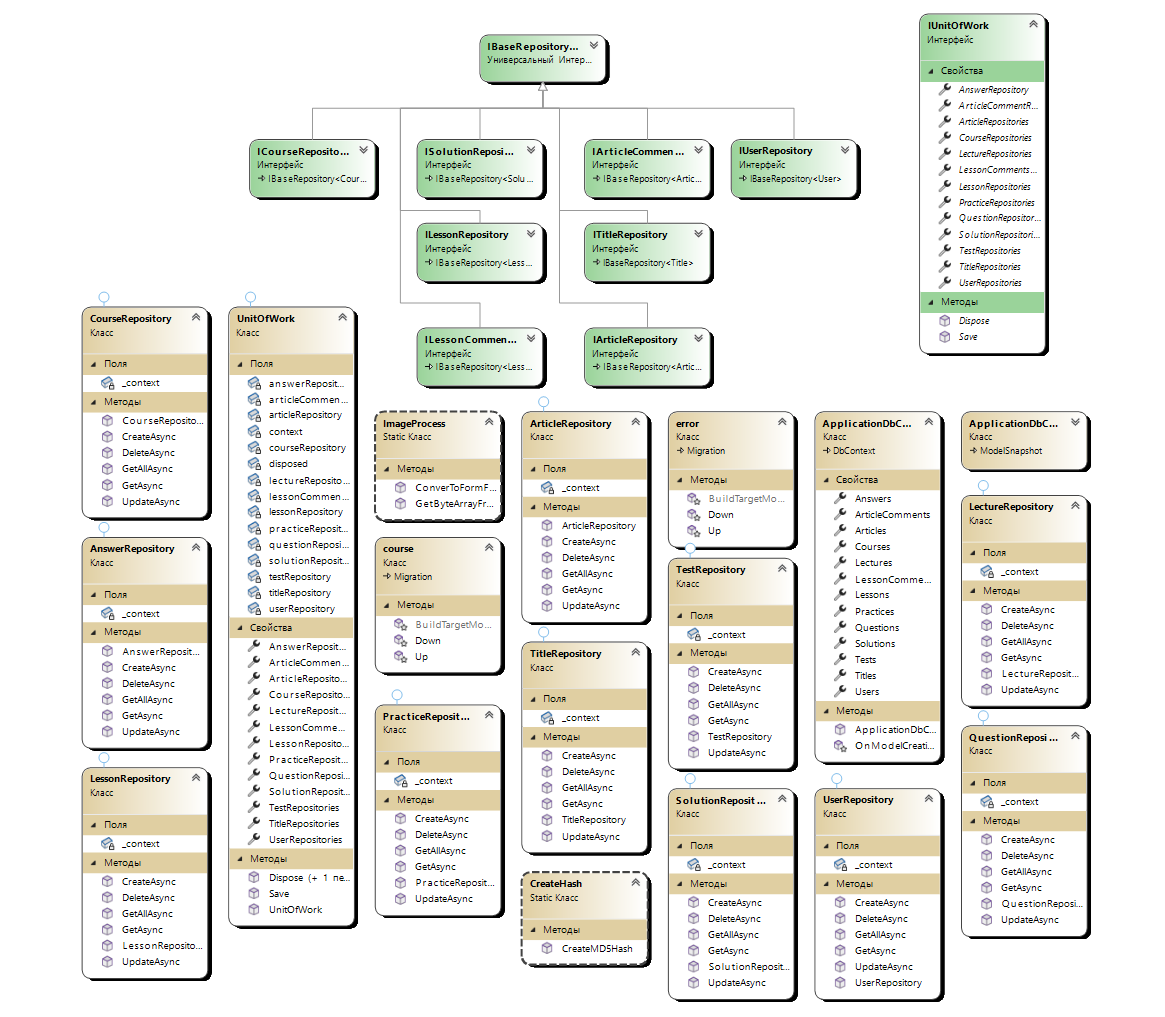


Рисунок 2.3 Диаграмма классов библиотеки Domain

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.

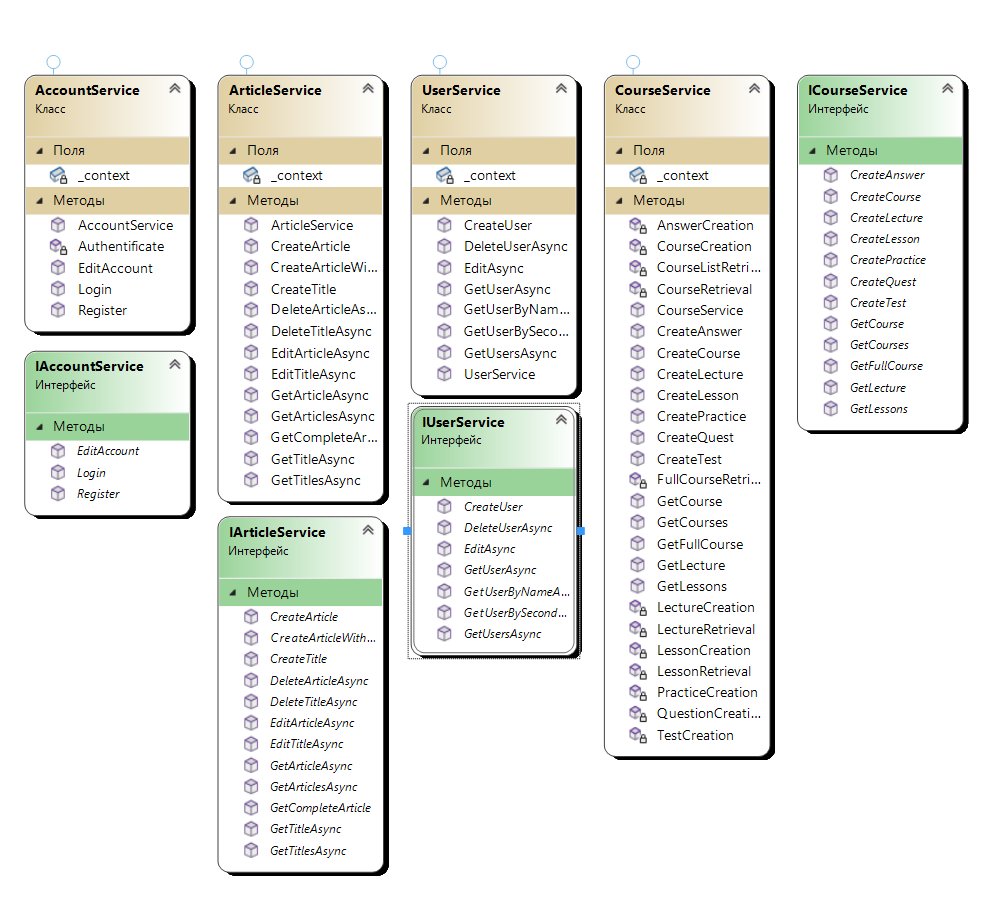


Рисунок 2.4 Диаграмма классов библиотеки Service

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

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.

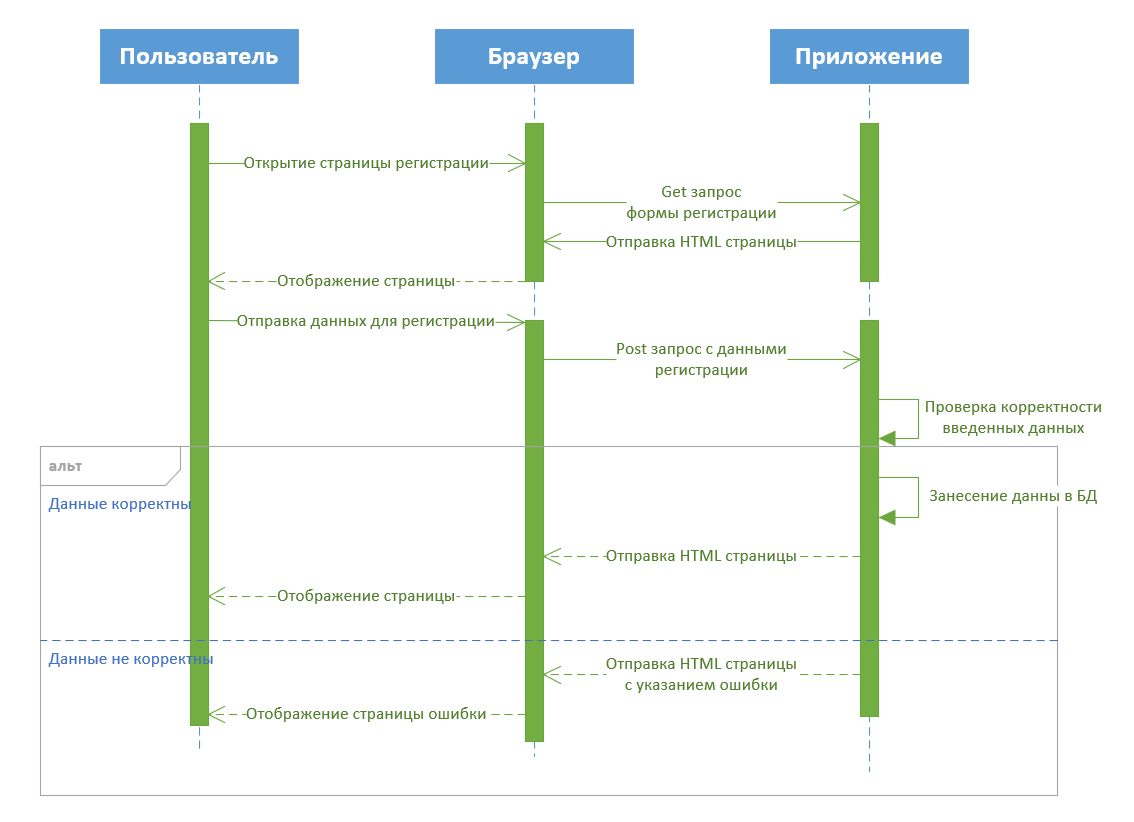


Рисунок 2.5 Диаграмма последовательности регистрации

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.

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.

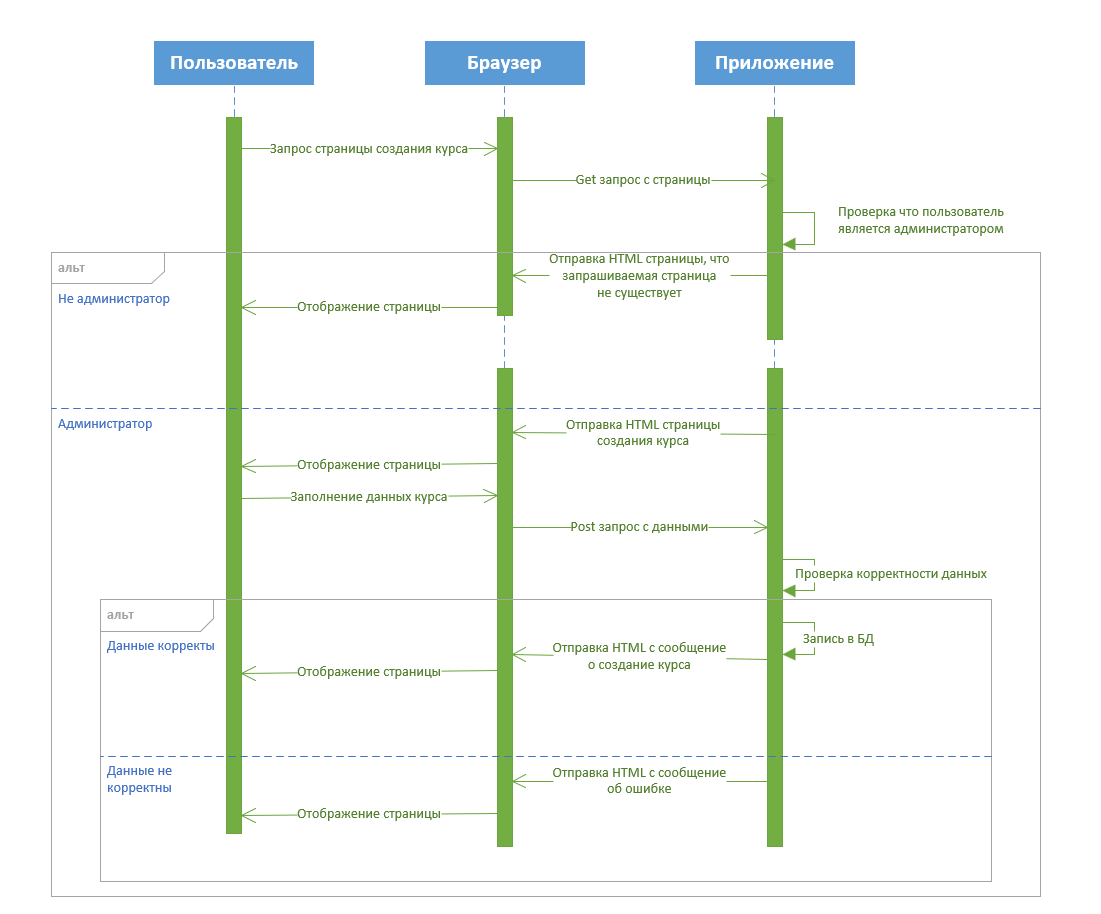


Рисунок 2.6 Диаграмма последовательности создания курс

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.

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).

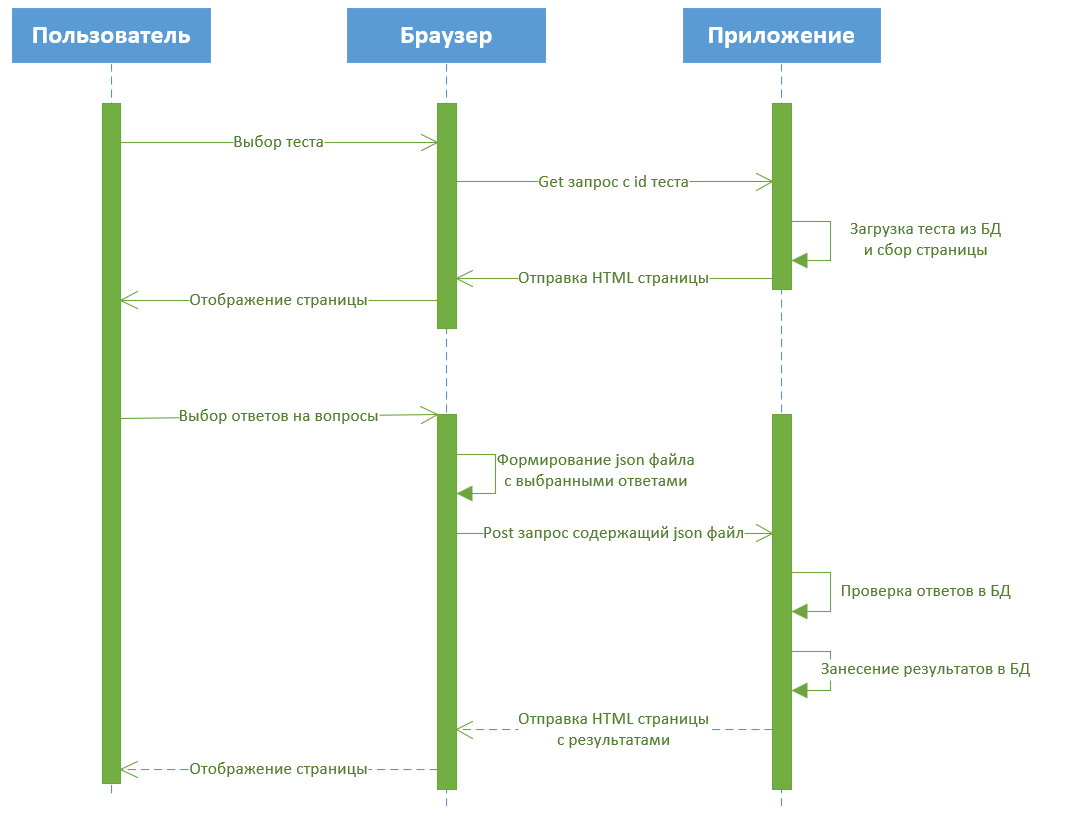


Рисунок 2.6 Диаграмма последовательности прохождение теста

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.

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.

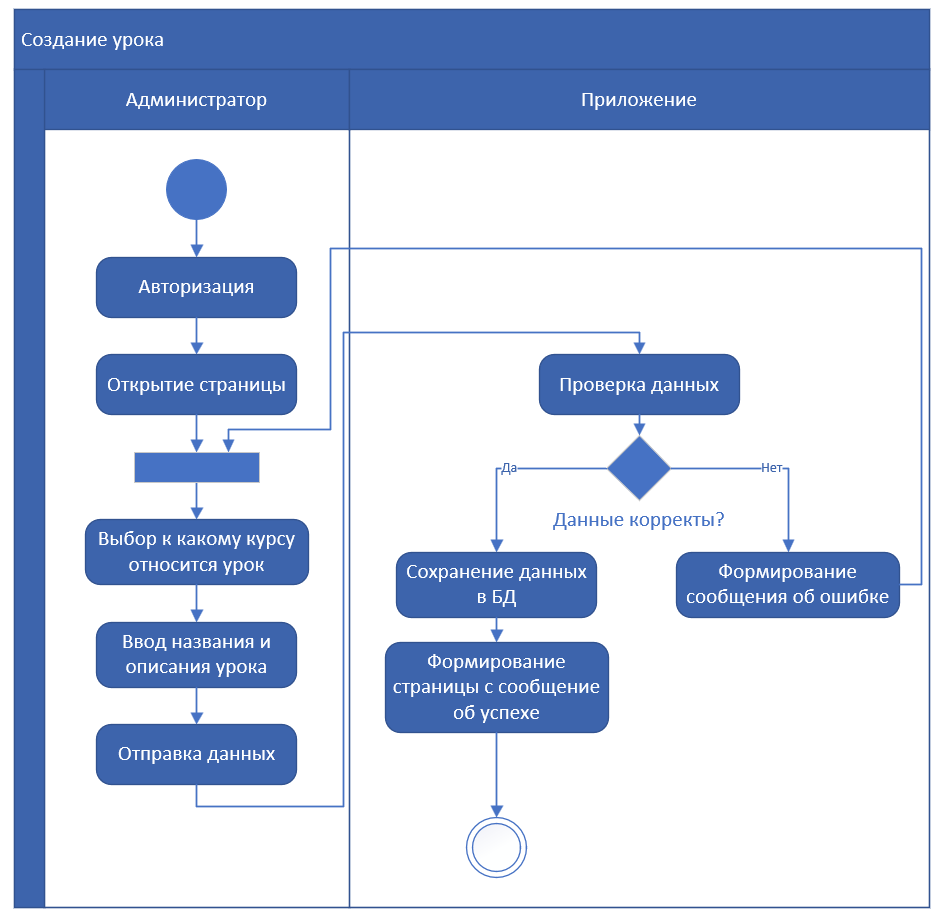


Рисунок 2.7 создание урока

Процесс создания урока представлен на рисунке 2.7.

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

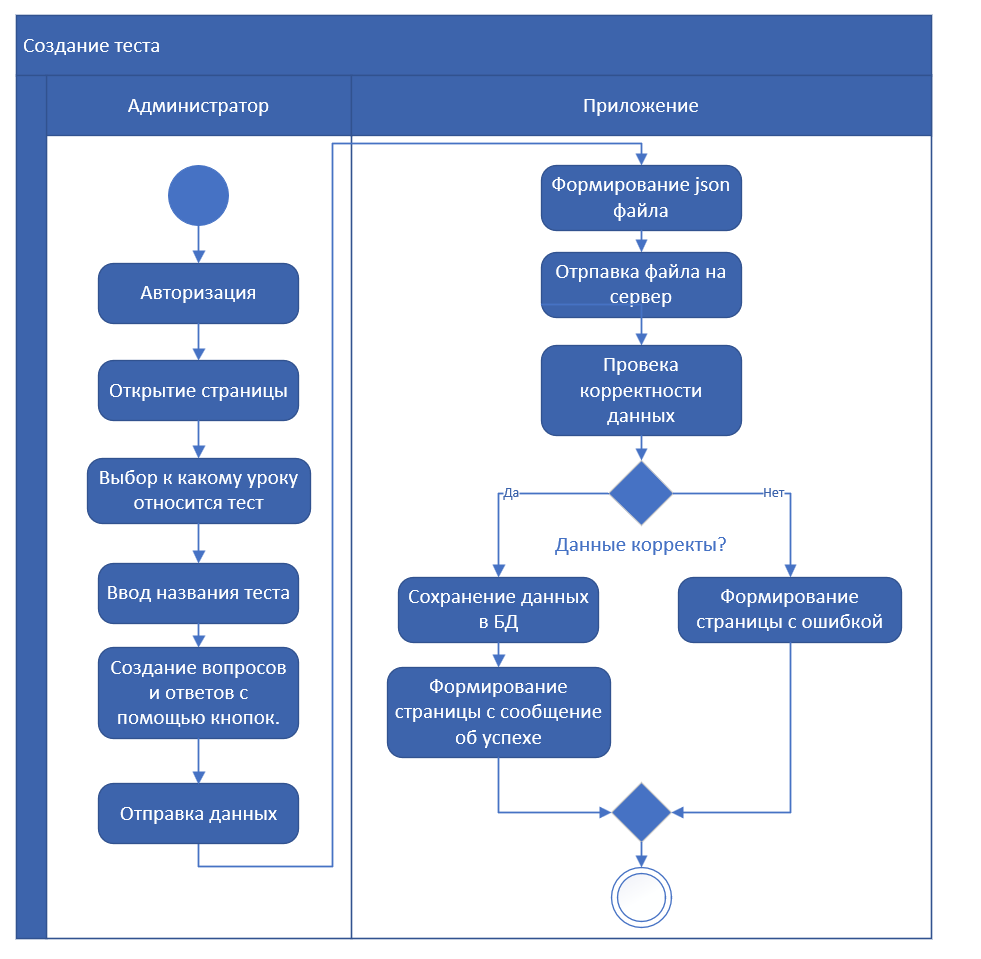


Рисунок 2.8 создание теста

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.

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.

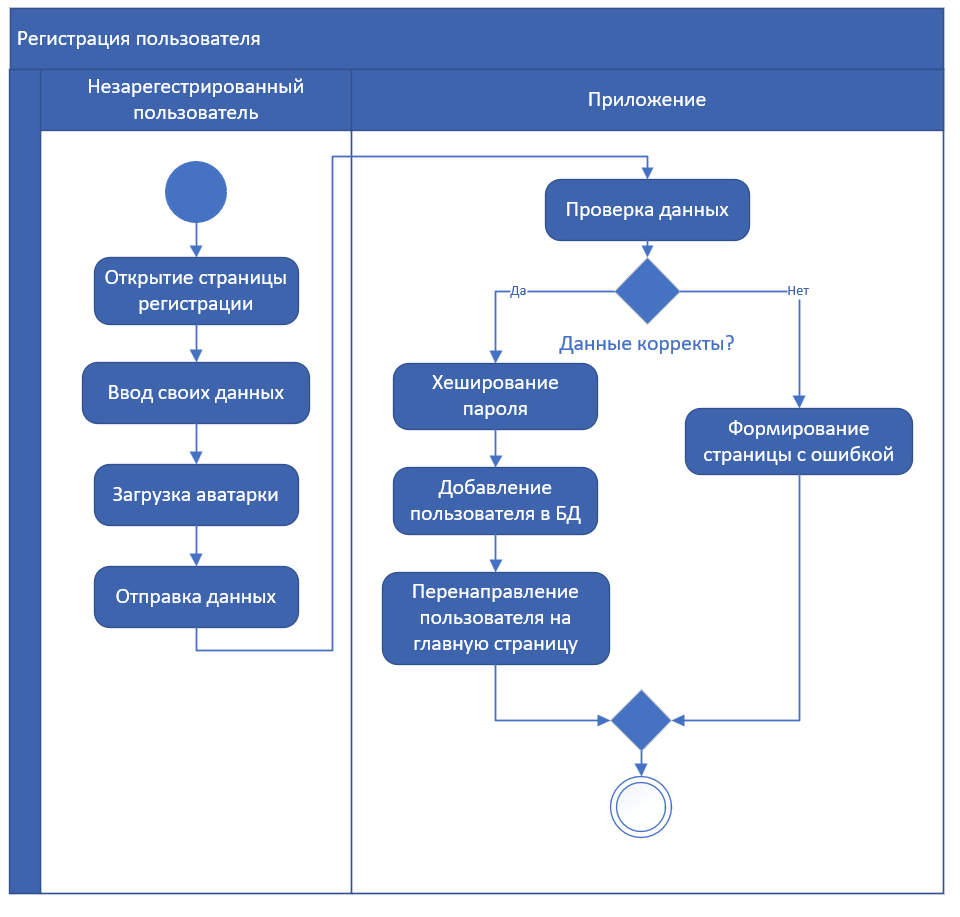


Рисунок 2.9 Регистрация пользователя

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.

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.

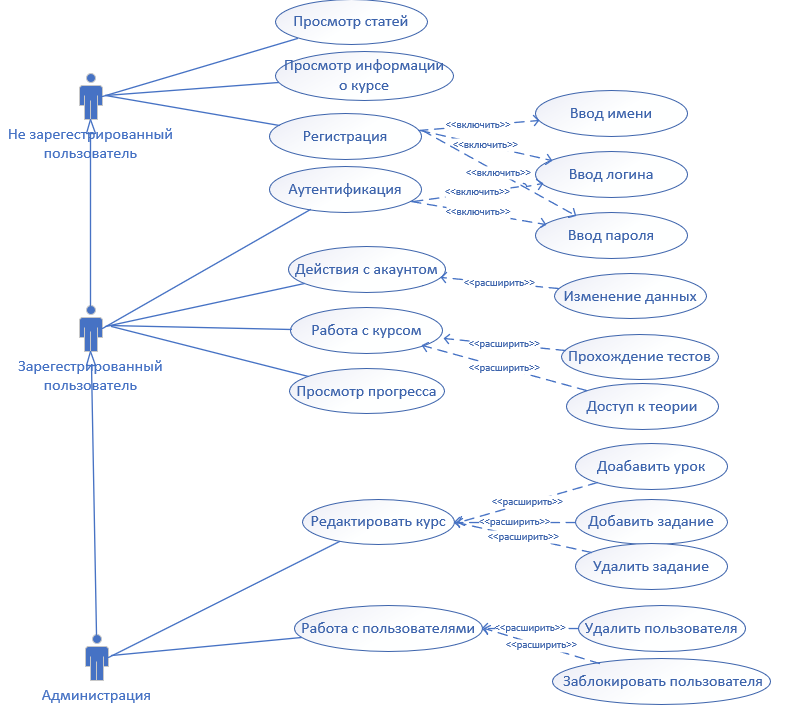


Рисунок 2.10 варианты использования

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

## 2.2. Проектирование графического интерфейса пользователя

Главная страница:

В ходе разработки графического интерфейса пользователя были использованы программно-аппаратные средства Adobe Photoshop 21 для создания макета сайта, а также для корректного отображения изображений.

Разработаны четыре основных цветовой палитры, такие как:

* #868686 - серый;
* #00897b – светло зелёный;
* #e4fdf9 – темно зелёный;
* #ffffff – белый;
* #000000 - черный

На главной странице имеются следующие компоненты:

* 1. Шапка с элементами навигации (Домашняя страница, контакты, обучение, блог, вход);
  2. Ниже расположен баннер с двумя кнопками, одна из которых ведет на страницу с уроками, а вторая на страницу со списком статей (рисунок 2.11);

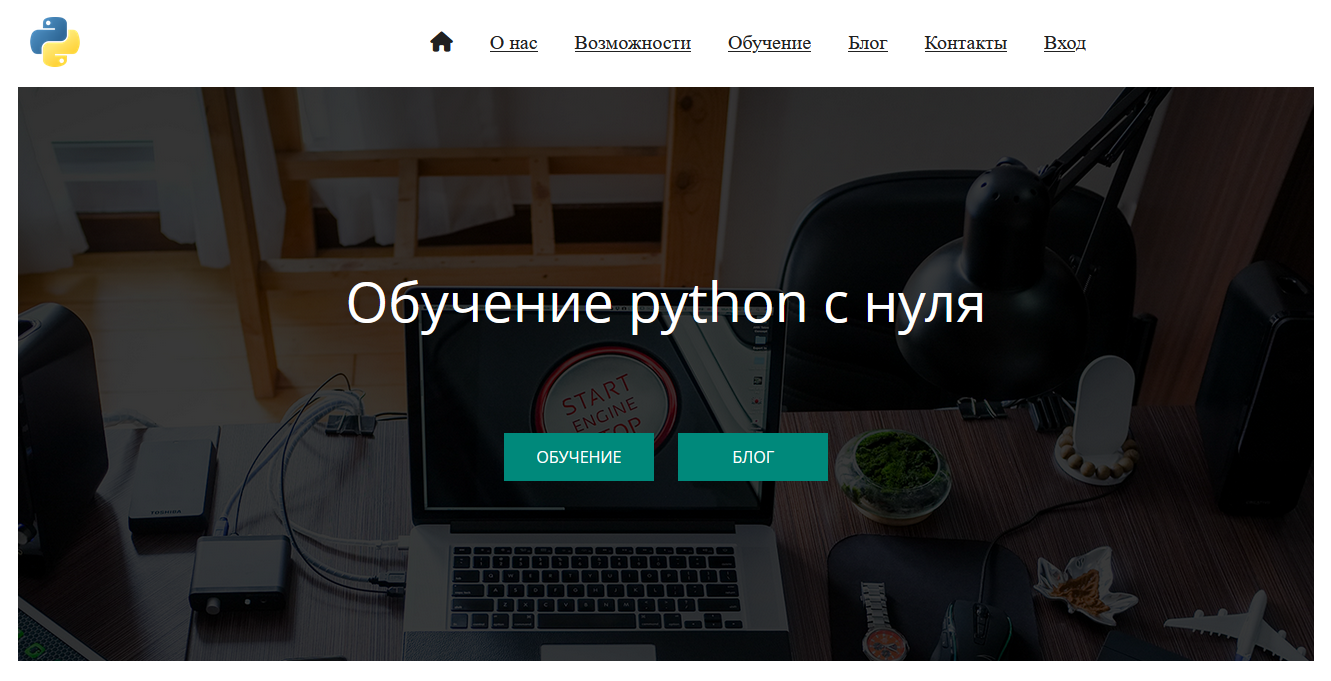


Рисунок 2.11 шапка и баннер сайта

* 1. В следующем блоке расположена общая информация о сервисе (рисунок 2.12);

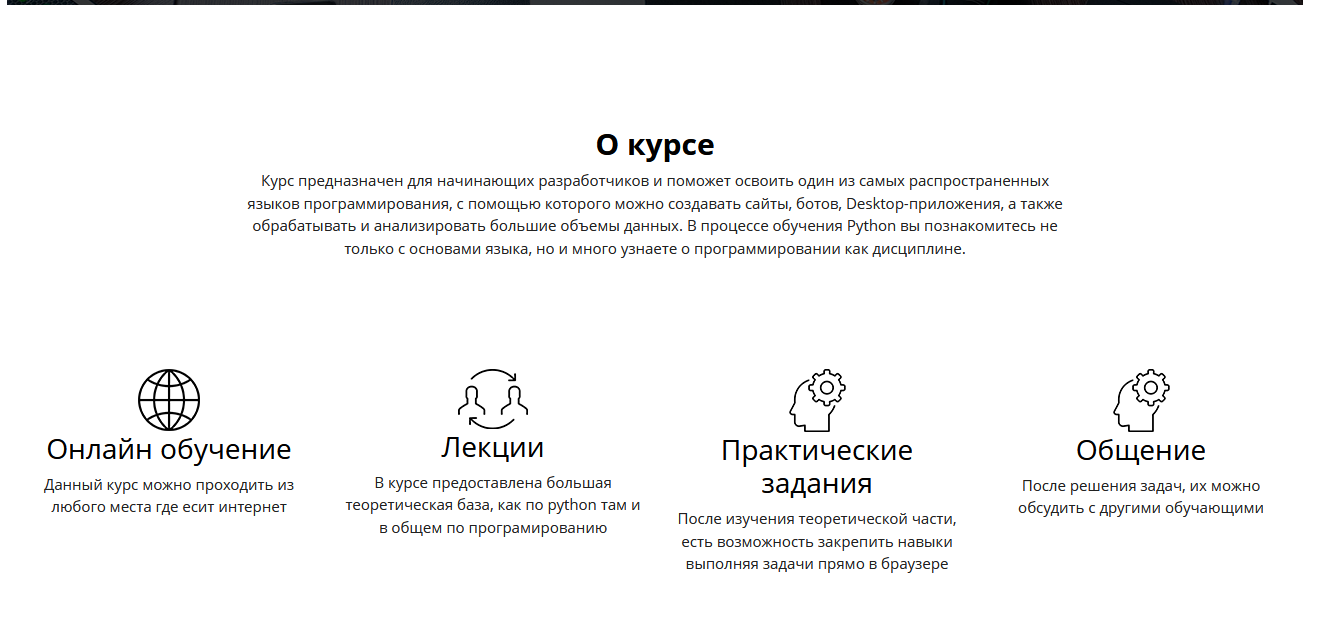


Рисунок 2.12 общая информация о сайте

* 1. Далее расположен блок с карточками, на которых расположены ссылки, и общая информация о том куда они ведут (рисунок 2.13);

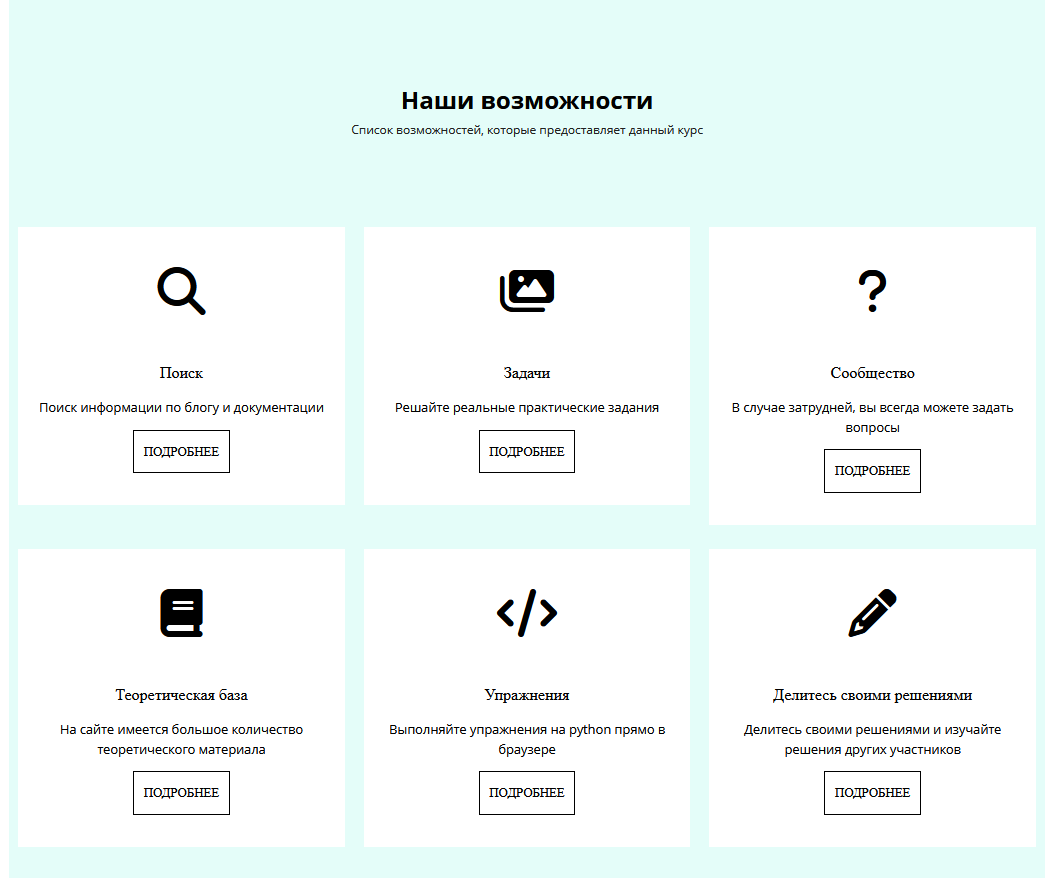


Рисунок 2.13 карточки

* 1. В последнем блоке находится блок с контактами (рисунок 2.14).

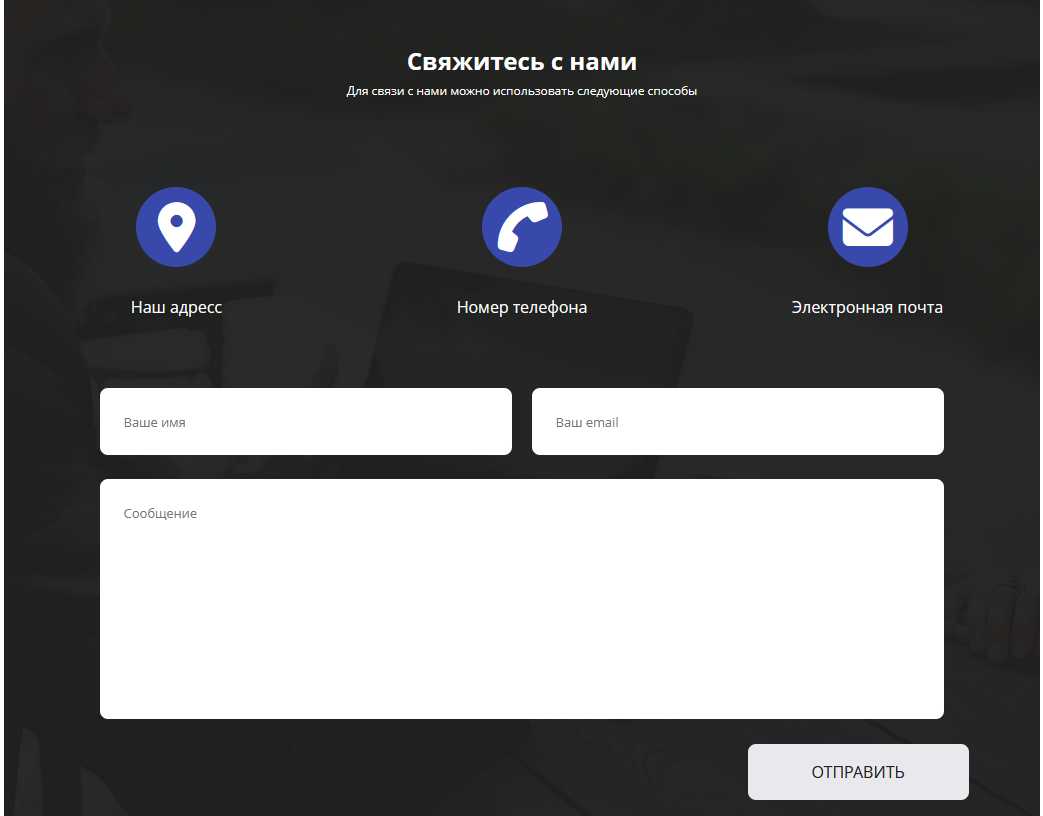


Рисунок 2.14 контакты

Страницы обучения:

На главной странице курса (рис 2.15) располагается шапка сайта с навигацией, которая позволяет посетителям легко перемещаться по различным разделам и страницам курса.

Под шапкой сайта находится описание курса, которое предоставляет общую информацию о том, что студенты смогут изучить и достичь, пройдя данный курс. Описание курса содержит информацию о его целях, основных темах, уровне сложности, целевой аудитории и возможностях, которые предоставляет курс.

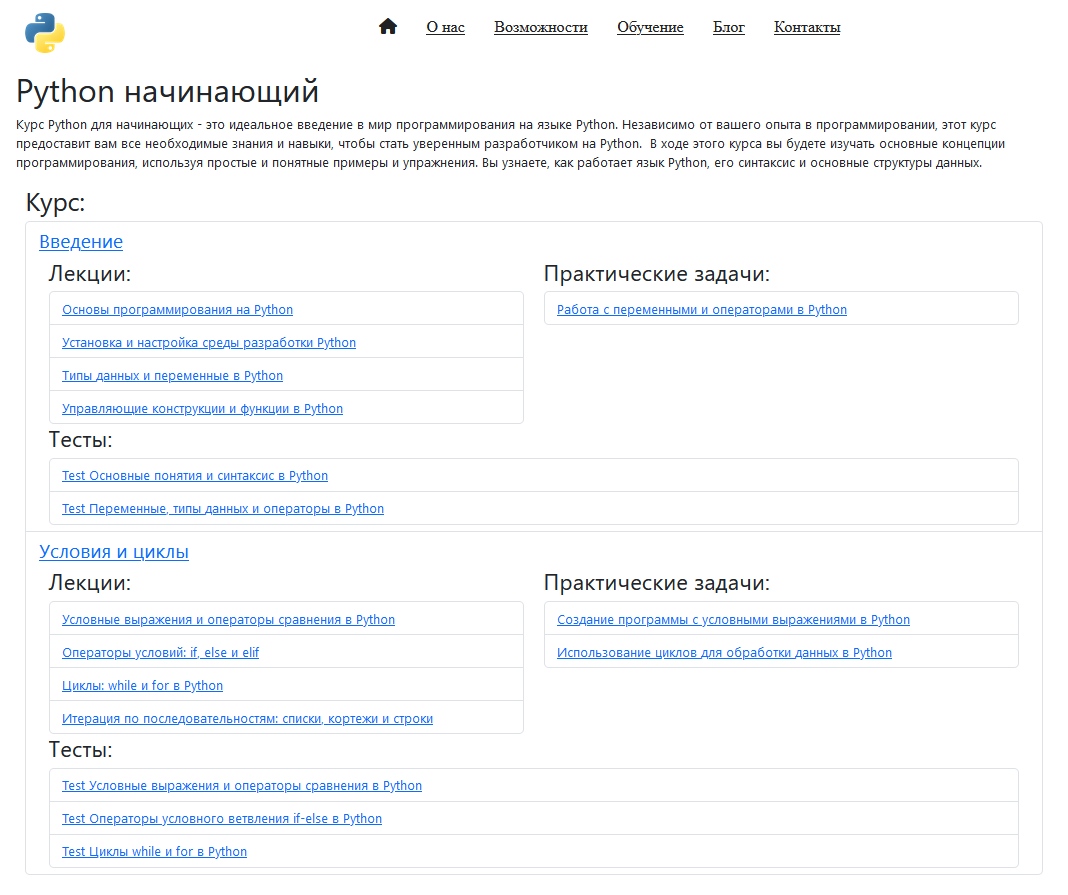


Рисунок 2.15 главная страница сайта

Далее на главной странице курса располагается список уроков в виде карточек. Каждая карточка представляет отдельный урок курса и содержит информацию о нем, заголовок урока, краткое описание и обложку/изображение, связанное с уроком. Карточки уроков могут быть представлены в виде сетки или вертикального списка для удобного просмотра и выбора.

При щелчке на карточку урока открывается отдельная страница урока, на которой размещается подробная информация о данном уроке, включая список лекций, тестов и практических заданий, связанных с данным уроком.

В разделе лекций урока представлен список лекций, которые студенты могут изучить для усвоения материала. Каждая лекция может иметь заголовок, краткое описание и содержание, представленное в текстовом или мультимедийном формате. Студенты могут просматривать лекции в определенном порядке и изучать материалы пошагово.

В разделе тестов урока предлагаются тесты и вопросы для проверки понимания материала. Это может включать множественный выбор, открытые вопросы или другие типы тестов. Тесты помогают студентам оценить свои знания и подготовиться к дальнейшему изучению курса.

В разделе практических заданий урока предлагаются реальные задачи и упражнения, которые студенты должны решить самостоятельно. Практические задания позволяют применить полученные знания на практике, развить навыки программирования и протестировать свои умения.

При нажатии на лекцию пользователь переходит на другую страницу   
(рис 2.16) содержащею её

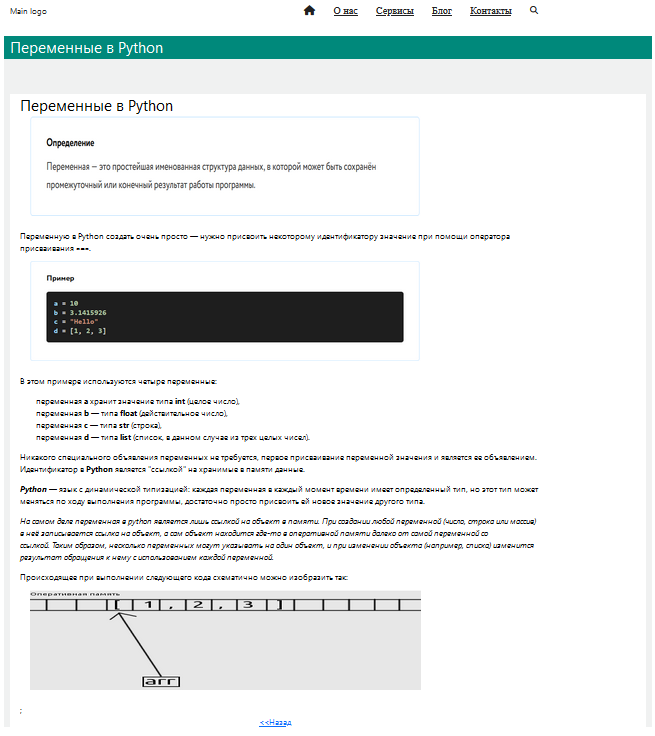


Рисунок 2.16 Страница лекции

Страницы блога:

На главной странице блога (рис 2.17), располагается список статей. Каждая статья имеет заголовок, краткое описание, имя автора, его аватар и дату создания.

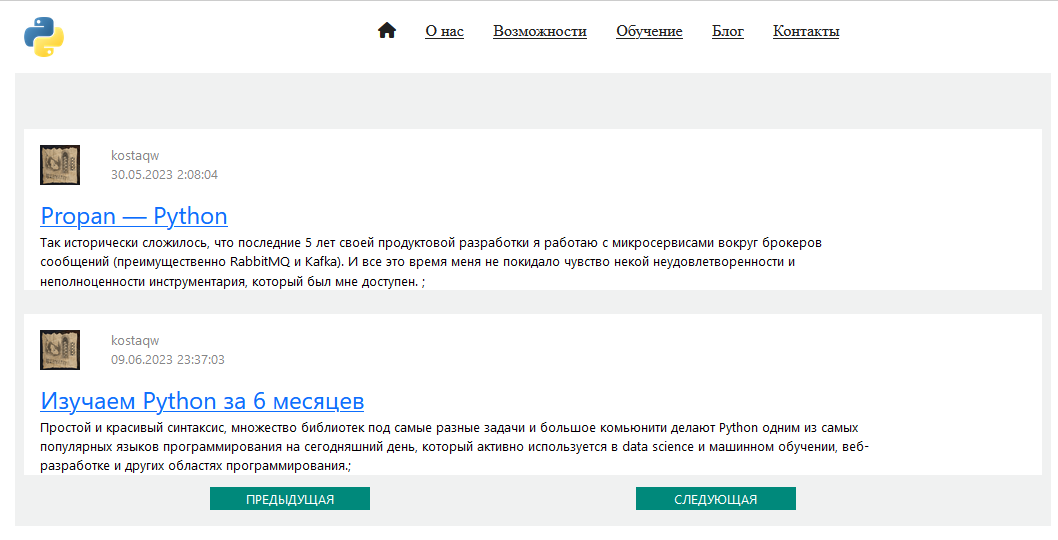


Рисунок 2.17 Главная страница блога

Страница позволяет пользователям просматривать и выбирать интересующие их статьи.

При щелчке на статью открывается отдельная страница статьи, где представлена полная статья (рис 2.18).

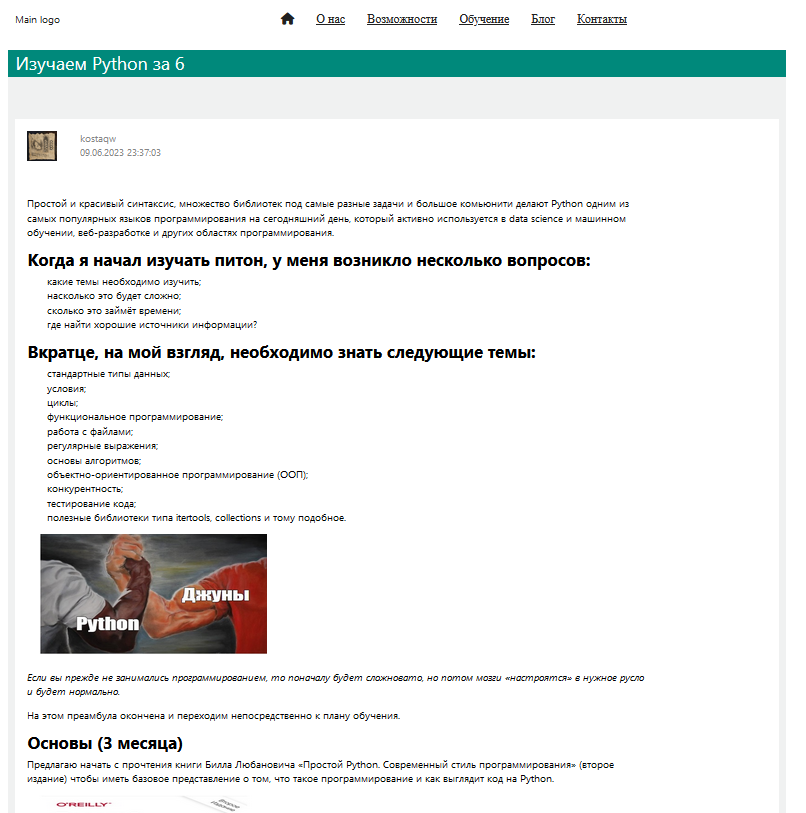


Рисунок 2.18 статья из блога

Таким образом, страница предоставляет удобный способ просмотра доступного контента и позволяет пользователям быстро ориентироваться в представленных статьях, их авторах.

Страницы создания контента:

Для создания различного контента были созданы следующие страницы:

* + 1. Страница создания статей;
    2. страница создания лекций;
    3. страница создания тестов;

На странице создания статей (рис 2.19), как и на всем остальном сайте имеется шапка, которая позволяет перемещаться по всему ресурсу.

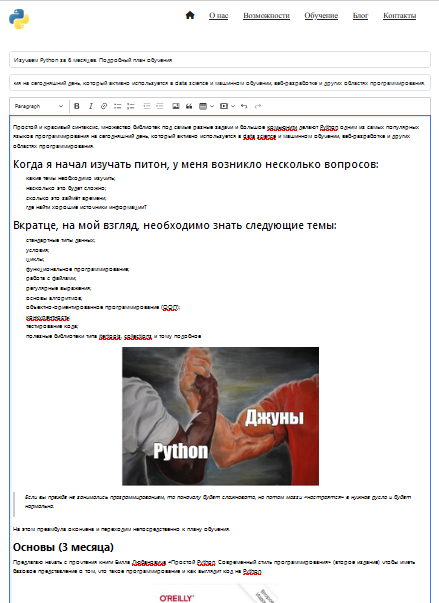


Рисунок 2.19 создание статьи

Под шапкой расположены два поля, в которые вводятся название статьи и её краткое описание, которое будет показано на странице со всех статьями. Далее идёт редактор под название ckeditor который позволяет легко создавать форматированные статьи, загружать фото и видео и генерировать html код, который будет помещён в БД, после всех полей находится кнопка отправить.

Страница создания лекций (рис 2.20) имеет похожую структуру, за исключением того, что там первым полем является выпадающий список, в котором администратор или модератор может выбрать к какому уроку относится данная лекция.

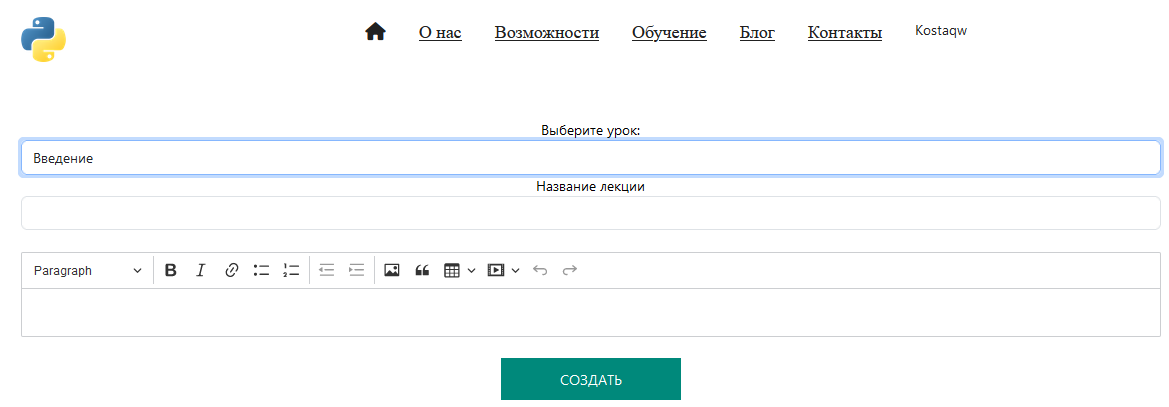


Рисунок 2.20 создание урока

На странице для создания (рисунок 2.21) тестирования, администраторы сайта имеют возможность подготовить и настроить тестовые задания для конкретного урока. Форма на странице позволяет им ввести название и выбрать урок, к которому этот тест будет относиться. Уроки представлены в виде выпадающего списка, где они могут выбрать урок для своего теста.

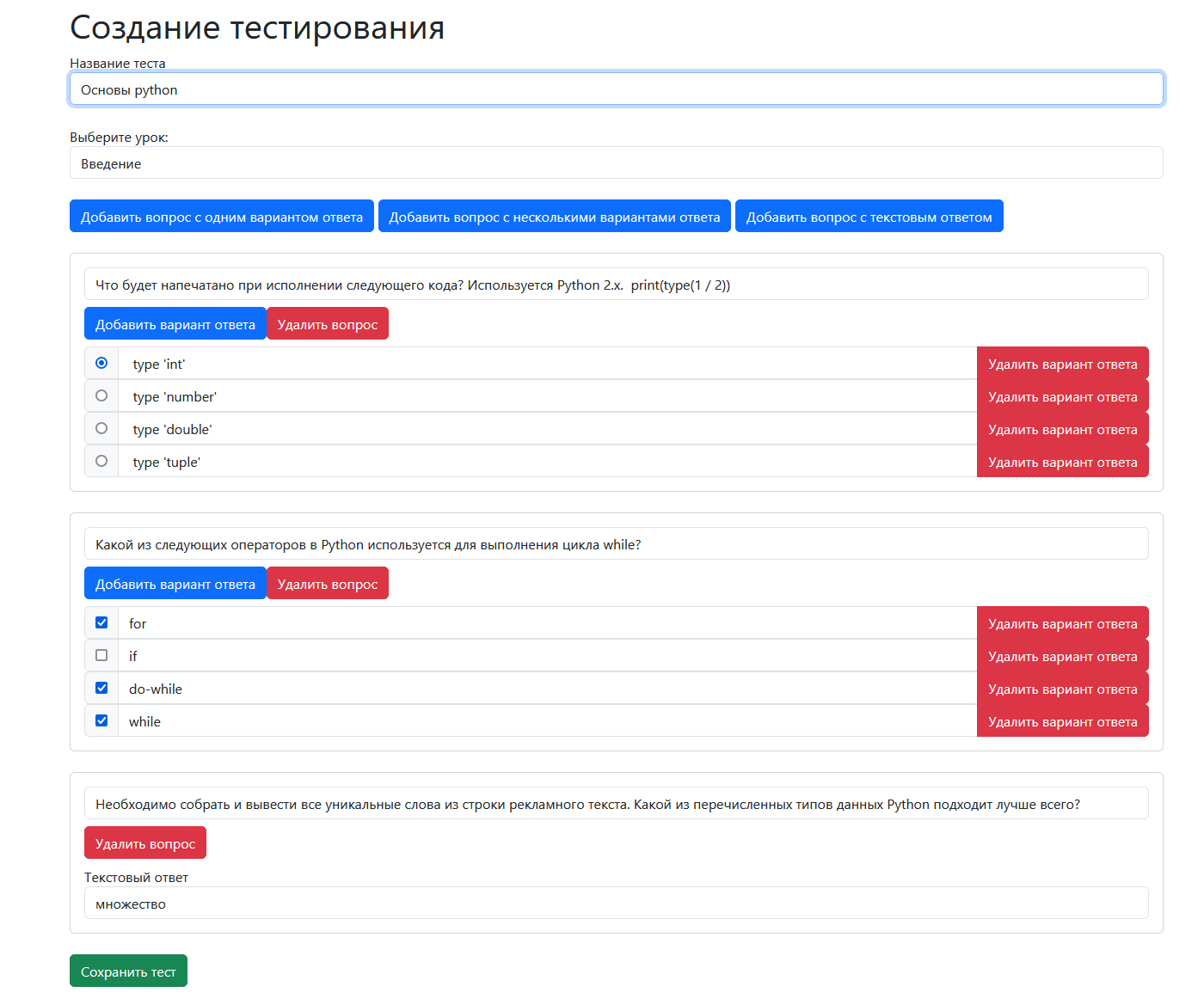


Рисунок 2.21 создание теста

После ввода названия теста и выбора урока, администраторы могут начать добавлять вопросы. Вопросы могут иметь различные типы, такие как множественный выбор, открытый вопрос и одиночный выбор. Администраторы могут добавлять текст вопроса и его варианты ответов. Для каждого варианта ответа администраторы указывают, является ли он правильным или нет.

Помимо вопросов, есть возможность добавлять и текстовые задания к тесту. Текстовые задания позволяют предоставить свободный ответ на вопрос или задачу, связанную с уроком.

После завершения создания теста, можно его сохранить. Сохраненный тест будет связан с выбранным уроком и будет доступен для прохождения студентами, которые изучают данный курс.

Страница регистрации:

На данной странице (рис 2.22) размещена форма регистрации.

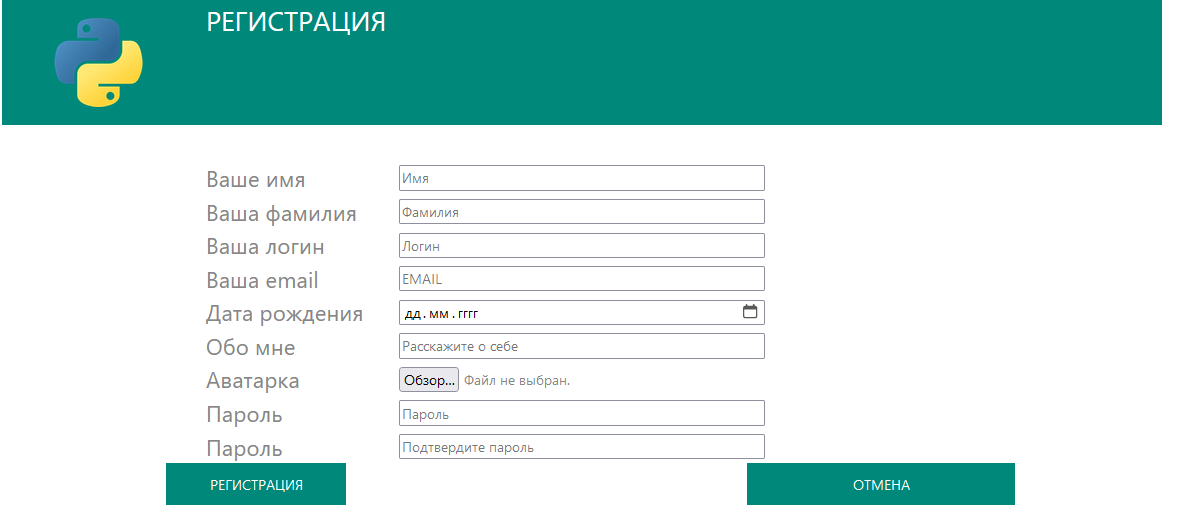


Рисунок 2.22 страница регистрации

Страница имеет следующую структуру:

В верхней части страницы находится заголовок и логотип сайта. Далее следует раздел "Профиль", который содержит форму регистрации. Форма включает различные поля, предназначенные для ввода информации администратора.

В поле "Ваше имя" пользователь должен указать свое имя.

В поле "Ваша фамилия" пользователь должен указать свою фамилию.

В поле "Ваш логин" пользователь должен указать свой логин для входа на сайт.

В поле "Ваш email" пользователь должен указать свой адрес электронной почты.

В поле "Дата рождения" пользователь должен указать свою дату рождения.

В поле "Обо мне" пользователь может рассказать немного о себе.

В поле "Аватарка" пользователь может выбрать изображение для своего профиля.

В поле "Пароль" пользователь должен ввести пароль для входа на сайт.

В поле "Подтвердите пароль" пользователь должен повторно ввести пароль для подтверждения.

## 2.3. Проектирование и разработка модели данных

В разрабатываемом приложение можно выделить следующие сущности:

* пользователи
* вопросы
* ответы
* статьи
* курсы
* лекции
* уроки
* практические задания
* тесты
* заголовки статей

В ходе разработки базы данных было выделено 10 сущностей и 13 связей типа один ко многим", а именно:

Таблица Users (Пользователи):

* Ссылка из столбца CourseId таблицы Lessons на столбец Id таблицы Courses.
* Ссылка из столбца UserId таблицы Articles на столбец Id таблицы Users.

Таблица Questions (Вопросы):

* Ссылка из столбца TestId таблицы Questions на столбец Id таблицы Tests.

Таблица Answers (Ответы):

* Ссылка из столбца QuestionId таблицы Answers на столбец Id таблицы Questions.

Таблица Articles (Статьи):

* Ссылка из столбца UserId таблицы Articles на столбец Id таблицы Users.
* Ссылка из столбца TitleId таблицы Articles на столбец Id таблицы Titles.
* Ссылка из столбца ArticleCommentId таблицы Articles на столбец Id таблицы ArticleComments.

Таблица Courses (Курсы):

* Ссылка из столбца CourseId таблицы Lessons на столбец Id таблицы Courses.

Таблица Lectures (Лекции):

* Ссылка из столбца LessonId таблицы Lectures на столбец Id таблицы Lessons.

Таблица Lessons (Уроки):

* Ссылка из столбца CourseId таблицы Lessons на столбец Id таблицы Courses.

Таблица Practices (Практические задания):

* Ссылка из столбца LessonId таблицы Practices на столбец Id таблицы Lessons.

Таблица Tests (Тесты):

* Ссылка из столбца LessonId таблицы Tests на столбец Id таблицы Lessons.

Таблица Titles (Заголовки статей):

* Ссылка из столбца ArticleId таблицы Titles на столбец Id таблицы Articles.

Далее представлена диаграмма сущность-связь (рис.2.23) для централизованной базы данных всей системы, а именно всех модулей, которые будут присутствовать в данной системе.

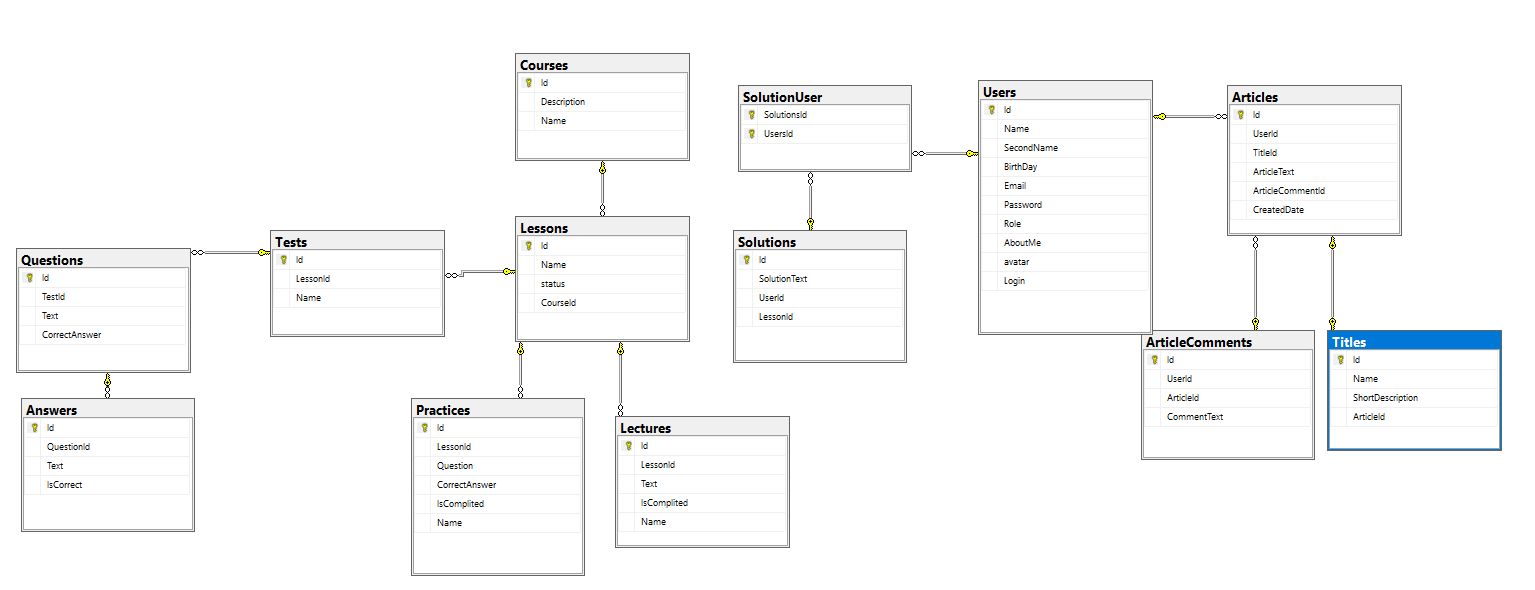


Рисунок 2.23 ER диаграмма БД

Разрабатываемая база данных предназначена для хранения всей долговременной информации системы, является реляционной и реализована при помощи системы управления базами данных MS SQL Server.

Данная БД имеет следующую структуру

1. Таблица Users (Пользователи):
   * + Id: Уникальный идентификатор пользователя.
     + Name: Имя пользователя;
     + Email: Электронная почта пользователя;
     + CourseId: Идентификатор курса, на который пользователь подписан.
2. Таблица Questions (Вопросы):
   * + Id: Уникальный идентификатор вопроса;
     + Text: Текст вопроса;
     + TestId: Идентификатор теста, к которому относится вопрос.
3. Таблица Answers (Ответы):
   * + Id: Уникальный идентификатор ответа;
     + Text: Текст ответа;
     + QuestionId: Идентификатор вопроса, к которому относится ответ.
4. Таблица Articles (Статьи):
   * + Id: Уникальный идентификатор статьи;
     + Title: Заголовок статьи;
     + Content: Содержание статьи;
     + UserId: Идентификатор пользователя, автора статьи;
     + TitleId: Идентификатор заголовка статьи;
5. Таблица Courses (Курсы):
   * + Id: Уникальный идентификатор курса;
     + Name: Название курса;
6. Таблица Lectures (Лекции):
   * + Id: Уникальный идентификатор лекции;
     + Title: Заголовок лекции;
     + Content: Содержание лекции;
     + LessonId: Идентификатор урока, к которому относится лекция.
7. Таблица Lessons (Уроки):
   * + Id: Уникальный идентификатор урока;
     + Title: Заголовок урока;
     + CourseId: Идентификатор курса, к которому относится урок;
8. Таблица Practices (Практические задания):
   * + Id: Уникальный идентификатор практического задания;
     + Description: Описание практического задания;
     + LessonId: Идентификатор урока, к которому относится задание;
9. Таблица Tests (Тесты):
   * + Id: Уникальный идентификатор теста;
     + Title: Заголовок теста;
     + LessonId: Идентификатор урока, к которому относится тест.
10. Таблица Titles (Заголовки статей):
    * + Id: Уникальный идентификатор заголовка;
      + ArticleId: Идентификатор статьи, к которой относится заголовок.
11. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА

## Разработка уровня Domain

Уровень DOMAIN представляет собой часть приложения, которая содержит основную бизнес-логику и модель предметной области. Он отвечает за описание и управление ключевыми концепциями, правилами и объектами, связанными с предметной областью приложения.

Уровень DOMAIN включает:

* Описание предметной области: Уровень Domain определяет модель предметной области, которая отражает ключевые сущности, их атрибуты, связи и поведение. Здесь определяются классы и интерфейсы, которые представляют бизнес-объекты и их взаимодействие.
* Бизнес-логика: Уровень содержит бизнес-логику приложения, которая определяет правила и операции, связанные с предметной областью. Что включает в себя валидацию данных, расчеты, обработку бизнес-правил, управление состоянием объектов и другие операции, необходимые для достижения целей предметной области.
* Независимость от инфраструктуры: Уровень Domain независим от конкретной реализации инфраструктуры и внешних фреймворков. Он является чистым и независимым от внешних зависимостей, таких как базы данных, веб-сервисы или пользовательский интерфейс. Это позволяет повысить переносимость, тестируемость и гибкость кода.
* Инкапсуляция бизнес-правил: Domain обеспечивает инкапсуляцию бизнес-правил и логики внутри бизнес-объектов. Это позволяет сделать код более понятным, модульным и легко поддерживаемым. Бизнес-правила могут быть проверены и применены в различных слоях приложения без необходимости повторного кодирования.
* Разделение ответственности: позволяет разделить ответственность между различными слоями приложения. Он фокусируется на бизнес-логике и представлении предметной области, в то время как другие слои, такие как слой представления (VIEW) и слой доступа к данным (DATA ACCESS), отвечают за отображение данных и взаимодействие с внешними источниками данных.

Уровень Domain имеет следующую структуру (рис 3.1):

* + - 1. Группа Entity содержит все модели данных, используемые в приложение;
      2. группа Enum, содержит перечисляемые типы, которые используется в разработке: тип элемента, список ролей пользователей, список статус кодов для страниц ответа;
      3. группа Interface содержит в себе список интерфейсов, используемых в программе;
    1. группа Response содержит шаблоны ответов от служб;
    2. группа ViewModel содержит в себе вью модели, которые упрощают передачу данных между слоями приложения. Данная группа также разделяется на подгруппы: статьи, курс, пользователи.

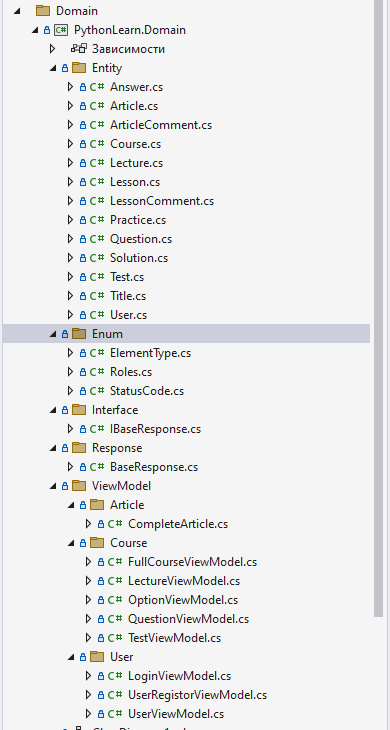


Рисунок 3.1. Структура уровня Domain

Группа Entity

Раздел Entity в уровне Domain представляет собой одну из ключевых составляющих модели предметной области. Он отвечает за определение и представление основных сущностей, объектов или понятий, которые играют важную роль в бизнес-процессах и операциях приложения.

В разделе entity содержаться следующие модели данных:

* Answer – модель содержащая все ответы на тестовые вопросы содержит следующие поля: id, QuestionId – id вопроса к которому относится ответ, Text – текст вопроса, IsCorrect - флаг является ли ответ верным;
* article – модель статьи содержит следующие поля: id, UserId – id пользователя создавшего статью, TitleId – id заголовка статьи, CreatedDate – дата создания статьи, ArticleText – текст статьи, также имеется два виртуальных поля для организации связи один ко многим, TitleId –заголовок, User – пользователь написавший статью;
* сourse представляет модель данных курса. Он содержит информацию Id: уникальный идентификатор курса. Name: название курса. Description: описание курса. Lessons: список уроков, связанных с данным курсом. Поле Lessons является коллекцией уроков, которые связаны с данным курсом;
* lecture представляет модель данных для лекции. Он содержит Id уникальный идентификатор лекции, LessonId идентификатор урока, к которому относится данная лекция, Name название лекции, Text текст лекции, IsCompleted флаг, указывающий, завершена ли лекция;
* lesson представляет модель данных для урока. Он содержит Id уникальный идентификатор урока, CourseId идентификатор курса, к которому относится данный урок, Name название урока, Status статус урока, который указывает на его состояние (например, завершен или в процессе), Course ссылка на объект курса, к которому принадлежит урок, Lectures список лекций, связанных с данным уроком, Tests список тестов, связанных с данным уроком;
* question представляет модель данных для вопроса в тесте. Он содержит информацию Id уникальный идентификатор вопроса, TestId идентификатор теста, к которому относится данный вопрос, Text: текст вопроса, CorrectAnswer правильный ответ на вопрос, Answers коллекция ответов, связанных с данным вопросом, Test ссылка на объект теста, к которому принадлежит вопрос;
* Test представляет модель данных для теста. Он содержит: Id уникальный идентификатор теста, LessonId идентификатор урока, к которому относится данный тест, Name название теста, Questions коллекция вопросов, связанных с данным тестом;
* Title представляет модель данных для заголовка статьи. Он содержит Id уникальный идентификатор заголовка, ArticleId идентификатор статьи, к которой относится данный заголовок, Name название заголовка (не более 20 символов). ShortDescription краткое описание заголовка (не более 300 символов), Article связанная статья
* User представляет модель данных для пользователя в системе. Он содержит Id уникальный идентификатор пользователя, Name имя пользователя (не более 30 символов), SecondName фамилия пользователя (не более 30 символов), BirthDay дата рождения пользователя, Login логин пользователя (не более 30 символов), Email электронная почта пользователя (не более 30 символов), Password пароль пользователя (не более 300 символов), AboutMe информация о пользователе (не более 400 символов), Avatar изображение аватара пользователя (в виде массива байтов) Role роль пользователя, Solutions список решений, связанных с пользователем.

Листинг класса User представлен на рисунке 3.2

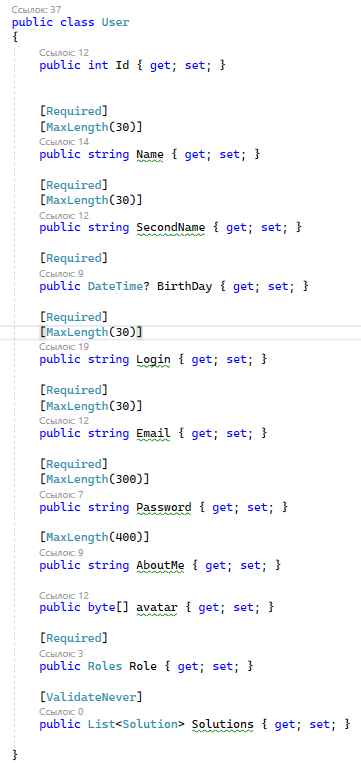


Рисунок 3.2. листинг класса user

Группа enum

В слое домена, перечисления (enum) используются для определения и представления различных константных значений, состояний или вариантов выбора, которые имеют отношение к бизнес-логике и доменной модели приложения. Они обеспечивают ясность, типобезопасность и ограничение допустимых значений для соответствующих полей или свойств классов в домене.

В проекте используются следующие перечисления:

* Roles определяет различные роли пользователей в системе. Каждый элемент в перечислении представляет определенную роль, которая может быть назначена пользователям.
  + Admin этот элемент представляет роль администратора. Администраторы имеют полный доступ и привилегии в системе, включая управление пользователями, настройками и ресурсами.
  + Moderator этот элемент представляет роль модератора. Модераторы имеют расширенные права и возможности в системе, но они ограничены по сравнению с администраторами.
  + User элемент представляет роль обычного пользователя.
* StatusCode (рис 3.3) определяет различные статусные коды, которые могут быть возвращены в результате выполнения операций или обработки запросов в системе. Каждый элемент в перечислении представляет определенный код и связанное с ним описание. StatusCode определяет следующие значения:
  + UserNotFound = 0: Этот элемент представляет статусный код, указывающий, что пользователь не найден;
  + UserAlreadyExists = 1 Этот элемент представляет статусный код, указывающий, что пользователь уже существует;
  + OK = 200: Этот элемент представляет статусный код, указывающий успешное выполнение операции или запроса;
  + Warn = 201 Этот элемент представляет статусный код, указывающий на предупреждение или уведомление;
  + BadRequest = 400: Этот элемент представляет статусный код, указывающий на ошибку в запросе со стороны клиента;
  + NotFound = 404: Этот элемент представляет статусный код, указывающий, что запрашиваемый ресурс не найден;
  + InternalServerError = 500: Этот элемент представляет статусный код, указывающий на внутреннюю ошибку на стороне сервера.

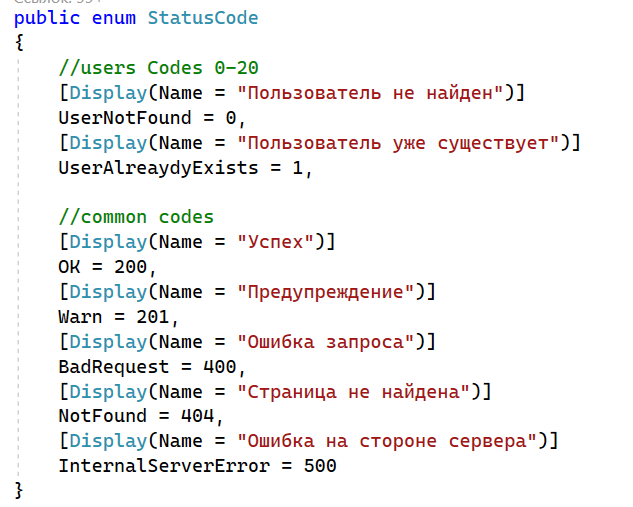


Рисунок 3.3. листинг enum status code

группы interface и baseResponse

Класс BaseResponse предназначен представлять базовую модель ответа, которая используется для возвращения результатов операций или обработки запросов в системе. Он служит в качестве основы для создания конкретных моделей ответа, которые содержат дополнительные поля и свойства, специфичные для каждого типа ответа.

Назначение BaseResponse состоит в следующем:

* Предоставление общей структуры для моделей ответа: BaseResponse определяет базовые свойства, которые часто присутствуют в моделях ответа, такие как статусный код (StatusCode) и сообщение (Message). Это позволяет унифицировать модели ответа и обеспечить их согласованное использование в системе.
* Передача информации о результате операции или запроса: BaseResponse содержит дополнительные поля, которые представляют информацию о результатах операции или запроса. Например, это может быть поле Data, содержащее данные, возвращаемые операцией, или поле Errors, содержащее информацию об ошибках, возникших при обработке запроса.
* Облегчение взаимодействия с другими слоями системы: BaseResponse используется для передачи результатов между различными слоями системы, такими как слой бизнес-логики и слой представления. Это позволяет более гибко управлять информацией о результатах операций и обработке запросов и обеспечивает единообразный способ обмена данными между различными компонентами системы.

Группа interface представляет собой список интерфейсов, применяемых в данном слое.

Далее представлены два листинга IbaseResponse (рис3.4) он определяет интерфейс baseResponse и является шаблоном, что позволяет передавать с помощью него данные любого типа



Рисунок 3.4. листинг IbaseResponse

Сам класс BaseResponse представлен на рисунке 3.5 он просто реализует выше представленный интерфейс

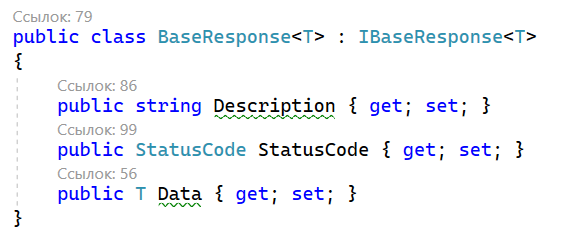


Рисунок 3.5. листинг BaseResponse

группа ViewModel

Группа viewModel используется для определения моделей представления (view models), которые служат для передачи данных между слоем представления (presentation layer) и слоем бизнес-логики (business logic layer) в приложении.

В проекте используются следующие вью модели:

CompleteArticle представляет модель представления для полной статьи. Он содержит следующие поля:

* TitleId: Идентификатор заголовка статьи.
* ArticleId: Идентификатор статьи.
* Name: Название статьи, ограничено максимальной длиной в 100 символов.
* ShortDescription: Краткое описание статьи, ограничено максимальной длиной в 1000 символов.
* ArticleText: Текст статьи. Обязательное поле, используется атрибут [ValidateNever] для игнорирования валидации при передаче данных между слоями.
* CreatedDate: Дата создания статьи. Обязательное поле, используется атрибут [ValidateNever] для игнорирования валидации.
* user: Пользователь, создавший статью.
* UserId: Идентификатор пользователя, создавшего статью.

FullCourseViewModel представляет модель для полного курса. Она содержит следующие поля:

* Name: Название курса.
* Description: Описание курса.
* Lessons: Список уроков, представленных объектами типа Lesson. Используется атрибут [ValidateNever] для игнорирования валидации при передаче данных между слоями.

LectureViewModel представляет модель для лекции. Она содержит следующие поля:

* LessonId: Идентификатор урока, к которому относится лекция.
* Name: Название лекции.
* Text: Текст лекции. Используется атрибут [ValidateNever] для игнорирования валидации при передаче данных между слоями.

OptionViewModel: представляет модель для варианта ответа. Содержит следующие поля:

* Text: Текст варианта ответа.
* IsCorrect: Флаг, указывающий, является ли данный вариант ответа правильным.

QuestionViewModel: представляет модель для вопроса. Содержит следующие поля:

* Text: Текст вопроса.
* Options: Список моделей вариантов ответа (OptionViewModel).

TestViewModel: представляет модель для теста. Содержит следующие поля:

* Lesson: Идентификатор урока, к которому относится тест.
* Name: Название теста.
* Questions: Список моделей представления вопросов (QuestionViewModel).

LoginViewModel: представляет модель для входа пользователя. Содержит следующие поля:

* UserName: Логин пользователя.
* Password: Пароль пользователя.

UserRegistorViewModel: представляет модель для регистрации пользователя. Основана на IdentityUser. Содержит следующие поля:

* Name: Имя пользователя.
* SecondName: Фамилия пользователя.
* BirthDay: Дата рождения пользователя.
* Login: Логин пользователя.
* Email: Email пользователя.
* Password: Пароль пользователя.
* ConfirmPassword: Подтверждение пароля пользователя.
* AboutMe: Описание пользователя.
* Avatar: Изображение пользователя (тип IFormFile).

UserViewModel: представляет модель для пользователя. Содержит следующие поля:

* Id: Идентификатор пользователя.
* Name: Имя пользователя.
* SecondName: Фамилия пользователя.
* BirthDay: Дата рождения пользователя.
* Login: Логин пользователя.
* Email: Email пользователя.
* AboutMe: Описание пользователя.
* Avatar: Изображение пользователя (тип IFormFile).
* Image: Бинарные данные изображения пользователя.

## Разработка уровня DAL (Domain access level)

DAL (Data Access Layer) - это слой доступа к данным в архитектуре приложения. Его основная цель - обеспечить абстракцию и управление операциями с базой данных.

Назначение слоя DAL включает:

* Управление подключением к базе данных: DAL отвечает за установку соединения с базой данных и управление жизненным циклом соединений. Это включает открытие и закрытие соединений, управление транзакциями и обработку ошибок связанных с базой данных.
* Выполнение запросов к базе данных: DAL предоставляет методы для выполнения CRUD-операций (создание, чтение, обновление и удаление данных) и других запросов к базе данных. Он обеспечивает конвертацию и маппинг данных из базы данных в объекты приложения и наоборот.
* Управление схемой базы данных: DAL предоставляет функциональность для создания, изменения и удаления таблиц, индексов, хранимых процедур и других объектов базы данных. Это позволяет разработчикам управлять структурой базы данных с помощью кода.
* Обеспечение безопасности данных: DAL обеспечивает защиту данных путем использования параметризованных запросов и предотвращения атак вроде SQL-инъекций. Он также может обеспечивать контроль доступа к данным и применение правил безопасности.
* Абстракция от конкретной базы данных: DAL позволяет абстрагироваться от конкретной реализации базы данных. Это позволяет разработчикам менять или переключаться между различными системами управления базами данных (например, SQL Server, MySQL, PostgreSQL) без необходимости изменения кода приложения.

Слой DAL помогает создавать модульные и легко поддерживаемые приложения, разделяя логику работы с данными от бизнес-логики и пользовательского интерфейса. Он способствует повышению производительности, безопасности и переносимости приложения.

В слое DAL содержаться следующие группы классов:

interface – в этой группе содержатся все интерфейсы используемые на слое DAL;

migrations – служебная папка сюда помещаются классы которые используются для переноса модели данных в базу данных, в большинстве случаев данные классы генерируются автоматически;

modelConfigure – тут размещаются классы, которые позволяют настраивать модели данных, производить мапинг одних моделей в другие, а также задавать начальные значения для БД;

repositories – группа хранящая классы репозитории необходимые для доступа к таблицам БД;

services – служебные классы используемые в различных частях программы;

группа interface

В проекте используется паттерн под название dependency injection, что дает следующие преимущества:

* Уменьшение связанности: DI позволяет уменьшить связанность между компонентами приложения. Объекты зависят от абстракций (интерфейсов), а не от конкретных реализаций. Это упрощает внесение изменений в код и замену зависимых компонентов без необходимости изменения кода, который использует эти компоненты.
* Улучшение тестируемости: DI упрощает юнит-тестирование компонентов приложения. Зависимости могут быть заменены заглушками или моками, что позволяет создавать изолированные тесты для каждого компонента. Таким образом, можно проверить функциональность компонента независимо от его зависимостей.
* Повышение гибкости и переиспользуемости: DI позволяет легко заменять зависимые компоненты и конфигурировать приложение без изменения его основного кода. Зависимости могут быть настроены и внедрены в приложение внешними компонентами или конфигурацией.
* Централизованное управление зависимостями: DI-контейнеры предоставляют механизм для централизованного управления зависимостями в приложении. Они автоматически создают и внедряют зависимости при создании объектов и позволяют легко настроить зависимости и их жизненный цикл.

Для реализации данного паттерна необходимо использовать интерфейсы, все интерфейсы создание в библиотеке DAL храниться в этой группе и имеют следующую структуру:

Интерфейс IbaseRepository (рис 3.6) хранит общие свойства, которые необходимо реализовать во всех классах репозиториях, эти свойства представляют собой CRUD операции.

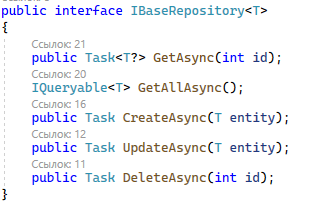


Рисунок 3.6. листинг IBaseRepository

Интерфейсы, которые представляют все сущности БД (IUserRepository, ILectureRepository, IQuestionRepository…) все они наследуются от интерфейса IBaseRepository и содержат дополнительные методы специфичные только для данной модели, например репозиторий IUserRepository (рис 3.7) содержит дополнительные методы позволяющие искать пользователей по имени или фамилии.

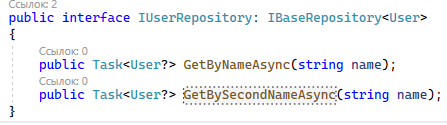


Рисунок 3.7. листинг IUserRepository

Интерфейс IUnitOfWork рис 3.8. В этой библиотеке также реализован паттерн Unit of Work который способствует улучшению организации кода, повышению производительности и обеспечению целостности данных в приложении.



Рисунок 3.8. листинг IUnitOfWork

В нем определяются все объекты репозитории как свойства, что позволяет получить к ним доступ, здесь также определены два метода Save и Dispose. Метод Save используется для сохранения всех накопленных изменений в базе данных. В рамках Unit of Work все операции с базой данных (создание, обновление, удаление) накапливаются и не применяются немедленно. Вызов метода Save выполняет фактическое сохранение изменений в базе данных, выполняя соответствующие операции (INSERT, UPDATE, DELETE) для каждого измененного объекта. Метод Dispose используется для освобождения ресурсов, занятых Unit of Work. В Unit of Work используются ресурсы базы данных или другие ресурсы, которые требуют явного освобождения, чтобы избежать утечки памяти или других проблем.

класс ApplicationDbContext

ApplicationDbContext (рис 3.9) является производным классом от DbContext и представляет контекст базы данных для приложения. Он предоставляет доступ к базе данных и определяет набор таблиц (наборы сущностей), которые могут быть использованы в приложении.

ApplicationDbContext позволяет установить соединения с базой данных, а также предоставляет настройки и параметры для подключения к конкретной базе данных. Он использует провайдер данных для установления соединения и взаимодействия. Он также определяет наборов сущностей и содержит свойства, которые доступные в базе данных. Каждое свойство представляет таблицу базы данных, и через него можно выполнять операции CRUD (создание, чтение, обновление, удаление) над сущностями. Основной задачей данного является предоставление доступа к данным приложения, он служит в качестве основного интерфейса для получения доступа к данным приложения и предоставляет методы и свойства для выполнения запросов к базе данных, фильтрации, сортировки и других операций.



Рисунок 3.9. листинг AplicationDbContext

Он содержит два конструктора, один по умолчанию, а второй позволяет передать настройки для взаимодействия с БД. В данном классе также есть метод OnModelCreating который позволяет использовать мапинг моделей.

классы репозитории

Для реализации доступа к данным были созданы классы репозитории они имеют следующее назначение:

* Изолирование доступа к данным: Репозитории предоставляют единый интерфейс для доступа к данным из различных источников, таких как база данных, веб-службы, файлы и т.д. Они абстрагируют детали взаимодействия с конкретными источниками данных и предоставляют стандартизированный способ работы с данными в рамках приложения.
* Облегчение тестирования: Репозитории могут быть легко заменены заглушками или фиктивными реализациями для проведения модульных тестов. Это позволяет изолировать компоненты приложения, которые зависят от доступа к данным, и проводить тестирование без фактического взаимодействия с реальными источниками данных.
* Сокрытие деталей работы с данными: Репозитории скрывают сложность и детали взаимодействия с базой данных или другими источниками данных. Они предоставляют простой и понятный интерфейс для выполнения операций CRUD (создание, чтение, обновление, удаление) и других операций работы с данными. Это облегчает разработку и поддержку приложения, поскольку другие компоненты могут использовать репозитории без необходимости знать подробности внутренней реализации доступа к данным.
* Централизация логики доступа к данным: Репозитории объединяют логику доступа к данным в одном месте. Они предоставляют методы для выполнения типичных операций с данными, таких как получение списка объектов, поиск по определенным критериям, фильтрация, сортировка и другие. Это упрощает управление и поддержку кода, так как логика доступа к данным сосредоточена в репозиториях, а не разбросана по различным частям приложения.
* Повышение переносимости: Репозитории позволяют сделать слой доступа к данным независимым от конкретной реализации источника данных. Это дает возможность легко изменять или заменять источник данных без изменения других частей приложения. Например, если приложение изначально использует базу данных, но в дальнейшем решено перейти к использованию веб-служб, то достаточно заменить реализацию репозиториев, не затрагивая остальной код приложения.

Все классы репозитории имеют однотипную логику (рис 3.10) и в основном реализуют CRUD операции:

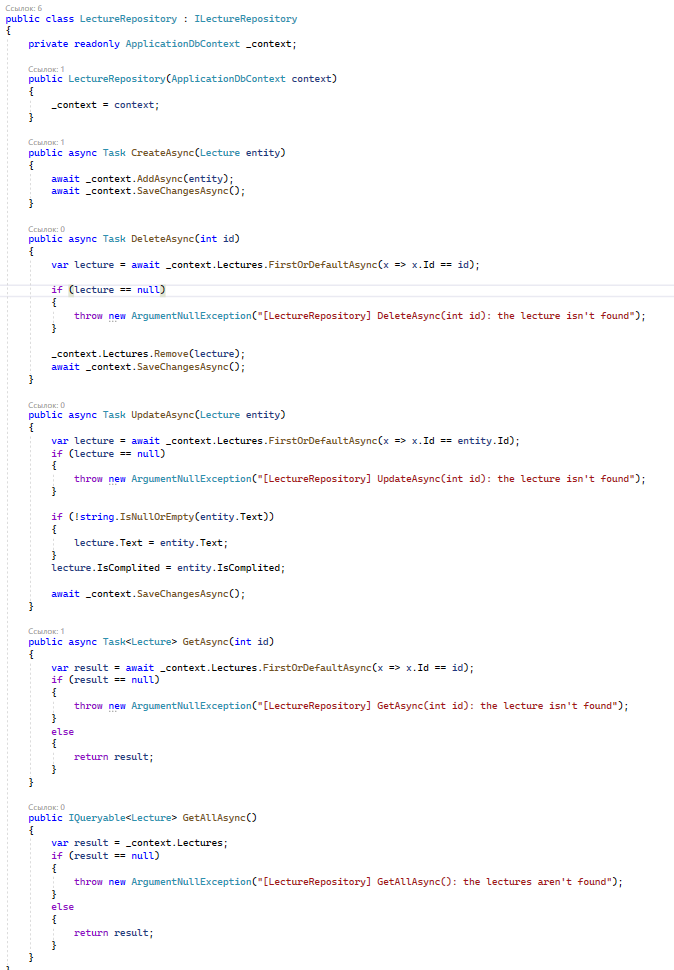


Рисунок 3.10. листинг репозитория Lecture

Репозиторий состоит из следующий частей:

Конструктор, в который передается контекст для доступа к базе данных;

метод CreateAsync в данный метод передаётся объект модели данных после чего происходит его проверка и добавление в БД.

метод DeleteAsync в данном методе происходит удаление объекта из базы данных, метод принимает на входи id объекта, который необходимо удалить, после чего ищет его в БД, если находит, то удаляет если нет, то кидает исключение;

метод UpdateAsync позволяет обновить данные в таблице, на вход принимает объект с новыми полями, после чего объект с таким id ищется в БД, если найден, то у него обновляются все поля, если нет, то кидается исключение;

метод GetAsync позволяет получить объект по его id, он на вход принимает id объекта, который надо найти, после ищет его в БД в случае успеха возвращает его, если объект не найден, то кидается исключение;

метод GetAllAsync позволяет вернуть все объекты из данной таблицы возвращаются объекты в виде коллекции IQueryable, что позволяет работать с результатами метода с помощью LINQ.

Служебные классы

CreateHash (рис 3.11) позволяет создавать MD5 хэш-значения для входной строки. MD5 (Message Digest Algorithm 5) - это криптографический хэш-алгоритм, который преобразует произвольный входной текст в уникальную строку фиксированной длины.

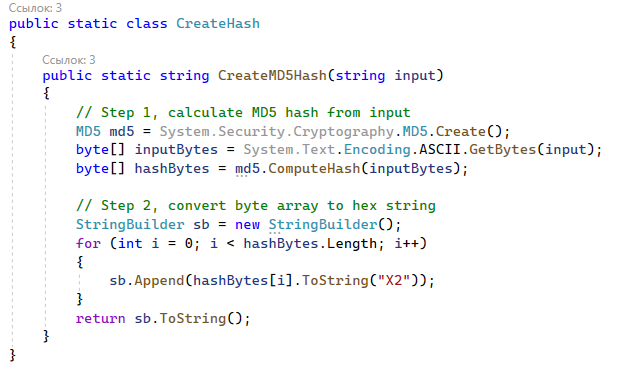


Рисунок 3.11. листинг класса CreateHash

Принцип работы класса CreateHash заключается в создании экземпляра класса MD5 из пространства имен System.Security.Cryptography.MD5, который будет использоваться для вычисления хэш-значения. Входная строка преобразуется в массив байтов с использованием кодировки ASCII. Вычисляется хэш-значение из массива байтов с помощью метода ComputeHash объекта md5. Полученное хэш-значение представляется в виде строки в шестнадцатеричном формате, используя объект StringBuilder. Каждый байт хэш-значения преобразуется в двузначное шестнадцатеричное число и добавляется к строке. Итоговая строка, представляющая MD5 хэш-значение входной строки, возвращается из метода.

Класс ImageProcess предоставляет статические методы для обработки изображений, в частности, для преобразования изображений между различными форматами и типами данных.

Метод ConverToFormFile преобразует массив байтов изображения (srcImage) в объект типа IFormFile, который является интерфейсом для представления файлов, отправляемых через HTTP-запросы. Создается экземпляр класса MemoryStream, который представляет поток памяти для хранения массива байтов изображения. Используя переданный массив байтов (srcImage), данные копируются в поток памяти. Создается объект FormFile с использованием созданного потока памяти, длины массива байтов и других параметров (имя файла и имя поля формы). Возвращается объект FormFile, приведенный к типу IFormFile.

Метод GetByteArrayFromIFormFile извлекает массив байтов из объекта IFormFile, представляющего загруженный файл. Создается экземпляр класса MemoryStream, который будет использоваться для сохранения данных файла. Выполняется копирование данных из объекта IFormFile (file) в поток памяти. Возвращается массив байтов, полученный из потока памяти с помощью метода ToArray().

## Разработка уровня Service

Слой сервиса (Service) в архитектуре приложения отвечает за обработку бизнес-логики и предоставление функциональности, необходимой для взаимодействия с другими слоями. Он служит прослойкой между слоем представления и слоем доступа к данным, обеспечивая абстракцию и инкапсуляцию бизнес-логики.

Назначение слоя сервиса:

* Реализация бизнес-логики: Слой сервиса содержит логику, которая связана с бизнес-правилами и операциями, которые должны быть выполнены над данными приложения. Он описывает, как данные должны быть обработаны, проверены, модифицированы и возвращены в слой представления.
* Координация работы различных компонентов: Слой сервиса может координировать работу различных компонентов и слоев приложения. Он может вызывать методы слоя доступа к данным для получения и сохранения данных, обрабатывать запросы от слоя представления и возвращать соответствующие результаты.
* Управление транзакциями: Слой сервиса может быть ответственным за управление транзакциями и обеспечение целостности данных при выполнении операций, которые требуют атомарности и консистентности.
* Предоставление интерфейса для внешних систем: Слой сервиса может предоставлять интерфейс для взаимодействия с внешними системами или API. Он может обрабатывать запросы и формировать соответствующие ответы в формате, необходимом для внешних систем.
* Управление зависимостями: Слой сервиса может быть ответственным за управление зависимостями между различными компонентами и слоями приложения. Он может использовать принцип внедрения зависимостей (Dependency Injection) для внедрения зависимостей и обеспечения легкой замены или модификации компонентов.

Общий принцип слоя сервиса состоит в том, чтобы отделить бизнес-логику и операции от слоя представления и слоя доступа к данным, обеспечивая более чистую и модульную структуру приложения. Это упрощает тестирование, повторное использование и обновление компонентов приложения.

Данный слой включает две группы классов interfaces которая содержит список интерфейсов для классов, необходимых для DI паттерна и группу implementation которая содержит классы реализующие интерфейсы.

AccountService

Класс AccountService, является частью слоя сервиса в приложении и отвечает за управление учетными записями пользователей. Он решает следующие задачи:

1. Регистрация новых пользователей: AccountService предоставляет методы для создания новых учетных записей пользователей. Также включает в себя проверку уникальности имени пользователя, валидацию пароля и других обязательных полей, сохранение учетной записи в хранилище данных;
2. Аутентификация пользователей: AccountService содержит методы для аутентификации пользователей;
3. Управление профилем пользователя: AccountService предоставляет методы для управления профилем пользователя, такие как обновление информации о пользователе, смена пароля, загрузка аватара и другие операции, связанные с профилем пользователя;
4. Управление правами доступа: AccountService обеспечивает управление правами доступа пользователя, такие как назначение ролей и разрешений, проверка прав доступа к определенным ресурсам или действиям в приложении.
5. Взаимодействие с хранилищем данных: AccountService использует методы и интерфейсы слоя доступа к данным (DAL) для выполнения операций чтения и записи данных, связанных с учетными записями пользователей, в хранилище данных.

Методы класса Accoutservice

Метод Authentificate (рис 3.12)

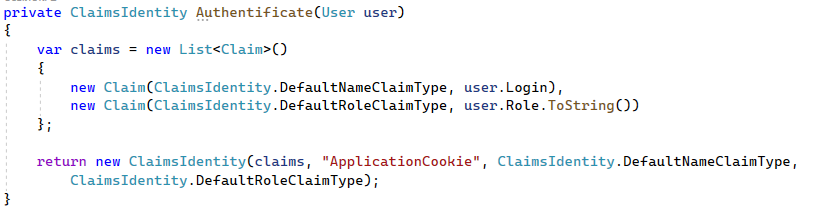


Рисунок 3.12. листинг метода Authentificate

Метод Authenticate принимает объект User в качестве аргумента и возвращает объект ClaimsIdentity. Его назначение заключается в создании идентификации (identity) пользователя на основе его данных, которые будет использоваться для аутентификации и авторизации пользователя в приложении. Внутри метода создается список утверждений (claims), которые представляют информацию о пользователе список содержит следующие параметры:

* ClaimsIdentity.DefaultNameClaimType: Утверждение, содержащее логин пользователя. Оно создается с использованием DefaultNameClaimType, который представляет стандартный тип утверждения для имени пользователя. Значение логина пользователя передается в качестве значения утверждения.
* ClaimsIdentity.DefaultRoleClaimType: Утверждение, содержащее роль пользователя. Оно создается с использованием DefaultRoleClaimType, который представляет стандартный тип утверждения для роли пользователя. Значение роли пользователя передается в качестве значения утверждения. Роль пользователя представлена в виде строки, полученной из перечисления user.Role и преобразованной в строку методом ToString().

Затем созданные утверждения добавляются в список claims. И наконец, создается новый объект ClaimsIdentity с использованием списка claims, а также указанием типа аутентификации ("ApplicationCookie"), типа утверждения для имени пользователя и типа утверждения для роли пользователя. Полученный объект ClaimsIdentity представляет идентификацию пользователя, содержащую его логин и роль.

Метод Register (рис 3.13) служит для регистрации нового пользователя в системе. Принимая объект UserRegistorViewModel в качестве аргумента

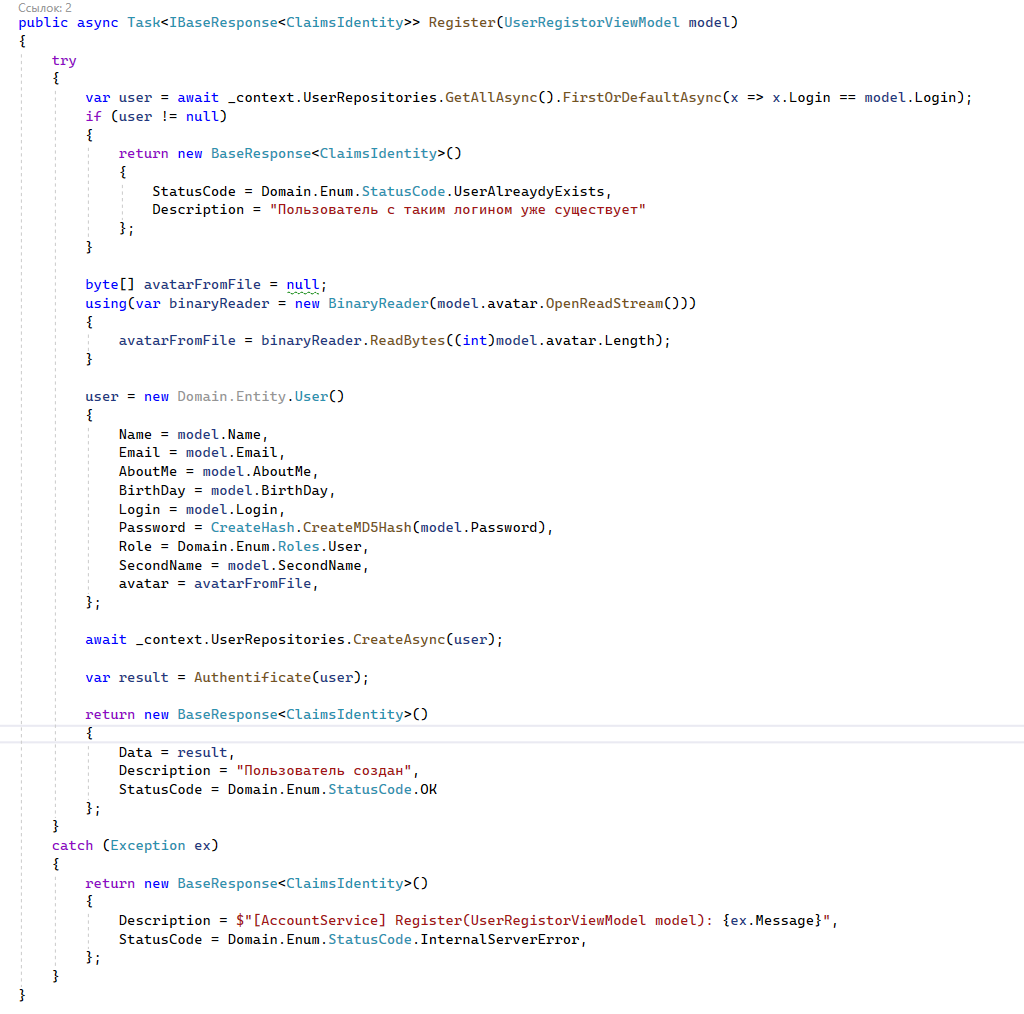


Рисунок 3.13. листинг метода Register

Метод Register выполняет следующие шаги:

* Проверяет, существует ли пользователь с указанным логином (model.Login) в базе данных. Если такой пользователь уже существует, возвращается объект BaseResponse<ClaimsIdentity>, содержащий соответствующий статус код (StatusCode.UserAlreaydyExists) и описание ошибки.
* Если пользователь с указанным логином не найден, производится чтение байтов из загруженного файла аватара (model.avatar). Для этого создается экземпляр BinaryReader, и производится чтение байтов с помощью метода ReadBytes. Полученные байты аватара сохраняются в переменную avatarFromFile.
* Создается новый объект User, заполняются его свойства с использованием значений из model и других данных, таких как хэшированный пароль (CreateHash.CreateMD5Hash(model.Password)) и роль пользователя (Domain.Enum.Roles.User).
* Вызывается метод CreateAsync репозитория \_context.UserRepositories для сохранения созданного пользователя в базе данных.
* Создается идентификация (identity) пользователя с помощью метода Authentificate(user). Результат идентификации присваивается свойству Data объекта BaseResponse<ClaimsIdentity>.
* Возвращается объект BaseResponse<ClaimsIdentity>, содержащий статус код StatusCode.OK, описание успешной операции и данные идентификации пользователя.
* В случае возникновения исключения, метод возвращает объект BaseResponse<ClaimsIdentity> с соответствующим статус кодом (StatusCode.InternalServerError) и описание ошибки, содержащее информацию об исключении (ex.Message).

Метод Login (рис 3.14) предназначен для аутентификации пользователя на основе предоставленных учетных данных. Он принимает объект LoginViewModel в качестве аргумента.

****

Рисунок 3.13. листинг метода Login

Метод Login выполняет следующие шаги:

* Выполняется запрос к базе данных с использованием репозитория \_context.UserRepositories для поиска пользователя с указанным логином (model.UserName). Результат запроса сохраняется в переменную user.
* Если пользователь не найден (переменная user равна null), возвращается объект BaseResponse<ClaimsIdentity> с соответствующим статус кодом (StatusCode.BadRequest) и описанием ошибки.
* Если найден пользователь, производится сравнение хэшированного пароля пользователя (user.Password) с хэшированным паролем, предоставленным в модели (CreateHash.CreateMD5Hash(model.Password)). Если пароли не совпадают, возвращается объект BaseResponse<ClaimsIdentity> с соответствующим статус кодом и описанием ошибки.
* Если аутентификация прошла успешно, вызывается метод Authentificate(user) для создания идентификации (identity) пользователя.
* Возвращается объект BaseResponse<ClaimsIdentity>, содержащий статус код StatusCode.OK, описание успешной аутентификации и данные идентификации пользователя.
* В случае возникновения исключения, метод возвращает объект BaseResponse<ClaimsIdentity> с соответствующим статус кодом (StatusCode.InternalServerError) и описание ошибки, содержащее информацию об исключении (ex.Message).

UserService

Класс UserService представляет сервисную реализацию интерфейса IUserService. Его основное назначение заключается в предоставлении операций CRUD (Create, Read, Update, Delete) для работы с сущностями пользователей.

Ниже представлено назначение каждого метода в классе UserService:

* CreateUser(UserViewModel user, byte[] imageData): Метод создает нового пользователя на основе предоставленной модели UserViewModel и данных аватарки. Он сохраняет пользователя в базе данных и возвращает асинхронный ответ, содержащий булево значение, указывающее на успешность операции.
* GetUsersAsync(): Метод получает список всех пользователей из базы данных и возвращает стандартный ответ, содержащий список пользователей. Если список пуст, возвращается соответствующий статус код и описание.
* GetUserAsync(int id): Метод получает пользователя по указанному идентификатору id из базы данных и возвращает стандартный ответ, содержащий найденного пользователя. Если пользователь не найден, возвращается соответствующий статус код и описание.
* GetUserByNameAsync(string name): Метод получает список пользователей с указанным именем name из базы данных и возвращает стандартный ответ, содержащий список пользователей. Если список пуст, возвращается соответствующий статус код и описание.
* GetUserBySecondNameAsync(string secondName): Метод получает список пользователей с указанной фамилией secondName из базы данных и возвращает стандартный ответ, содержащий список пользователей. Если список пуст, возвращается соответствующий статус код и описание.
* EditAsync(int id, UserViewModel model): Метод изменяет поля пользователя с указанным идентификатором id на основе предоставленной модели UserViewModel. Он обновляет данные пользователя в базе данных и возвращает стандартный ответ, содержащий измененного пользователя. Если пользователь не найден, возвращается соответствующий статус код и описание.
* DeleteUserAsync(int id): Метод удаляет пользователя с указанным идентификатором id из базы данных. Он возвращает стандартный ответ, содержащий булево значение, указывающее на успешность операции удаления. Если пользователь не найден, возвращается соответствующий статус код и описание.

Класс UserService использует внедрение зависимостей для получения репозитория пользователей (IUnitOfWork), необходимого для взаимодействия с базой данных. Он предоставляет удобный способ для выполнения операций создания, чтения, обновления и удаления пользователей в системе.

ArticleService

Класс ArticleService представляет собой сервисный компонент, который обеспечивает операции, связанные с управлением статьями и заголовками. Он реализует интерфейс IArticleService, определяющий набор методов для работы со статьями и заголовками. Класс ArticleService предоставляет функционал для создания, чтения, обновления и удаления статей и заголовков в системе. Он является промежуточным слоем между контроллером (или другими клиентскими компонентами) и репозиториями данных.

Основные задачи ArticleService включают:

* Создание статьи или заголовка и сохранение их в базе данных.
* Получение информации о конкретной статье или заголовке по их идентификаторам.
* Получение списка всех статей или заголовков.
* Изменение данных статьи или заголовка и их сохранение.
* Удаление статьи или заголовка по их идентификаторам.

Класс ArticleService использует внедрение зависимостей (dependency injection) для получения экземпляра IUnitOfWork, который представляет контекст работы с данными. Это позволяет ему взаимодействовать с репозиториями статей и заголовков, предоставляемыми IUnitOfWork, для выполнения операций с данными.

Методы ArticleService

Метод CreateArticle принимает в качестве параметра объект Article, содержащий информацию о новой статье, такую как заголовок, содержание, автор и другие связанные данные. После создает новый экземпляр статьи на основе переданных данных и получает экземпляр IUnitOfWork через внедрение зависимостей. с помощью IUnitOfWork вызывает метод репозитория для сохранения созданной статьи в базе данных. Возвращает результат операции в виде объекта BaseResponse<Article>, содержащего информацию о статусе операции, описании и самой созданной статье.

Метод CreateTitle принимает в качестве параметра объект Title, содержащий информацию о новом заголовке, такую как текст заголовка и другие связанные данные. Создает новый экземпляр заголовка на основе переданных данных.   
С помощью IUnitOfWork вызывает метод репозитория для сохранения созданного заголовка в базе данных. В конце возвращает результат операции в виде объекта BaseResponse<Title>, содержащего информацию о статусе операции, описании и самом созданном заголовке.

Метод CreateArticleWithTitle принимает в качестве параметра объект CompleteArticle, содержащий информацию о новом заголовке и связанной с ним статье, такие как текст заголовка, содержание статьи, автор и другие связанные данные. Создает новый экземпляр заголовка на основе переданных данных. Создает новый экземпляр статьи на основе переданных данных. Связывает созданный заголовок со статьей, устанавливая ссылку на заголовок в поле статьи, и сохраняет связь между ними. С помощью IUnitOfWork вызывает методы репозиториев для сохранения созданного заголовка и статьи в базе данных. Возвращает результат операции в виде объекта BaseResponse<CompleteArticle>, содержащего информацию о статусе операции, описании и самом созданном заголовке и статье.

Метод GetArticleAsync принимает в качестве параметра идентификатор статьи. С помощью IUnitOfWork вызывает метод репозитория для получения статьи по указанному идентификатору из базы данных. Возвращает результат операции в виде объекта BaseResponse<Article>, содержащего информацию о статусе операции, описании и полученной статье.

Метод GetCompleteArticle принимает в качестве параметра идентификатор статьи. С помощью IUnitOfWork вызывает метод репозитория для получения полной информации о статье (включая заголовок) по указанному идентификатору из базы данных. Возвращает результат операции в виде объекта BaseResponse<CompleteArticle>, содержащего информацию о статусе операции, описании и полученной полной статье.

Метод GetArticlesAsync получает экземпляр, с помощью IUnitOfWork вызывает метод репозитория для получения списка всех статей из базы данных. Возвращает результат операции в виде объекта BaseResponse<List<Article>>, содержащего информацию о статусе операции, описании и списке полученных статей.

Метод GetTitleAsync принимает в качестве параметра идентификатор заголовка. С помощью IUnitOfWork вызывает метод репозитория для получения заголовка по указанному идентификатору из базы данных. Возвращает результат операции в виде объекта BaseResponse<Title>, содержащего информацию о статусе операции, описании и полученном заголовке.

Метод GetTitlesAsync с помощью IUnitOfWork вызывает метод репозитория для получения списка всех заголовков из базы данных. возвращает результат операции в виде объекта BaseResponse<List<Title>>, содержащего информацию о статусе операции, описании и списке полученных заголовков.

Метод EditArticleAsync принимает в качестве параметров идентификатор статьи и объект Article, содержащий обновленные данные статьи. С помощью IUnitOfWork вызывает метод репозитория для получения статьи по указанному идентификатору из базы данных. Если статья с указанным идентификатором существует, производит обновление данных статьи на основе переданного объекта Article. Сохраняет изменения в базе данных с помощью IUnitOfWork. Возвращает результат операции в виде объекта BaseResponse, содержащего информацию о статусе операции и описании.

Метод EditTitleAsync принимает в качестве параметров идентификатор заголовка и объект Title, содержащий обновленные данные заголовка. С помощью IUnitOfWork вызывает метод репозитория для получения заголовка по указанному идентификатору из базы данных. Если заголовок с указанным идентификатором существует, производит обновление данных заголовка на основе переданного объекта Title. Сохраняет изменения в базе данных с помощью IUnitOfWork. Возвращает результат операции в виде объекта BaseResponse, содержащего информацию о статусе операции и описании.

Метод DeleteArticleAsync принимает в качестве параметра идентификатор статьи, которую необходимо удалить. С помощью IUnitOfWork вызывает метод репозитория для получения статьи по указанному идентификатору из базы данных. Если статья с указанным идентификатором существует, производит удаление статьи. Сохраняет изменения в базе данных с помощью IUnitOfWork. Возвращает результат операции в виде объекта BaseResponse, содержащего информацию о статусе операции и описании.

Метод DeleteTitleAsync принимает в качестве параметра идентификатор заголовка, который необходимо удалить. С помощью IUnitOfWork вызывает метод репозитория для получения заголовка по указанному идентификатору из базы данных. Если заголовок с указанным идентификатором существует, производит удаление заголовка. Сохраняет изменения в базе данных с помощью IUnitOfWork. Возвращает результат операции в виде объекта BaseResponse, содержащего информацию о статусе операции и описании.

CourseService

Класс CourseService представляет сервис для работы с курсами и связанными с ними сущностями. Он реализует интерфейс ICourseService и содержит ряд методов для получения, создания и обновления курсов, лекций, тестов и других связанных объектов.

Конструктор класса принимает экземпляр IUnitOfWork через внедрение зависимостей, который используется для взаимодействия с базой данных.

Данный класс содержит следующие методы:

GetCourse(int id) получает курс по указанному идентификатору. Возвращает результат операции в виде объекта BaseResponse<Course>, содержащего информацию о статусе операции и сам курс.

GetLecture(int id) получает лекцию по указанному идентификатору. Возвращает результат операции в виде объекта BaseResponse<Lecture>, содержащего информацию о статусе операции и саму лекцию.

GetFullCourse(int id) получает полную информацию о курсе (включая лекции) по указанному идентификатору. возвращает результат операции в виде объекта BaseResponse<FullCourseViewModel>, содержащего информацию о статусе операции и полную информацию о курсе.

GetCourses() получает список всех курсов. возвращает результат операции в виде объекта BaseResponse<List<Course>>, содержащего информацию о статусе операции и список курсов.

GetLessons() получает список всех уроков. возвращает результат операции в виде объекта BaseResponse<List<Lesson>>, содержащего информацию о статусе операции и список уроков.

CreateCourse(Course course) создает новый курс на основе переданного объекта Course. Возвращает результат операции в виде объекта BaseResponse<bool>, содержащего информацию о статусе операции.

CreateLesson(Lesson lesson) создает новый урок на основе переданного объекта Lesson. Возвращает результат операции в виде объекта BaseResponse<bool>, содержащего информацию о статусе операции.

CreateLecture(LectureViewModel lecture) создает новую лекцию на основе переданного объекта LectureViewModel. Возвращает результат операции в виде объекта BaseResponse<bool>, содержащего информацию о статусе операции.

CreateTest(Test test) создает новый тест на основе переданного объекта Test. Возвращает результат операции в виде объекта BaseResponse<bool>, содержащего информацию о статусе операции.

CreateQuest(Question question) создает новый вопрос на основе переданного объекта Question. Возвращает результат операции в виде объекта BaseResponse<bool>, содержащего информацию о статусе операции.

CreateAnswer(Answer answer) создает новый ответ на основе переданного объекта Answer. Возвращает результат операции в виде объекта BaseResponse<bool>, содержащего информацию о статусе операции.

Каждый метод внутри CourseService выполняет соответствующие операции с использованием объектов репозиториев из IUnitOfWork и возвращает результат операции в виде объекта BaseResponse<T>, где T - тип результата операции.

## Разработка уровня Controller

Уровень контроллера (Controller) в архитектуре приложения отвечает за обработку запросов от клиента и взаимодействие с другими компонентами системы для выполнения требуемых операций. Контроллеры определяют точки входа для клиентских запросов и обеспечивают маршрутизацию запросов к соответствующим действиям (actions) или методам.

Задачи контроллера:

* Обработка запросов: Контроллеры принимают входящие HTTP-запросы от клиента и обрабатывают их. Они извлекают данные из запроса (параметры, заголовки, тело запроса и т.д.) и определяют, какое действие должно быть выполнено.
* Взаимодействие с моделью: Контроллеры используют соответствующие сервисы, репозитории или другие компоненты, чтобы получить или изменить данные модели, необходимые для выполнения запросов. Они вызывают соответствующие методы или операции этих компонентов, чтобы получить требуемые данные или выполнить требуемые операции.
* Подготовка данных для представления: Контроллеры получают данные из модели или других источников, их обрабатывают и подготавливают для передачи представлениям (views) или клиенту. Это может включать создание моделей представления или упаковку данных в определенный формат (например, JSON или XML) для передачи клиенту.
* Вызов представлений: Контроллеры вызывают соответствующие представления для отображения данных или визуализации результатов операций. Они передают необходимые данные представлениям в виде моделей представления или других объектов, чтобы представления могли правильно отобразить данные для клиента.
* Обработка ошибок и исключений: Контроллеры обрабатывают ошибки и исключения, возникающие в процессе выполнения операций. Они могут выполнять обработку и логирование ошибок, отправлять соответствующие сообщения об ошибке клиенту или принимать решение о дальнейшей обработке ошибок.
* Управление жизненным циклом запроса: Контроллеры обычно управляют жизненным циклом запроса, выполняя инициализацию и очистку ресурсов перед и после выполнения операций. Они могут выполнять дополнительные действия до и после обработки запроса, такие как аутентификация, авторизация, журналирование и другие аспекты обработки запроса.

HomeController

Данный контроллер (рис 3.14) обрабатывает запросы, связанные с домашней страницей (Index) и ошибками (Error). Он наследуется от базового класса Controller, который предоставляет базовую функциональность для контроллеров в ASP.NET Core.



Рисунок 3.14 Листинг homeController

В этом контроллере имеются следующие методы:

Index(UserViewModel viewModel): Этот метод обрабатывает GET-запросы на домашнюю страницу. Он принимает объект UserViewModel в качестве параметра и передает его в представление для отображения. Представление, связанное с этим методом, будет отображать информацию, содержащуюся в UserViewModel.

Error(): Этот метод обрабатывает GET-запросы на страницу ошибки. Он создает новый объект ErrorViewModel, заполняет его данными об ошибке и передает его в представление, связанное с ошибкой, для отображения.

Кроме того, контроллер имеет конструктор, в котором внедряется зависимость от ILogger<HomeController>. Это позволяет использовать функциональность журналирования внутри контроллера, например, для регистрации сообщений, ошибок или другой информации о действиях контроллера.

Общий функционал контроллера HomeController состоит в обработке запросов, взаимодействии с представлениями и обеспечении соответствующего отображения данных на клиентской стороне.

UserController

Данный контроллер обрабатывает запросы, связанные с пользователями. Он наследуется от базового класса Controller, который предоставляет базовую функциональность для контроллеров в ASP.NET Core.

Контроллер содержит следующие методы:

[HttpGet] Register(): Этот метод обрабатывает GET-запрос на страницу регистрации пользователей. Он возвращает представление, связанное с регистрацией, которое будет отображаться клиенту.

[HttpGet] Save(int id): Этот метод обрабатывает GET-запрос на страницу редактирования учетной записи пользователя. Если идентификатор пользователя равен 0, то возвращается представление для создания нового пользователя. В противном случае, выполняется запрос к сервису (\_service.GetUserAsync(id)) для получения пользователя по идентификатору. Если пользователь найден, то возвращается представление для редактирования пользователя. В противном случае, происходит перенаправление на страницу ошибки.

[HttpPost] [Authorize(Roles = "Admin")] Save(UserViewModel model): Этот метод обрабатывает POST-запрос на редактирование пользователя. Если модель данных UserViewModel проходит валидацию (ModelState.IsValid), то выполняется создание нового пользователя или редактирование существующего пользователя через вызов соответствующих методов сервиса.

[HttpPost] Register(string name, string secondName, DateTime Birthday, string login, string email, string password, string aboutMe, byte[] avatar): Этот метод обрабатывает POST-запрос на регистрацию пользователя. Он принимает параметры, связанные с данными пользователя, и возвращает перенаправление на страницу "RegisterSuccess".

[HttpGet] RegisterSuccess(): Этот метод обрабатывает GET-запрос на страницу успешной регистрации пользователя. Он возвращает представление, связанное с успешной регистрацией, которое будет отображаться клиенту.

[HttpGet] GetUsers(): Этот метод обрабатывает GET-запрос на получение списка пользователей. Он выполняет запрос к сервису (\_service.GetUsersAsync()) для получения списка пользователей и возвращает представление, связанное с отображением списка пользователей.

Контроллер также имеет конструктор, в котором внедряется зависимость от IUserService. Это позволяет использовать функциональность, предоставляемую сервисом пользователей (\_service) для выполнения операций с пользователями, таких как создание, редактирование и получение пользователей.

Общий функционал контроллера UserController состоит в обработке запросов, связанных с пользователями, взаимодействии с представлениями и взаимодействии с сервисом пользователей для выполнения соответствующих операций.

AccountController

Данный контроллер обрабатывает запросы, связанные с учетными записями пользователей (регистрация, вход, выход, профиль).

Метода содержащиеся в этом контроллере:

[HttpGet] Register(): Этот метод обрабатывает GET-запрос на отображение страницы регистрации. Он возвращает представление "Register", которое будет отображаться клиенту.

[HttpPost] Register(UserRegistorViewModel model): Этот метод обрабатывает POST-запрос с данными регистрации пользователя. Если модель данных UserRegistorViewModel проходит валидацию (ModelState.IsValid), то вызывается сервис регистрации (\_service.Register(model)), который выполняет регистрацию пользователя. Если регистрация прошла успешно, происходит вход пользователя с использованием аутентификации с помощью куки, и происходит перенаправление на домашнюю страницу. Если возникает ошибка регистрации, ошибка добавляется в ModelState и возвращается представление регистрации с ошибками.

[HttpGet] Login(): Этот метод обрабатывает GET-запрос на отображение страницы входа. Он возвращает представление "Login", которое будет отображаться клиенту.

[HttpPost] Login(LoginViewModel model): Этот метод обрабатывает POST-запрос с данными входа пользователя. Если модель данных LoginViewModel проходит валидацию (ModelState.IsValid), то вызывается сервис входа (\_service.Login(model)), который выполняет аутентификацию пользователя. Если аутентификация прошла успешно, происходит вход пользователя с использованием аутентификации с помощью куки. Затем происходит поиск информации о пользователе в сервисе пользователей и создается соответствующая модель представления UserViewModel. Далее происходит перенаправление на домашнюю страницу с передачей модели представления. Если возникает ошибка входа, ошибка добавляется в ModelState и возвращается представление входа с ошибками.

[ValidateAntiForgeryToken] Logout(): Этот метод обрабатывает GET-запрос на выход из учетной записи пользователя. Он выполняет выход пользователя путем удаления аутентификационных куки и перенаправляет на домашнюю страницу.

[HttpGet] Profile(): Этот метод обрабатывает GET-запрос на отображение страницы профиля пользователя. Если пользователь аутентифицирован (User.Identity.IsAuthenticated), то происходит поиск информации о пользователе в сервисе пользователей (\_userService.GetUsersAsync()) и создается соответствующая модель представления UserViewModel. Затем возвращается представление "Profile" с передачей модели представления. Если пользователь не аутентифицирован, возвращается представление "Profile" без модели данных.

[HttpPost] Profile(UserViewModel model): Этот метод обрабатывает POST-запрос с данными изменения профиля пользователя. Сначала извлекается информация о текущем пользователе из сервиса пользователей (\_userService.GetUsersAsync()), а затем обновляются соответствующие поля модели представления model. Если модель данных проходит валидацию (ModelState.IsValid), вызывается сервис изменения учетной записи (\_service.EditAccount(model)), который выполняет обновление профиля пользователя. Если обновление прошло успешно, происходит перенаправление на домашнюю страницу с передачей логина пользователя в качестве параметра. Если возникает ошибка при обновлении, ошибка добавляется в ModelState и возвращается представление "Profile" с ошибками.

В контроллере также присутствуют конструктор и приватные поля, которые используются для внедрения зависимостей сервисов, таких как IAccountService, IUserService и IMapper. Это позволяет контроллеру использовать функциональность, предоставляемую этими сервисами, для выполнения операций с учетными записями и пользователями.

CourseController

Данный контроллер обрабатывает запросы, связанные с курсами и лекциями.

Методы, содержащиеся в этом контроллере:

[HttpGet] Main(int id): Этот метод обрабатывает GET-запрос на отображение основной информации о курсе. Он вызывает сервис (\_service.GetFullCourse(id)) для получения полной информации о курсе по заданному идентификатору id. Затем возвращается представление "Main" с передачей данных о курсе (course.Data) в качестве модели.

[HttpGet] Lecture(int id): Этот метод обрабатывает GET-запрос на отображение информации о лекции. Он вызывает сервис (\_service.GetLecture(id)) для получения информации о лекции по заданному идентификатору id. Затем создается модель представления LectureViewModel с данными о названии лекции и тексте лекции, и возвращается представление "Lecture" с передачей модели представления.

[HttpGet] CreateTest(): Этот метод обрабатывает GET-запрос на отображение страницы создания теста. Он вызывает (\_service.GetLessons()) для получения списка уроков. Затем создается SelectList с данными об уроках и передается в ViewBag.Lessons. Возвращается представление "CreateTest".

[HttpGet] CreateLecture(): Этот метод обрабатывает GET-запрос на отображение страницы создания лекции. Он вызывает сервис (\_service.GetLessons()) для получения списка уроков. Затем создается SelectList с данными об уроках и передается в ViewBag.Lessons. Возвращается представление "CreateLecture".

[HttpPost] CreateTest([FromBody] TestViewModel test): Этот метод обрабатывает POST-запрос на создание теста. Он принимает модель представления TestViewModel в качестве параметра, которая содержит информацию о тесте и вопросах. Затем создается сущность Test и сохраняется в сервисе курсов (\_service.CreateTest(testEntity)). Затем происходит итерация по вопросам и ответам, и каждый вопрос и ответ сохраняется в сервисе курсов (\_service.CreateQuest(questionEntity) и \_service.CreateAnswer(optionEntity)). После завершения сохранения возвращается ответ Ok().

[HttpPost] CreateLecture(LectureViewModel lecture): Этот метод обрабатывает POST-запрос на создание лекции. Он принимает модель представления LectureViewModel в качестве параметра, которая содержит информацию о лекции. Затем извлекается текст лекции из HttpContext.Request.Form, присваивается свойству Text модели lecture. Если модель проходит валидацию (ModelState.IsValid), вызывается сервис курса (\_service.CreateLecture(lecture)), который выполняет создание лекции. Если создание прошло успешно, происходит перенаправление на домашнюю страницу. Если возникает ошибка при создании, ошибка добавляется в ModelState и возвращается представление "CreateLecture" с ошибками.

[HttpPost] UplodImage(List<IFormFile> files, string folder): Этот метод обрабатывает POST-запрос на загрузку изображения. Он принимает список файлов files и путь folder. Если folder не задан, генерируется случайное имя папки. Затем создается путь к папке и, если папка не существует, она создается. Затем происходит итерация по каждому файлу в списке Request.Form.Files, файл сохраняется на сервере, и возвращается URL загруженного файла в формате JSON.

1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 4.1. История изменений

Таблица 4.1 История изменений

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Версия | Автор | Описание изменений |
| 07.02.2023 | 1.0 | Чистяков К.В. | Создание моделей для БД и их экспорт в таблицы |
| 10.02.2023 | 1.1 | Чистяков К.В. | Создание репозиториев и unit of work классов |
| 15.02.2023 | 1.2 | Чистяков К.В. | Верстка главной страницы сайта. |
| 25.02.2023 | 1.3.1 | Чистяков К.В. | Реализация службы создания пользователей. |
| 26.02.2023 | 1.3.2 | Чистяков К.В. | Коррекция моделей пользователей, создание вью моделей для них |
| 01.03.2023 | 1.4.1 | Чистяков К.В. | Создание службы для регистрации и авторизации пользователей |
| 04.03.2023 | 1.4.2 | Чистяков К.В. | Верстка страниц регистрации, авторизации и редактирования профиля |
| 10.03.2023 | 1.5.1 | Чистяков К.В. | Создание службы для управления блогом |
| 12.03.2023 | 1.5.2 | Чистяков К.В. | Корректировка моделей статей, заголовков и создание вью моделей |
| 13.03.2023 | 1.5.3 | Чистяков К.В. | Добавление методов в службу создания статей |
| 16.03.2023 | 1.5.4 | Чистяков К.В. | Верстка страницы блога и страницы статей |
| 17.03.2023 | 1.5.5 | Чистяков К.В. | Настройка ckeditor на странице создания статей |
| 25.03.2023 | 1.6.1 | Чистяков К.В. | Создание службы для управления курсом и реализация методов отображения курса, урока, лекции, теста |
| 27.03.2023 | 1.6.2 | Чистяков К.В. | Создание вью моделей, для полного курса, урока, лекции, теста |
| 29.03.2023 | 1.6.3 | Чистяков К.В. | Верстка страниц для отображения целого курса, лекции и теста |
| 04.04.2023 | 1.6.4 | Чистяков К.В. | Создание методов для добавления, курсов, уроков, лекций, тестов |
| 10.04.2023 | 1.6.5 | Чистяков К.В. | Верстка страниц с добавлением, курсов, уроков, лекций, тестов |
| 15.04.2023 | 1.7 | Чистяков К.В. | Редактирование web страниц |

## 4. 2. Терминология тестирования программного обеспечения

Вот список терминов и их определений, которые используются далее в документе и в общем тестировании:

1. Производительность: Количество операций, которые выполняются за определенный период времени, например, N операций за M часов;
2. Тестирование стабильности: это форма тестирования, при которой система подвергается средней нагрузке в течение продолжительного времени. Она позволяет выявить проблемы, связанные с утечками памяти и некорректными настройками программного обеспечения;
3. Архитектура "клиент-сервер": это сетевая архитектура, в которой задачи или сетевая нагрузка распределяются между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами;
4. Нагрузка: общее количество операций, выполняемых на общем ресурсе в единицу времени (например, транзакции в секунду или запросы в секунду);
5. Виртуальный пользователь: Программный процесс, который циклически выполняет моделируемые операции в системе;
6. Стратегия тестирования: план проведения работ по тестированию системы или ее модуля, который учитывает специфику функциональности и зависимости от других компонентов системы и платформы;
7. Позитивное тестирование: тестирование, основанное на сценариях, которые соответствуют нормальному поведению системы;
8. Негативное тестирование: тестирование, основанное на сценариях, которые соответствуют ненормальному или внештатному поведению системы;
9. Адаптивность: способность сайта или приложения "подстраиваться" под различные технические условия, например, под разные размеры экрана устройств;
10. Тестирование методом "открытого ящика": Метод тестирования программного продукта, при котором тестировщик имеет доступ к внутренней структуре, устройству и реализации системы;

## 4.3. Стратегия тестирования программного обеспечения

Стратегия тестирования программного обеспечения заключается в использовании метода тестирования “открытого ящика” для анализа и оценки защиты информационных потоков данных, а также отслежка выходных в входных данных на прецедент ошибки, пустого значения.

## 4.4 Определение объектов тестирования

Таблица 4.2. Определение объектов тестирования

| **Тесты** | **Объект тестирования** |
| --- | --- |
| **Виды тестирования** | |
| Функциональное тестирование | * работа основной страницы и навигации; * работа системы регистрации и авторизации; * работа системы редактирования профиля; * работа блога; * работа системы создания статей; * работа системы создания лекций и тестов; * работа системы прохождения обучения; |
| Тестирование производительности | * скорость перехода по формам; * скорость сохранения данных в БД; * скорость отображения информации из БД; |
| Нагрузочное тестирование | * загрузка страниц; * загрузка тестов; * загрузка изображений в БД; * загрузка изображений из БД; * загрузка изображений на сервер; |
| Тестирование совместимости | * корректное отображение в различных браузерах; * корректное отображение при различных разрешениях; * корректное отображение на различных устройствах; |
| **Различные типы тестов** | |
| Позитивные тесты | * регистрация пользователей ввод корректного email; * создание статьи с уникальными именем; * создание лекции для существующего урока; * создание теста для существующего урока; |
| Негативные тесты | * регистрация пользователей ввод не корректного email; * создание статьи с не уникальными именем; * создание лекции для не существующего урока; * создание теста для не существующего урока; |
| **Уровни тестирования** | |
| Модульное тестирование | * проверка, что при передаче валидной модели в метод Register() и успешной регистрации пользователя, происходит перенаправление на домашнюю страницу; * проверка, что при передаче невалидной модели в метод Login(), в модель добавляются ошибки валидации; * проверка, что при вызове метода Profile() для неаутентифицированного пользователя, возвращается представление "Profile"; * проверка, что при вызове метода Lecture() с определенным идентификатором лекции, возвращается соответствующее представление с данными лекции; * проверка, что при вызове метода CreateLecture() для неаутентифицированного пользователя или пользователя без роли "Admin", происходит перенаправление на домашнюю страницу; * проверка, что при передаче списка файлов и имени папки в метод UplodImage(), файлы успешно сохраняются на сервере и возвращается URL-адрес загруженного файла; |
| Интеграционное тестирование | * проверка, что при отправке POST-запроса на Register с невалидными данными пользователя, возвращается соответствующее представление с ошибками валидации; * проверка, что при отправке POST-запроса на Login с неправильными учетными данными пользователя, возвращается соответствующее представление с сообщением об ошибке; * проверка, что при отправке GET-запроса на Main с определенным идентификатором курса, возвращается соответствующее представление с данными курса; * проверка, что при отправке POST-запроса на CreateTest с валидной моделью теста, происходит успешное создание теста и перенаправление на домашнюю страницу; * проверка, что при отправке POST-запроса на CreateLecture с невалидной моделью лекции, возвращается соответствующее представление с ошибками валидации; * проверка, что при отправке POST-запроса на UplodImage с файлами и указанием папки, файлы успешно загружаются на сервер и возвращается URL-адрес загруженного файла; |
| Системное тестирование | * проверка, что при успешной регистрации нового пользователя, он добавляется в базу данных и может успешно войти в систему; * проверка, что при выходе из аккаунта, пользователь больше не имеет доступа к защищенным ресурсам и перенаправляется на страницу входа; * проверка, что при открытии страницы курса, пользователь может видеть основную информацию о курсе, включая его название, описание и автора; * проверка, что пользователь может просматривать содержимое лекции, включая текст, изображения и другие медиа-элементы; * проверка, что администратор может создавать новые лекции для определенного курса и добавлять содержимое, включая текст, изображения и другие медиа-элементы; |

## 4.5 Архитектура тестируемой системы

Архитектура клиент-серверной системы:

Оборудование:

клиент: персональные компьютеры, ноутбуки, мобильные устройства;

сервер: физический сервер, который обрабатывает запросы от клиентов.

Операционная система:

клиент: Windows, Linux, мобильные операционные системы (iOS, Android).

Сервер: Windows Server.

Тип тестирования:

Требования к тестируемому окружению:

* оборудование среднего сегмента рынка;
* версия операционной системы выше Windows 8;
* используемая база данных MS SQL Server.

## 4. 6. Описание процесса тестирования

Таблица 4.3 Тестовые сценарии с указанием конкретных тестовых наборов

| **Название тестового сценария** | **Тестовый набор** | **Ожидаемый результат** | **Актуальный результат** | **Результат тестирования** |
| --- | --- | --- | --- | --- |
| Регистрация нового пользователя | Ввод корректных данных для регистрации | Пользователь успешно зарегистрирован | Пользователь успешно зарегистрирован | Пройдено |
| Вход зарегистрированного пользователя | Ввод правильных учетных данных | Пользователь успешно вошел в систему | Пользователь успешно вошел в систему | Пройдено |
| Вход незарегистрированного пользователя | Ввод неправильных учетных данных | Система отображает сообщение об ошибке входа | Система отображает сообщение об ошибке входа | Пройдено |
| Создание нового задания | Ввод корректных данных для задания | Задание успешно создано | Задание успешно создано | Пройдено |
| Просмотр списка заданий | Переход на страницу списка заданий | Отображается список всех доступных заданий | Отображается список всех доступных заданий | Пройдено |
| Завершение задания | Отметка задания как завершенного | Задание отмечено как завершенное | Задание отмечено как завершенное | Пройдено |
| Проверка показателей производительности | Загрузка системы с максимальной нагрузкой | Система должна поддерживать требуемый уровень производительности | Система поддерживает требуемый уровень производительности | Пройдено |
| Тестирование стабильности | Запуск системы на протяжении 48 часов средней нагрузки | Не должно быть утечек памяти или некорректных настроек ПО | Отсутствие утечек памяти или некорректных настроек ПО | Пройдено |
| Редактирование профиля пользователя | Внесение изменений в профиль пользователя | Изменения успешно сохранены | Изменения успешно сохранены | Пройдено |
| Удаление задания | Выбор задания для удаления | Задание успешно удалено из системы | Задание успешно удалено из системы | Пройдено |
| Обработка ошибок | Вызов ошибочного сценария | Система обрабатывает ошибку и выводит соответствующее сообщение | Система обрабатывает ошибку и выводит соответствующее сообщение | Пройдено |
| Проверка безопасности | Попытка несанкционированного доступа или взлома | Система успешно предотвращает несанкционированный доступ и взлом | Система успешно предотвращает несанкционированный доступ и взлом | Пройдено |

1. ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

**5.1. Определение защищаемых информационных активов.**

Ценные информационные активы в системе включают данные о студентах, преподавателях. Кроме того, важными активами являются настройки и конфигурации системы, а также программное обеспечение, необходимое для ее функционирования.

Для обеспечения защиты этих активов от несанкционированного доступа и возможных угроз безопасности необходимо принимать соответствующие меры безопасности. Эти меры включают в себя:

1. Установка паролей и ограничений доступа: необходимо устанавливать сильные пароли и механизмы аутентификации для контроля доступа к системе и различным функциям. это поможет предотвратить несанкционированное использование и доступ к ценным данным.
2. Резервное копирование данных: Регулярное создание резервных копий информационных активов поможет обеспечить их сохранность в случае возникновения сбоев, атак или несчастных случаев. Важно также проверять и восстанавливать данные с резервных копий, чтобы убедиться в их целостности.
3. Использование шифрования: шифрование данных является важным средством защиты конфиденциальной информации. Важные активы, такие как личные данные студентов и преподавателей, могут быть зашифрованы, чтобы предотвратить несанкционированное раскрытие при несанкционированном доступе.
4. Регулярное обновление программного обеспечения и антивирусной защиты: Системное программное обеспечение и антивирусные программы должны быть регулярно обновляться, чтобы устранить известные уязвимости и предотвратить возможные атаки вредоносного программного обеспечения.

Все эти меры безопасности помогут обеспечить надежную защиту защищаемых информационных активов в автоматизированной системе учета образовательных процессов.

**5.2. Вид и содержание персональных данных**

Данный сервис содержит в себе разнообразные личные данные пользователей, включая:

* Полное имя и контактные данные (адрес электронной почты, номер телефона)
* Учетные данные для аутентификации (логин и пароль)
* Информацию о уровне образования и квалификации участников
* Историю участия в проектах
* Результаты тестирования и оценки по проектам
* Сведения о месте работы или учебе участников

Обработка личных данных в системе должна соответствовать требованиям законодательства о защите персональных данных. Оператор системы обязан обеспечивать надлежащую защиту личных данных, получать согласие пользователей на обработку и контролировать соблюдение их прав на доступ, изменение, удаление и блокирование своих личных данных.

Для шифрования паролей используются алгоритмы RSA и MD5 HASH. Личные данные хранятся в базе данных в зашифрованном виде, а ключи для расшифровки доступны только пользователям.

**5.3. Модель угроз ПДн**

По мере завершения проекта по защите персональных данных и со временем требования к модели угроз могут меняться, поэтому важно регулярно обновлять модель угроз и принимать соответствующие меры для защиты персональных данных.

После внедрения проекта могут появиться новые угрозы, связанные, например, с развитием технологий или изменением внутренней и внешней среды. Поэтому модель угроз должна регулярно обновляться, чтобы включать новые угрозы и оценивать их влияние на информационную систему.

В процессе использования информационной системы может измениться вероятность возникновения угроз и их последствия. Например, обнаружение уязвимостей в используемом программном обеспечении может повысить вероятность угрозы. Поэтому модель угроз должна регулярно обновляться и пересматриваться с учетом новой информации.

Со временем уязвимости информационной системы могут изменяться, поэтому модель угроз должна регулярно обновляться для обнаружения новых уязвимостей и принятия соответствующих мер по их устранению.

В процессе использования информационной системы могут возникать новые требования к защите персональных данных. Поэтому модель угроз должна регулярно обновляться, чтобы разработать новые меры по защите персональных данных и улучшить существующие меры.

Модель угроз должна регулярно обновляться, чтобы учитывать изменения в информационной системе и окружающей среде. При обновлении модели угроз необходимо анализировать эффективность принятых мер по защите персональных данных и принимать дополнительные меры при необходимости.

В целом, модель угроз персональных данных должна регулярно обновляться и приспосабливаться к изменениям в информационной системе и окружающей среде. Это позволит эффективно защищать персональные данные и минимизировать риски утечки или несанкционированного доступа к ним.

**5.4. Сценарии угроз**

Таблица 5.1 Сценарии угроз

| УБИ | Наименование угрозы | Актуальность угрозы | Потенциал нарушителя | Тактика | Основные  техники |
| --- | --- | --- | --- | --- | --- |
| 1 | Физический доступ к системе | Высокая | Внешний нарушитель | Физическое проникновение в помещение | Взлом замков, использование поддельных идентификационных карт |
| 2 | Вредоносное ПО | Высокая | Внешний и внутренний нарушитель | Заражение системы через вредоносные программы | Вирусы, троянские программы, шпионское ПО |
| 3 | Фишинг | Средняя | Внешний нарушитель | Обман пользователя с целью получения конфиденциальной информации | Подделка электронных писем, сайтов, социальная инженерия |
| 4 | Социальная инженерия | Средняя | Внешний и внутренний нарушитель | Манипуляция людьми для получения несанкционированного доступа | Обман, подкуп, вымогательство |
| 5 | DDoS-атака | Высокая | Внешний нарушитель | Перегрузка системы путем массового запроса | Ботнеты, отказ в обслуживании (DoS), отраженные атаки |
| 6 | Внутренний доступ к системе | Средняя | Внутренний нарушитель | Несанкционированный доступ к данным и ресурсам системы | Нарушение политики безопасности, использование привилегированных учетных записей |
| 7 | Утечка информации | Высокая | Внутренний и внешний нарушитель | Несанкционированное раскрытие конфиденциальной информации | Внутренние журналы, несанкционированный доступ к данным |
| 8 | Физическое повреждение оборудования | Средняя | Внешний и внутренний нарушитель | Повреждение или уничтожение аппаратных средств | Саботаж, вандализм, пожар |
| 9 | Отказ в обслуживании | Высокая | Внешний нарушитель | Перегрузка ресурсов для прекращения работы системы | Перегрузка сетевых ресурсов, атака на протоколы связи |
| 10 | Внутренний шпионаж | Средняя | Внутренний нарушитель | Получение и передача конфиденциальной информации третьим лицам | Подслушивание, кража данных, использование собственных привилегий |

**5.5.Организация защиты данных в проекте**

данной программе организация защиты данных основана на следующих принципах:

* Принцип минимизации данных: Мы собираем и обрабатываем только необходимую информацию, чтобы уменьшить риски утечки данных и неправомерного использования.
* Принцип доступности: Мы обеспечиваем доступ к данным только для уполномоченных пользователей, чтобы гарантировать конфиденциальность информации.
* Принцип конфиденциальности: Мы защищаем информацию от несанкционированного доступа, чтения и раскрытия.
* Принцип целостности: Мы защищаем информацию от изменений или уничтожения без разрешения пользователя или системы.
* Принцип аутентификации и авторизации: Мы проверяем личность пользователя при входе в систему и устанавливаем его права доступа к данным.
* Принцип шифрования: Мы защищаем данные, шифруя их перед передачей по открытым каналам связи.
* Принцип резервного копирования: Мы создаем резервные копии данных, чтобы защитить их от потери или повреждения.
* Принцип мониторинга и аудита: Мы контролируем доступ к данным и системе, а также регистрируем действия пользователей для обеспечения безопасности.
* Принцип постоянного совершенствования: Мы стремимся обеспечить стабильность и развитие системы защиты данных, адаптируя ее к новым угрозам и рискам в соответствии с современными стандартами и технологиями.

**5.6. Защита программного продукта**

Для обеспечения безопасности программы необходимо принять ряд мер, направленных на защиту данных. Вот некоторые варианты мер безопасности:

Усиление защиты паролей: Использование сложных паролей для доступа к системе, требование периодической смены паролей, запрет использования одного и того же пароля для разных пользователей.

Контроль доступа: Определение уровней доступа для различных пользователей и установление прав доступа к функциональным возможностям системы.

Шифрование данных: Применение современных методов шифрования для защиты данных при их передаче и хранении, предотвращение несанкционированного доступа.

Регулярное создание резервных копий данных: Создание и хранение резервных копий данных в безопасном месте, обеспечение возможности восстановления данных в случае потери или повреждения.

Физическая безопасность: Обеспечение физической безопасности серверов и оборудования, ограничение физического доступа к ним.

Аудит безопасности: Проведение регулярных аудитов безопасности системы для выявления и устранения уязвимостей и недостатков.

Обновление программного обеспечения: Регулярное обновление программного обеспечения, включая патчи безопасности, для устранения известных уязвимостей.

Установка брандмауэра и антивирусного программного обеспечения: Защита системы с помощью установки и настройки брандмауэра и антивирусного ПО для обнаружения и предотвращения вредоносных программ и атак.

Управление уязвимостями: Регулярный анализ и устранение выявленных уязвимостей в системе для предотвращения возможных атак.

1. ТЕХНИЧЕСКАЯ ДОКУМЕНТАЦИЯ НА ПРОГРАММНЫЙ ПРОДУКТ

**6.1. Руководство пользователя.**

Руководство пользователя является частью пакета эксплуатационной документации и имеет основную цель предоставить пользователям необходимую информацию для самостоятельной работы с автоматизированной системой.

Руководство пользователя должно быть написано доступным языком и предоставлять информацию, необходимую для успешного использования программы. Оно должно быть структурировано и организовано таким образом, чтобы пользователь мог легко найти нужную информацию и решить свои задачи при работе с программой.

**Назначение системы**

Назначение данного заключается в предоставлении пользователю возможности освоить язык программирования Python и приобрести навыки разработки программных приложений на данном языке. Сервис предоставляет обучающие материалы, интерактивные уроки, практические задания и другие ресурсы, которые помогают пользователям изучить основные концепции, синтаксис и возможности Python.

Основные цели образовательного сервиса по Python включают:

1. Обучение основам Python: Сервис предлагает курсы и уроки, которые помогают новичкам овладеть основными концепциями и принципами языка программирования Python. Это включает изучение синтаксиса, переменных, операторов, функций, структур данных и других основных элементов языка.
2. Разработка навыков программирования: Сервис предоставляет задачи и практические упражнения, которые помогают пользователям развивать свои навыки программирования на Python. Это может включать создание простых программ, написание функций, работу с файлами, использование библиотек и другие практические задания.
3. Расширение знаний и применение Python: Сервис предоставляет продвинутые курсы и материалы, которые помогают пользователям углубить свои знания Python и применить их для разработки различных типов приложений. Это может включать создание веб-приложений, анализ данных, машинное обучение, автоматизацию задач и другие области применения Python.
4. Поддержка сообщества и обмен опытом: Сервис может предоставлять форумы, чаты или другие средства общения, которые позволяют пользователям обмениваться опытом, задавать вопросы и получать поддержку от опытных разработчиков или преподавателей. Это способствует формированию сообщества, где пользователи могут взаимодействовать и учиться друг у друга.

**Краткое описание возможностей**

Разработанный программный продукт определяет следующие возможности для пользователя: «Ученик»

- просмотр лекций;

- прохождение тестов;

- доступ к статьям;

**Условия применения системы**

Требования к аппаратному обеспечению:

-Минимальная скорость 1.0 Ггц, 2 ядра, 4 потоков

-Минимальный размер оперативной памяти 4 Гб

-Минимальный объем жесткого диска 50 Гб

-Версия установленной ОС windows 8

Пользователи должны обладать навыками работы с версиями операционной системы windows 8

**Подготовка системы к работе**

Пошаговая инструкция для запуска приложения:

Шаг 1. Зайти на веб страницу приложения;

Шаг 2. авторизоваться;

**Описание операций**

1. Регистрация:
   * Пользователь должен открыть сайт или приложение образовательного сервиса.
   * На главной странице пользователь должен найти раздел "Регистрация" или "Создать аккаунт" и перейти в него.
   * Пользователь должен заполнить регистрационную форму, предоставив необходимую информацию, такую как имя, электронную почту и пароль.
   * После заполнения формы пользователь должен подтвердить свою регистрацию, следуя инструкциям, которые могут включать подтверждение по электронной почте или ввод кода подтверждения.
   * После успешной регистрации пользователь получит доступ к своему аккаунту и сможет войти в систему.
2. Авторизация:
   * Пользователь должен открыть сайт или приложение образовательного сервиса.
   * На главной странице пользователь должен найти раздел "Вход" или "Авторизация" и перейти в него.
   * Пользователь должен ввести свои учетные данные, такие как электронную почту и пароль, которые он указал при регистрации.
   * После ввода учетных данных пользователь должен нажать кнопку "Войти" или аналогичную, чтобы войти в систему.
   * Если предоставленные учетные данные верны, пользователь будет успешно авторизован и получит доступ к своему аккаунту.
3. Просмотр статей на сайте:
   * После успешной авторизации пользователь должен найти раздел или меню, где перечислены статьи или ресурсы доступные для просмотра.
   * Пользователь может использовать навигационные элементы, фильтры или поиск, чтобы найти интересующие его статьи.
   * При выборе статьи пользователь должен щелкнуть по заголовку или превью, чтобы открыть полный контент статьи.
   * Пользователь может просматривать содержимое статьи, прокручивая его вверх и вниз, и использовать предоставленные ссылки или элементы навигации для перемещения по статье.
4. Изучение лекций:
   * После успешной авторизации пользователь должен перейти на страницу с лекциями, которая может быть доступна из главного меню или раздела "Лекции".
   * Пользователь может выбрать нужную лекцию из списка доступных лекций.
   * Лекция может содержать текстовую информацию, примеры кода, графики, видеоматериалы и другие образовательные ресурсы.
   * Пользователь может изучать лекцию, просматривая ее содержимое, читая текст, анализируя примеры кода или просматривая визуальные материалы.
   * Пользователь может перемещаться по лекции, используя ссылки, кнопки навигации или предоставленные таблицы содержания.
5. Прохождение тестов:
   * После успешной авторизации пользователь должен перейти на страницу с тестами, которая может быть доступна из главного меню или раздела "Тесты".
   * Пользователь может выбрать нужный тест из списка доступных тестов.
   * Тест может состоять из вопросов с выбором ответа, задач, кейсов или других форматов.
   * Пользователь должен читать вопросы и выбирать или вводить ответы в соответствии с требованиями теста.
   * После ответа на все вопросы пользователь может подтвердить свои ответы и получить результаты теста, которые могут включать оценку, обратную связь или детальную статистику.

**6.2.Руководство администратора**

Руководство администратора – это составная часть эксплуатационной Руководство администратора является частью эксплуатационной документации и предназначено для ответственных пользователей, которым поручается управление функционированием системы. Оно предоставляет необходимую информацию и инструкции по выполнению операций, связанных с поддержкой работы системы, управлением правами доступа, редактированием данных и исправлением ошибок.

Это руководство адресовано администраторам, чья задача заключается в обеспечении правильного функционирования системы. Администратор является пользователем системы, но обладает специальными привилегиями, позволяющими ему осуществлять управление и контроль за системой.

**Обязанности и задачи администратора**

* Осуществлять контроль над назначением ролей пользователями, чтобы обеспечить правильное распределение доступа и функциональных возможностей.
* Анализировать требования пользователей и оптимизировать работу системы с целью повышения ее эффективности и удобства использования.
* Предоставлять поддержку пользователей и решать возникающие у них проблемы, связанные с работой программного продукта.
* Создавать резервные копии данных и осуществлять их восстановление в случае сбоев или потери информации.
* Осуществлять мониторинг работы системы с целью обнаружения возможных проблем и неполадок, а также принимать меры по их устранению.

**Обслуживание системы**

-назначение пользователям прав доступа;

-учетные записи пользователей и управление ими;

-настройка параметров работы системы;

-загрузка и выгрузка данных;

**Ответственность администратора**

Администратор несет ответственность за:

-за невыполнение своих должностных обязанностей

-за несоблюдение инструкций, приказов, и распоряжений по сохранению конфиденциальной информации

-за нарушение правил охраны труда, противопожарной безопасности.

6.3 Определение качества программного продукта по метрикам

|  |  |  |
| --- | --- | --- |
| Количество строк  программы | 35625 | |
| Строк в каждом сегменте | 5 937 | |
| Количество сегментов  программы | Количество комментариев |  |
| 1 сегмент | 400 | -1 |
| 2 сегмент | 1500 | 1 |
| 3 сегмент | 300 | -1 |
| 4 сегмент | 1500 | 1 |
| 5 сегмент | 525 | -1 |
| 6 сегмент | 1120 | -1 |

1. МЕРОПРИЯТИЯ ПО ТЕХНИКЕ БЕЗОПАСНОСТИ И ОХРАНЫ ТРУДА

При разработке программного продукта необходимо принимать меры по обеспечению безопасности и охране труда сотрудников, чтобы защитить их здоровье и жизнь. Вот некоторые из этих мер:

* Обеспечение комфортных и безопасных рабочих мест, соответствующих нормам охраны труда. Каждое рабочее место должно пройти специальную оценку условий труда (СОУТ), которая позволяет измерить уровень воздействия вредных производственных факторов.
* Проведение своевременных инструктажей по охране труда для всех сотрудников.
* Соблюдение техники безопасности при работе с компьютерами и другими электронными устройствами.
* Соблюдение правил внутреннего трудового распорядка и локальных нормативных актов организации.
* Предоставление первой медицинской помощи при необходимости.
* Обеспечение правильного освещения и проветривания рабочих помещений.
* Использование антивандальных систем безопасности на рабочих станциях и серверах.
* Проведение регулярных тренингов и обучение сотрудников правилам безопасности и процедурам эвакуации.
* Обеспечение безопасного хранения конфиденциальных данных.
* Соблюдение всех норм по охране труда, установленных законодательством.
* Установка пожарных датчиков и систем автоматического пожаротушения в помещениях.
* Регулярный контроль состояния техники и оборудования, использование только сертифицированного программного обеспечения и обновление антивирусных программ.

## 7.1. Характеристика объекта проектирования и условий его эксплуатации

При проектировании и эксплуатации офисных помещений, где работают сотрудники, необходимо соблюдать общие нормы по охране труда. Вот некоторые из них:

Освещение: В офисе должно быть достаточное освещение, которое может быть как естественным, так и искусственным. Цель состоит в том, чтобы предотвратить усталость глаз и головную боль у сотрудников.

Качество воздуха: Важно обеспечить свежий воздух в офисном помещении и поддерживать оптимальный уровень влажности. Это поможет создать комфортные условия для работы сотрудников и предотвратить негативные последствия для их здоровья.

Эргономика рабочих мест: Стулья, столы, клавиатуры, мыши и другое оборудование должны быть правильно настроены и предназначены для обеспечения комфорта и безопасности сотрудников. Целью является минимизация риска развития заболеваний, связанных с длительным пребыванием за компьютером, таких как мышечные и скелетные расстройства.

Предупреждение травм: Все предметы, которые могут представлять опасность и вызвать травму, должны быть удалены из рабочей зоны. Это включает в себя организацию рабочего пространства таким образом, чтобы предотвратить падение предметов или столкновение с ними.

При соблюдении данных норм по охране труда в офисных помещениях, можно создать безопасную и комфортную среду для работы сотрудников, что способствует их эффективности и благополучию.

**Организационные меры**:

* Проведение обучения и тренингов для сотрудников по эргономике рабочего места и правильным рабочим позам.
* Регулярные перерывы и упражнения для разминки и растяжки, чтобы предотвратить статическую нагрузку на мышцы и суставы.
* Организация ротации сотрудников между различными видами работ для снижения монотонности и перегрузки определенных групп мышц.
* Соблюдение регламента рабочего времени и оптимального распределения нагрузки для предотвращения переутомления.
* Проведение мероприятий по психологической поддержке сотрудников для снижения уровня стресса и эмоциональной перегрузки.

**Технические меры:**

* Использование эргономических мебели и оборудования, таких как регулируемые стулья, столы, подставки для мониторов и клавиатур.
* Установка эргономических клавиатур и мышей с учетом удобства использования и предотвращения травм рук и запястий.
* Регулировка положения монитора, чтобы предотвратить перенапряжение глаз и шеи.
* Использование антибликовых покрытий на мониторах для снижения нагрузки на зрительный орган.
* Установка систем приточно-вытяжной вентиляции для обеспечения свежего воздуха и поддержания оптимальных температур и влажности.
* Применение акустических панелей и шумопоглощающих материалов для снижения уровня шума на рабочем месте.
* Регулярное обслуживание и техническая поддержка компьютерного оборудования для предотвращения неполадок и повреждений.

**Организация рабочего пространства:**

* Правильное расположение рабочих мест с учетом эргономических требований, освещения и вентиляции.
* Размещение компьютерных кабелей и проводов таким образом, чтобы они не создавали помех и не представляли опасности для сотрудников.
* Регулярная уборка рабочих мест и оборудования для предотвращения скопления пыли и грязи.
* Предоставление возможности для перерывов и отдыха, а также организация зон для отдыха и релаксации.

При работе с персональным компьютером (ПК) необходимо соблюдать определенные меры безопасности, чтобы предотвратить возможные опасности для здоровья и жизни человека. Вот основные предосторожности:

* Правильное подключение к сети электропитания: убедитесь, что ПК подключен к заземленной розетке и использует защитный контакт на кабеле питания.
* Использование стабилизатора напряжения: рекомендуется установить стабилизатор напряжения, чтобы защитить ПК от возможных скачков напряжения в электросети.
* Избегайте повреждения кабеля питания: не изгибайте слишком сильно кабель и не помещайте на него тяжелые предметы.
* Работайте с ПК в перчатках и обуви с изолирующей конструкцией.
* Держите устройства силовой электроники на расстоянии не менее 1,25 метра от себя.
* Регулярно проверяйте, соответствует ли система заземления требованиям, проводя ежемесячную проверку.
* Перед очисткой ПК всегда выключайте его и отсоединяйте от сети.

Противопожарная безопасность также имеет большое значение при работе с ПК. Вот некоторые меры по обеспечению пожарной безопасности:

* Используйте качественные компоненты и устройства для ПК, чтобы избежать риска повреждения системы и возникновения пожара.
* Регулярно чистите ПК от пыли, так как ее накопление внутри может вызвать перегрев и стать причиной пожара. Используйте специальные средства для очистки.
* Не оставляйте компьютер включенным без присмотра на длительное время, чтобы избежать перегрева и возгорания.
* Используйте огнезащитные материалы при размещении и организации рабочего места с ПК.
* Соблюдайте правила пожарной безопасности и регулярно проводите тренировки по эвакуации и использованию средств пожаротушения.

1. ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА

**1.Расчет затрат на разработку программного обеспечения**

Затраты на оплату (ЗОТ) труда разработчика ПО включают затраты на оплату труда и отчисления от фонда заработной паты. Также затраты на оплату труда разработчика ПО складывается из двух составляющих: основной заработной платы и дополнительной заработной платы

|  |  |  |  |
| --- | --- | --- | --- |
| Категории работников, исполнители работ | Тарифная ставка  Руб./ чел.-ч. | Количество рабочих часов | Всего |
| Frontend-разработчик | 520 | 30 ч. | 15600 |
| Backend-разработчик | 1100 | 50 ч. | 55000 |
| Архитектор базы данных | 520 | 15 ч. | 7800 |
| Системный администратор | 515 | 10 ч. | 5150 |
| Тестеровщик | 470 | 10 ч. | 4700 |

Таблица 1 Расчет текущих тарифных ставок оплаты труда персонала

Дополнительная заработная плата составляет 20% от основной заработной платы, рассчитывается по формуле

Здоп = (20 /100) ⋅ Зосн,

Frontend-разработчик Здоп = 3120

Backend-разработчик 3доп = 11000

Архитектор базы данных Здоп = 1560

Системный администратор Здоп = 1030

Тестеровщик Здоп = 940

Затраты на оплату труда разработчика ПО в рублях (ФОТ - фонд оплаты труда) представляет собой сумму основной и дополнительной заработной платы и рассчитывается по формуле

ФОТ = Зосн + Здоп

Frontend-разработчик ФОТ= 18720

Backend-разработчик ФОТ = 66000

Архитектор базы данных ФОТ = 9360

Системный администратор ФОТ = 6180

Тестеровщик ФОТ = 5640

Отчисления от фонда оплаты труда включают страховые взносы:

Таблица 2 Страховые взносы с зарплаты

|  |  |  |
| --- | --- | --- |
| **Frontend-разработчик** |  | ФОТ = 18720 |
| Пенсионное страхование | 22% | 4118,4 |
| Социальное страхование | 2,9% | 542,88 |
| Медицинское страхование | 5,1% | 954,72 |
| Всего | 30% | 5616 |

|  |  |  |
| --- | --- | --- |
| **Backend-разработчик** |  | ФОТ = 66000 |
| Пенсионное страхование | 22% | 14520 |
| Социальное страхование | 2,9% | 1914 |
| Медицинское страхование | 5,1% | 3366 |
| Всего | 30% | 19800 |

|  |  |  |
| --- | --- | --- |
| **Архитектор базы данных** |  | ФОТ = 9360 |
| Пенсионное страхование | 22% | 2059,2 |
| Социальное страхование | 2,9% | 271,44 |
| Медицинское страхование | 5,1% | 477,36 |
| Всего | 30% | 2808 |

|  |  |  |
| --- | --- | --- |
| **Системный администратор** |  | ФОТ = 6180 |
| Пенсионное страхование | 22% | 1359,6 |
| Социальное страхование | 2,9% | 179,22 |
| Медицинское страхование | 5,1% | 315,18 |
| Всего | 30% | 1854 |

|  |  |  |
| --- | --- | --- |
| **Тестировщик** |  | ФОТ = 5640 |
| Пенсионное страхование | 22% | 1240,8 |
| Социальное страхование | 2,9% | 163,56 |
| Медицинское страхование | 5,1% | 287,64 |
| Всего | 30% | 1692 |

Отчисления от фонда оплаты труда рассчитываются по формуле:

ОФОТ = (30 / 100) ⋅ ФОТ

Frontend-разработчик ОФОТ = 5616

Backend-разработчик ОФОТ = 19800

Архитектор базы данных ОФОТ = 2808

Системный администратор ОФОТ = 1854

Тестеровщик ОФОТ = 1692

Затраты на оплату труда разработчика ПО рассчитываются по формуле:

ЗОТ = ФОТ + ОФОТ,

Frontend-разработчик ЗОТ = 24336

Backend-разработчик ЗОТ = 85800

Архитектор базы данных ЗОТ = 12168

Системный администратор ЗОТ = 8034

Тестеровщик ЗОТ = 7332

**2. Эксплуатационные затраты на оборудование**

При написании программы в качестве оборудования предполагается

персональный компьютер, стоимость которого составляет: Cобор.– 100000 руб.

**2.1** Суммарная годовая стоимость эксплуатационных затрат Cэз рассчитывается по формуле :

Cэз = СТО +СМК +СЭЭ + Агод,

где СТО - затраты на техническое обслуживание и ремонт оборудования, руб.,

СМК – затраты на материалы и комплектующие, руб.,

СЭЭ - годовая стоимость электроэнергии, руб.,

Агод - годовые амортизационные отчисления, руб.

Затраты на техническое обслуживание и ремонт (СТО) составляют 3 % от

стоимости оборудования по формуле :

СТО = 0,03 ⋅ Собор,

Сто = 0.03 \* 100000 = 3000.00

Затраты на материалы и комплектующие принимаются в размере 2% от стоимости оборудования по формуле :

СМК = 0,02 ⋅ Собор,

Смк = 0.02 \* 150000 = 2000.00

Амортизационные отчисления производятся по установленным нормам амортизации, выражаются, в процентах к стоимости оборудования и рассчитываются по формуле :

Агод = Собор ⋅ (НА / 100),

Агод = 150000\*((100:5):100) = 20000.00

где Cобор – стоимость компьютера;

НА – норма амортизации, которая рассчитывается по формуле :

НА = 100 / Тнорм,,

где Тнорм – нормативный срок службы (для персонального компьютера Тнорм = 5 лет)

Стоимость электроэнергии вычисляется по формуле :

СЭЭ = M ⋅ kз ⋅ Fэф ⋅ Сквт.ч ⋅ КС,

Сээ= 0.5 \* 0.8 \* 1936.48 \* 6,5 \* 1.05 = 5286.5

где M – мощность компьютера, Квт (емкость аккумулятора);

kз – коэффициент загрузки, учитывающий использование оборудования по

времени (0,8);

Fэф – эффективный фонд рабочего времени;

Cквт.ч – стоимость 1 квт-час электроэнергии (от 6,76 - 7,03- 8,84 - 9,2 руб./квт по состоянию на март 2022 года);

КС - коэффициент, учитывающий потери в сети (Kc=1,05).

Fэф – эффективный фонд рабочего времени, рассчитывается по формуле

Fэф = Дном ⋅ d ⋅ (1 – (f / 100))

Fэт=247\*8\*(1-(2/100))=1936.48

Дном = 247 – номинальное число рабочих дней в году при пятидневной рабочей неделе;

d = 8 – продолжительность рабочего дня, час;

f = 2% – планируемый процент времени на ремонт оборудования.

**2.7** Рассчитываем суммарную годовую стоимость эксплуатационных затрат Cэз используя формулу:

Cэз = СТО +СМК +СЭЭ + Агод

Сзз=3000+2000+5286.5+ 20000=30286.59

Эз = ∑t х (Сэз / Fэф),

Frontend разработчик:

Эз = ∑9880 х (38336.88/ 1936.48)=195596.32

Бэкен-разработчик:

Эз = ∑9880 х (38336.88/ 1936.48)=195596.32

Архитектор баз данных:

Эз = ∑8645 х (38336.88/ 1936.48)=171.146.78

Системный администратор:

Эз = ∑7410 х (38336.88/ 1936.48)=146.697.24

Тестировщик:

Эз = ∑2470 х (38336.88/ 1936.48)=48899.08

**Общие значения годовых расходов составляет : 757935.74 руб.**

**3. Затраты на материалы**

Расчет затрат на материалы осуществляется по формуле:

ЗМ = (100 х 10) / 100 = 100 руб

ЗМ – затраты на материалы, руб.;

Vк - общий объем количества строк на листе;

Н – норматив затрат на материалы в расчете на 100 строк, руб.

Возможен расчет затрат на материалы (ЗМ ) прямым счетом (по факту).

Таблица 3

Расчет затрат на материалы

|  |  |  |  |
| --- | --- | --- | --- |
| **Материалы** | **Количество, шт.** | **Цена за единицу, руб.** | **Всего, руб.** |
| Оплата услуг провайдера | 1 | 900 | 900 |
| Источник бесперебойного питания | 0 | 0 | 0 |
| Бумага | 100 листов | 10 | 1000 |
| Картридж | 1 | 3000 | 3000 |
|  |  |  |  |
| Итого (ЗМ): | | | 4900 |

**Накладные расходы**, связанные с управлением, организационными расходами и прочими дополнительными затратами, составляют 40% от фонда оплаты труда, вычисляются по формуле:

Cнакл = 0,4× ФОТ,

Frontend-разработчик Cнакл = 7488

Backend-разработчик Cнакл = 26400

Архитектор базы данных Cнакл = 3744

Системный администратор Cнакл = 2472

Тестеровщик Cнакл = 2256

**4.**  **Себестоимость разработки программного обеспечения**

Таблица 4

Стоимость программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| **№**  **п/п** | **Наименование статей затрат** | **Обозначение** | **Сумма, руб.** |
| 1 | Затраты на материалы | ЗМ | 100 |
| 2 | Эксплуатационные затраты (ф.12) | Эз | 757935.74 |
| 3 | Фонд оплаты труда (ф.2) | ФОТ | 105900 |
| 4 | Отчисления от фонда оплаты труда (ф.3) | ОФОТ | 31770 |
| 5 | Накладные расходы (ф.14) | Cнакл | 42360 |
| 6 | Себестоимость ПО без учета расходов на сопровождение и адаптацию | п.1+п.2+п.3+п.4+п.5 | 967529,41 |
| 7 | Расходы на сопровождение и адаптацию | 10% от п. 6 | 96752.94 |
| 8 | Полная (плановая) себестоимость ПО | Спол.= п.6 + п.7 | 1147659,41 |

**Расчет экономического эффекта разработчика и пользователя (заказчика) программного обеспечения**

**Экономический эффект у разработчика программного обеспечения.**

Заказчик оплачивает разработчику всю сумму расходов по проекту (полная себестоимость ПО из таблицы 4 с учетом прибыли разработчика и налога на добавленную стоимость с учетом качества, потребительских свойств продукции (ПО) и конъюнктуры рынка.

Таким образом, в дипломном проекте отпускная цена программного обеспечения, представляет собой не цену за единицу продукции, а цену проекта вместе с его исходными кодами и документацией, за которую его можно продать и получить определенную выгоду. Прогнозируемая отпускная цена ПО (ЦПО) с учетом НДС рассчитывается по формуле :

ЦПО = ,

где

ЦПО -отпускная цена программного обеспечения;

Сполн – полная (плановая) себестоимость ПО, руб., (из таблицы 4);

П – прибыль разработчика ПО, руб.;

СТНДС – ставка налога на добавленную стоимость (=20%), в %.

ЦПО = ,

Прибыль закладывается в цену исходя из уровня рентабельности (устанавливается студентом самостоятельно), расчет производится по формуле (16):

П = ,

где

П – прибыль, руб;

R – уровень рентабельности, % (в рамках дипломного проекта рекомендуемый уровень рентабельности ≈ 15-20-30%).

Сполн – полная (плановая) себестоимость ПО, руб

П = руб.

Ввиду того, что программное обеспечение разрабатывается для одного объекта, в качестве экономического эффекта разработчика от реализованного программного обеспечения можно рассматривать чистую прибыль (ЧП), которая рассчитывается по формуле:

ЧП = ,

ЧП - чистая прибыль, руб.;

П – прибыль, руб;

СТП – ставка налогообложения прибыли составляет 20%.

ЧП = ,

ЗАКЛЮЧЕНИЕ

В результате данной исследовательской работы удалось успешно разработать образовательный сервис по обучению языку программирования Python. Были достигнуты поставленные цели и выполнены все необходимые задачи для достижения успеха проекта.

В ходе работы был проведен анализ предметной области и технической литературы, а также был выполнен обзор аналогичных решений и актуальности программного продукта. Были сформулированы функциональные и нефункциональные требования к сервису. Также был проведен обзор программно-аппаратных средств и архитектуры программного обеспечения.

В рамках работы была разработана модель программного обеспечения, включая use-case и диаграмму последовательности. Была спроектирована база данных и графический интерфейс пользователей. Разработанный программный продукт успешно прошел тестирование. Были созданы логическая модель и модули программного продукта. Также было выполнено технико-экономическое обоснование проекта.

Дальнейшие перспективы исследования в этом направлении могут включать расширение функциональности сервиса и его адаптацию для других образовательных учреждений или организаций в области здравоохранения.

Результаты данного исследования могут быть полезны педагогам при подготовке занятий по разработке информационных систем. Кроме того, данная работа может стать отправной точкой для дальнейших исследований в рамках Государственной программы "Развитие здравоохранения" Ставропольского края.

В процессе выполнения данного проекта я приобрел знания, умения и навыки в разработке программных проектов. Я уверен, что полученный опыт поможет мне избежать ошибок в будущей профессиональной карьере в области техники-программирования. Особые трудности вызвало разработать эффективное взаимодействие программы с базой данных, но благодаря этим сложностям я приобрел ценный опыт и развил свои навыки в данной области.

СПИСОК ИСТОЧНИКОВ ИНФОРМАЦИИ

1. Кузнецов С.Д. Основы баз данных. - 2-е изд. - М.: "Интернет-университет информационных технологий - ИНТУИТ. ру", 2018;
2. Норенков И. П. Основы автоматизированного проектирования: Учеб. для вузов. 2-е изд., перераб. и доп. — М.: Изд-во МГТУ им. Н. Э. Баумана, 2018 — 336 с.;
3. Г.Н. Федорова «Разработка, внедрение и адаптация программного обеспечения отраслевой направленности»: Учебное пособие / Г.Н. Федорова. – Москва: КУРС: ИНФРА – М, 2021. – 336 c.;
4. Л.Г. Гагарина “Разработка и эксплуатация автоматизированных информационных систем”: Учебное пособие / Л.Г. Гагарина. – Москва: ИД “ФОРУМ”: ИНФРА – М, 2021. – 384 с.;
5. В.В. Лисяк “Разработка информационных систем”: Учебное пособие / В.В. Лисяк; Южный федеральный университет. – Ростов-на-Дону; Таганрог: Издательство Южного федерального университета, 2019. – 96 с.;
6. А.В. Абрамян “Разработка пользовательского интерфейса на основе системы Windows From”: Учебник / А.В. Абрамян, М.Э. Абрамян; Южный федеральный университет. – Ростов-на-Дону; Таганрог: Издательство Южного федерального университета, 2020. – 301 с.;
7. H. ABIMBOLA SORIY AN, ANJA MURSU, and MIKKO KORPELA Information System Development Methodologies, 2019. – pp 154;
8. "C# in Depth" автора Джон Скинне - 2018
9. "Pro C# 9: With .NET 5" авторов Andrew Troelsen и Philip Japikse - 2019.
10. "Head First C#" авторов Эндрю Стеллман и Дженнифер Грин - 2018.
11. "C# 9.0 in a Nutshell: The Definitive Reference" авторов Джозеф Албахари и Бен Албахари - 2020.
12. "C# Programming Yellow Book" автора Роб Майлс - 2018.
13. "C# 9 and .NET 5 – Modern Cross-Platform Development Cookbook" автора Mark J. Price - 2019.

ПРИЛОЖЕНИЕ 1. ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ

public interface IBaseResponse<T>

{

string Description { get; set; }

StatusCode StatusCode { get; set; }

T Data { get; }

}

public class BaseResponse<T> : IBaseResponse<T>

{

public string Description { get; set; }

public StatusCode StatusCode { get; set; }

public T Data { get; set; }

}

public class Answer

{

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

public int QuestionId { get; set; }

public string Text { get; set; }

public bool IsCorrect { get; set; }

}

public class Article

{

[Key]

public int Id { get; set; }

[Required]

public int UserId { get; set; }

[Required]

public int TitleId { get; set; }

[Required]

public DateTime CreatedDate { get; set; }

[Required]

public string? ArticleText { get; set; }

[Required]

[ForeignKey("TitleId")]

public virtual Title Title { get; set; }

[Required]

[ForeignKey("UserId")]

public virtual User User { get; set; }

}

public class Course

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

[ValidateNever]

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

}

public class Lecture

{

public int Id { get; set; }

public int LessonId { get; set; }

public string Name { get; set; }

public string Text { get; set; }

public bool IsComplited { get; set; }

}

public class Lesson

{

public int Id { get; set; }

public int CourseId { get; set; }

[Required]

[MaxLength(60)]

public string Name { get; set; }

public bool status { get; set; }

public virtual Course course { get; set; }

public List<Lecture>? Lectures { get; set; }

public List<Test>? Tests { get; set; }

public List<Practice>? Practices { get; set; }

}

public class LessonComment

{

public int Id { get; set; }

[Required]

public int UserId { get; set; }

[Required]

public int LessonId { get; set; }

[Required]

[MaxLength(300)]

public string CommentText { get; set; }

}

public class Question

{

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

public int TestId { get; set; }

public string Text { get; set; }

public string CorrectAnswer { get; set; }

public ICollection<Answer>? Answers { get; set; }

public Test Test { get; set; }

}

public class Solution

{

public int Id { get; set; }

[Required]

[MaxLength(300)]

public string SolutionText { get; set; }

[Required]

public int UserId { get; set; }

[Required]

public int LessonId { get; set; }

[ValidateNever]

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

}

public class Test

{

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

public int LessonId { get; set; }

public string Name { get; set; }

public ICollection<Question>? Questions { get; set; }

}

public class Title

{

public int Id { get; set; }

public int ArticleId { get; set; }

[Required]

[StringLength(20)]

public string Name { get; set; }

[Required]

[StringLength(300)]

public string ShortDescription { get; set; }

[Required]

public virtual Article Article { get; set; }

}

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; }

}

public enum Roles

{

//users Codes 0-20

[Display(Name = "Админ")]

Admin = 1,

Moderator = 2,

User = 3

}

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

}

public class CompleteArticle

{

public int TitleId { get; set; }

public int ArticleId { get; set; }

[Required]

[StringLength(100)]

public string Name { get; set; }

[Required]

[StringLength(1000)]

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; }

}

public class FullCourseViewModel

{

public string Name { get; set; }

public string Description { get; set; }

[ValidateNever]

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

}

public class LectureViewModel

{

public int LessonId { get; set; }

public string Name { get; set; }

[ValidateNever]

public string Text { get; set; }

}

public class OptionViewModel

{

public string Text { get; set; }

public bool IsCorrect { get; set; }

}

public class QuestionViewModel

{

public string Text { get; set; }

public List<OptionViewModel> Options { get; set; }

}

public class TestViewModel

{

public int Lesson { get; set; }

public string Name { get; set; }

public List<QuestionViewModel> Questions { get; set; }

public class LoginViewModel

{

/// <summary>

/// Логин пользователя

/// </summary>

[Required(ErrorMessage = "Введите логин")]

[MaxLength(30, ErrorMessage ="Имя должно быть менее 30 символов")]

[MinLength(3, ErrorMessage = "Имя должно быть длинее 3 символов")]

public string UserName { get; set; }

/// <summary>

/// Пароль пользователя

/// </summary>

[Required(ErrorMessage = "Введите пароль")]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

}

}

public class UserRegistorViewModel : IdentityUser

{

[Required]

public string Name { get; set; }

[Required]

public string SecondName { get; set; }

[Required]

public DateTime? BirthDay { get; set; }

[Required]

[PersonalData]

public string Login { get; set; }

[Required]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required]

[DataType(DataType.Password)]

[Compare("Password", ErrorMessage = "Пароли не совпадают")]

public string ConfirmPassword { get; set; }

public string AboutMe { get; set; }

[Required]

public IFormFile avatar { get; set; }

}

public class UserViewModel

{

public int Id { get; set; }

public string Name { get; set; }

public string SecondName { get; set; }

public DateTime? BirthDay { get; set; }

[ValidateNever]

public string Login { get; set; }

[ValidateNever]

public string Email { get; set; }

public string AboutMe { get; set; }

[ValidateNever]

public IFormFile avatar { get; set; }

[ValidateNever]

public byte[] Image { get; set; }

}

public interface IBaseRepository<T>

{

public Task<T?> GetAsync(int id);

IQueryable<T> GetAllAsync();

public Task CreateAsync(T entity);

public Task UpdateAsync(T entity);

public Task DeleteAsync(int id);

}

public interface IUnitOfWork

{

public AnswerRepository AnswerRepository { get; }

public ArticleCommentRepository ArticleCommentRepository { get; }

public ArticleRepository ArticleRepositories{ get; }

public CourseRepository CourseRepositories { get; }

public LectureRepository LectureRepositories { get; }

public LessonCommentRepository LessonCommentsRepositories { get; }

public LessonRepository LessonRepositories { get; }

public PracticeRepository PracticeRepositories { get; }

public SolutionRepository SolutionRepositories { get; }

public TitleRepository TitleRepositories { get; }

public TestRepository TestRepositories { get; }

public QuestionRepository QuestionRepositories { get; }

public UserRepository UserRepositories { get; }

public void Save();

public void Dispose();

}

public class UserConfiguration : IEntityTypeConfiguration<User>

{

public void Configure(EntityTypeBuilder<User> builder)

{

builder.HasData(new User

{

Id = 1,

Name = "Kosta",

SecondName = "Chistiakov",

BirthDay = DateTime.Parse("1995-06-03"),

Email = "3kers@mail.ry",

Password = CreateHash.CreateMD5Hash("123456"),

Role = Roles.Admin,

AboutMe = "text",

avatar = new byte[] {1,2,3,4,5},

Login = "kostaqw"

}) ;

builder.ToTable("Users").HasKey(x => x.Id);

builder.Property(x => x.Id).ValueGeneratedOnAdd();

builder.Property(x => x.Password).IsRequired();

builder.Property(x => x.Login).IsRequired();

builder.Property(x => x.Name).IsRequired();

builder.Property(x => x.SecondName).IsRequired();

builder.Property(x => x.Email).IsRequired();

}

}

public static class CreateHash

{

public static string CreateMD5Hash(string input)

{

// Step 1, calculate MD5 hash from input

MD5 md5 = System.Security.Cryptography.MD5.Create();

byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(input);

byte[] hashBytes = md5.ComputeHash(inputBytes);

// Step 2, convert byte array to hex string

StringBuilder sb = new StringBuilder();

for (int i = 0; i < hashBytes.Length; i++)

{

sb.Append(hashBytes[i].ToString("X2"));

}

return sb.ToString();

}

}

public static class ImageProcess

{

public static IFormFile ConverToFormFile(byte[] srcImage)

{

var stream = new MemoryStream(srcImage);

IFormFile file = new FormFile(stream, 0, srcImage.Length, "image", "avatar");

return file;

}

public static byte[] GetByteArrayFromIFormFile(IFormFile file)

{

using var memoryStream = new MemoryStream();

file.CopyTo(memoryStream);

return memoryStream.ToArray();

}

}

}

public class AnswerRepository : IAnswerRepository

{

private readonly ApplicationDbContext \_context;

public AnswerRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(Answer entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var answer = await \_context.Answers.FirstOrDefaultAsync(x => x.Id == id);

if (answer == null)

{

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

}

\_context.Answers.Remove(answer);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Answer entity)

{

var answer = await \_context.Answers.FirstOrDefaultAsync(x => x.Id == entity.Id);

if (answer == null)

{

throw new ArgumentNullException("[AnswerRepository] UpdateAsync(Answer entity): the answer isn't found");

}

answer.Text = entity.Text;

answer.IsCorrect = entity.IsCorrect;

await \_context.SaveChangesAsync();

}

public async Task<Answer> GetAsync(int id)

{

var answer = await \_context.Answers.FirstOrDefaultAsync(x => x.Id == id);

if (answer == null)

{

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

}

return answer;

}

public IQueryable<Answer> GetAllAsync()

{

return \_context.Answers.AsQueryable();

}

}

public class ArticleCommentRepository : IArticleCommentRepository

{

ApplicationDbContext \_context;

public ArticleCommentRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(ArticleComment entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var articleComment = \_context.ArticleComments.FirstOrDefault(x => x.Id == id);

if (articleComment == null)

{

throw new ArgumentNullException("[ArticleCommentRepository] DeleteAsync(int id): the article comment isn't found");

}

\_context.ArticleComments.Remove(articleComment);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(ArticleComment entity)

{

var articleComment = \_context.ArticleComments.FirstOrDefault(x => x.Id == entity.Id);

if (articleComment == null)

{

throw new ArgumentNullException("[ArticleCommentRepository] UpdateAsync(int id): the article comment isn't found");

}

if (!string.IsNullOrEmpty(entity.CommentText))

{

articleComment.CommentText = entity.CommentText;

}

articleComment.UserId = entity.UserId;

articleComment.ArticleId = entity.ArticleId;

await \_context.SaveChangesAsync();

}

public async Task<ArticleComment?> GetAsync(int id)

{

var result = await \_context.ArticleComments.FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

throw new ArgumentNullException("[ArticleCommentRepository] GetAsync(int id): the article comment isn't found");

}

else

{

return result;

}

}

public IQueryable<ArticleComment> GetAllAsync()

{

var result = \_context.ArticleComments;

if (result == null)

{

throw new ArgumentNullException("[ArticleCommentRepository] GetAsync(int id): the article comments arn't found");

}

else

{

return result;

}

}

}

public class ArticleRepository : IArticleRepository

{

ApplicationDbContext \_context;

public ArticleRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(Article entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

/// <summary>

/// Удалить статью и теме по id статьи

/// </summary>

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

/// <returns></returns>

/// <exception cref="ArgumentNullException">Статья не найдена</exception>

public async Task DeleteAsync(int id)

{

var atricle = \_context.Articles.FirstOrDefault(x => x.Id == id);

if (atricle == null)

{

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

}

var title = \_context.Titles.FirstOrDefault(x => x.Id == atricle.TitleId);

\_context.Titles.Remove(title);

\_context.Articles.Remove(atricle);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Article entity)

{

var article = \_context.Articles.FirstOrDefault(x => x.Id == entity.Id);

if (article == null)

{

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

}

if (!string.IsNullOrEmpty(article.ArticleText))

{

entity.ArticleText = article.ArticleText;

}

article.TitleId = entity.TitleId;

article.UserId = entity.UserId;

await \_context.SaveChangesAsync();

}

public async Task<Article?> GetAsync(int id)

{

var result = await \_context.Articles.Include(c=>c.Title).Include(d=>d.User).FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

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

}

else

{

return result;

}

}

public IQueryable<Article> GetAllAsync()

{

var result = \_context.Articles.Include(c => c.Title).Include(d => d.User);

if (result == null)

{

throw new ArgumentNullException("[ArticleRepository] GetAllAsync(): the articles arn't found");

}

else

{

return result;

}

}

}

public class CourseRepository : ICourseRepository

{

ApplicationDbContext \_context;

public CourseRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(Course entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var course = \_context.Courses.FirstOrDefault(x => x.Id == id);

if (course == null)

{

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

}

\_context.Courses.Remove(course);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Course entity)

{

var course = \_context.Courses.FirstOrDefault(x => x.Id == entity.Id);

if (course == null)

{

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

}

await \_context.SaveChangesAsync();

}

public async Task<Course?> GetAsync(int id)

{

var result = await \_context.Courses

.Include(c => c.Lessons).ThenInclude(a => a.Tests).ThenInclude(q=>q.Questions)

.Include(c => c.Lessons).ThenInclude(e => e.Lectures)

.Include(c => c.Lessons).ThenInclude(a => a.Practices)

.FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

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

}

else

{

return result;

}

}

public IQueryable<Course> GetAllAsync()

{

var result = \_context.Courses.Include(c=>c.Lessons);

if (result == null)

{

throw new ArgumentNullException("[CourseRepository] GetAllAsync(): the courses arn't found");

}

else

{

return result;

}

}

}

public class LectureRepository : ILectureRepository

{

private readonly ApplicationDbContext \_context;

public LectureRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(Lecture entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var lecture = await \_context.Lectures.FirstOrDefaultAsync(x => x.Id == id);

if (lecture == null)

{

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

}

\_context.Lectures.Remove(lecture);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Lecture entity)

{

var lecture = await \_context.Lectures.FirstOrDefaultAsync(x => x.Id == entity.Id);

if (lecture == null)

{

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

}

if (!string.IsNullOrEmpty(entity.Text))

{

lecture.Text = entity.Text;

}

lecture.IsComplited = entity.IsComplited;

await \_context.SaveChangesAsync();

}

public async Task<Lecture> GetAsync(int id)

{

var result = await \_context.Lectures.FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

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

}

else

{

return result;

}

}

public IQueryable<Lecture> GetAllAsync()

{

var result = \_context.Lectures;

if (result == null)

{

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

}

else

{

return result;

}

}

}

public class LessonCommentRepository : ILessonCommentRepository

{

ApplicationDbContext \_context;

public LessonCommentRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(LessonComment entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var lessonComment = \_context.LessonComments.FirstOrDefault(x => x.Id == id);

if (lessonComment == null)

{

throw new ArgumentNullException("[LessonCommentRepository] DeleteAsync(int id): the lesson comment isn't found");

}

\_context.LessonComments.Remove(lessonComment);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(LessonComment entity)

{

var lessonComment = \_context.LessonComments.FirstOrDefault(x => x.Id == entity.Id);

if (lessonComment == null)

{

throw new ArgumentNullException("[LessonCommentRepository] UpdateAsync(int id): the lesson comment isn't found");

}

if (!string.IsNullOrEmpty(entity.CommentText))

{

lessonComment.CommentText = entity.CommentText;

}

lessonComment.UserId = entity.UserId;

lessonComment.LessonId = entity.LessonId;

await \_context.SaveChangesAsync();

}

public async Task<LessonComment> GetAsync(int id)

{

var result = await \_context.LessonComments.FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

throw new ArgumentNullException("[LessonCommentRepository] GetAsync(int id): the lesson comment isn't found");

}

else

{

return result;

}

}

public IQueryable<LessonComment> GetAllAsync()

{

var result = \_context.LessonComments;

if (result == null)

{

throw new ArgumentNullException("[LessonCommentRepository] GetAllAsync(): the lesson comments arn't found");

}

else

{

return result;

}

}

}

public class LessonRepository : ILessonRepository

{

ApplicationDbContext \_context;

public LessonRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(Lesson entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var lesson = \_context.Lessons.FirstOrDefault(x => x.Id == id);

if (lesson == null)

{

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

}

\_context.Lessons.Remove(lesson);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Lesson entity)

{

var lesson = \_context.Lessons.FirstOrDefault(x => x.Id == entity.Id);

if (lesson == null)

{

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

}

if (!string.IsNullOrEmpty(entity.Name))

{

lesson.Name = entity.Name;

}

lesson.status = entity.status;

await \_context.SaveChangesAsync();

}

public async Task<Lesson> GetAsync(int id)

{

var result = await \_context.Lessons.FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

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

}

else

{

return result;

}

}

public IQueryable<Lesson> GetAllAsync()

{

var result = \_context.Lessons.Include(c=>c.Tests).Include(d=>d.Lectures).Include(e=>e.Practices);

if (result == null)

{

throw new ArgumentNullException("[LessonRepository] GetAllAsync(): the lessons arn't found");

}

else

{

return result;

}

}

}

public class QuestionRepository : IQuestionRepository

{

private readonly ApplicationDbContext \_context;

public QuestionRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(Question entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var question = await \_context.Questions.FindAsync(id);

if (question == null)

{

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

}

\_context.Questions.Remove(question);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Question entity)

{

var question = await \_context.Questions.FindAsync(entity.Id);

if (question == null)

{

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

}

if (!string.IsNullOrEmpty(entity.Text))

{

question.Text = entity.Text;

}

if (!string.IsNullOrEmpty(entity.CorrectAnswer))

{

question.CorrectAnswer = entity.CorrectAnswer;

}

if (question.Answers != null)

{

question.Answers = entity.Answers;

}

await \_context.SaveChangesAsync();

}

public async Task<Question> GetAsync(int id)

{

var question = await \_context.Questions.FindAsync(id);

if (question == null)

{

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

}

else

{

return question;

}

}

public IQueryable<Question> GetAllAsync()

{

var questions = \_context.Questions;

if (questions == null)

{

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

}

else

{

return questions;

}

}

}

public class SolutionRepository : ISolutionRepository

{

ApplicationDbContext \_context;

public SolutionRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(Solution entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var solution = \_context.Solutions.FirstOrDefault(x => x.Id == id);

if (solution == null)

{

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

}

\_context.Solutions.Remove(solution);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Solution entity)

{

var solution = \_context.Solutions.FirstOrDefault(x => x.Id == entity.Id);

if (solution == null)

{

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

}

if(!string.IsNullOrEmpty(entity.SolutionText))

{

solution.SolutionText= entity.SolutionText;

}

await \_context.SaveChangesAsync();

}

public async Task<Solution> GetAsync(int id)

{

var result = await \_context.Solutions.FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

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

}

else

{

return result;

}

}

public IQueryable<Solution> GetAllAsync()

{

var result = \_context.Solutions;

if (result == null)

{

throw new ArgumentNullException("[SolutionRepository] GetAllAsync(): the Solutions arn't found");

}

else

{

return result;

}

}

}

}

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;

}

}

}

public class TitleRepository : ITitleRepository

{

ApplicationDbContext \_context;

public TitleRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(Title entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var title = \_context.Titles.FirstOrDefault(x => x.Id == id);

if (title == null)

{

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

}

\_context.Titles.Remove(title);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Title entity)

{

var title = \_context.Titles.FirstOrDefault(x => x.Id == entity.Id);

if (title == null)

{

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

}

if (!string.IsNullOrEmpty(entity.Name))

{

title.Name = entity.Name;

}

await \_context.SaveChangesAsync();

}

public async Task<Title> GetAsync(int id)

{

var result = await \_context.Titles.FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

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

}

else

{

return result;

}

}

public IQueryable<Title> GetAllAsync()

{

var result = \_context.Titles;

if (result == null)

{

throw new ArgumentNullException("[TitleRepository] GetAllAsync(): the titles arn't found");

}

else

{

return result;

}

}

}

public class UserRepository : IUserRepository

{

ApplicationDbContext \_context;

public UserRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task CreateAsync(User entity)

{

await \_context.AddAsync(entity);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var user = \_context.Users.FirstOrDefault(x => x.Id == id);

if(user == null)

{

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

}

\_context.Users.Remove(user);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(User entity)

{

var user = \_context.Users.FirstOrDefault(x => x.Id == entity.Id);

if (user == null)

{

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

}

if (!string.IsNullOrEmpty(entity.Name))

{

user.Name = entity.Name;

}

if (!string.IsNullOrEmpty(entity.SecondName))

{

user.SecondName = entity.SecondName;

}

if (entity.BirthDay != null)

{

user.BirthDay = entity.BirthDay;

}

if (!string.IsNullOrEmpty(entity.Email))

{

user.Email = entity.Email;

}

if (!string.IsNullOrEmpty(entity.Password))

{

user.Password = entity.Password;

}

if (!string.IsNullOrEmpty(entity.AboutMe))

{

user.AboutMe = entity.AboutMe;

}

if (entity.avatar != null)

{

user.avatar = entity.avatar;

}

await \_context.SaveChangesAsync();

}

public async Task<User> GetAsync(int id)

{

var result = await \_context.Users.FirstOrDefaultAsync(x => x.Id == id);

if (result == null)

{

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

}

else

{

return result;

}

}

public IQueryable<User> GetAllAsync()

{

var result = \_context.Users;

if (result == null)

{

throw new ArgumentNullException("[UserRepository] GetAllAsync(): the users arn't found");

}

else

{

return result;

}

}

}

public class UnitOfWork : IDisposable, IUnitOfWork

{

private readonly ApplicationDbContext context;

private AnswerRepository answerRepository;

private ArticleCommentRepository articleCommentRepository;

private ArticleRepository articleRepository;

private CourseRepository courseRepository;

private LectureRepository lectureRepository;

private LessonCommentRepository lessonCommentsRepositories;

private LessonRepository lessonRepository;

private PracticeRepository practiceRepository;

private SolutionRepository solutionRepository;

private TitleRepository titleRepository;

private TestRepository testRepository;

private UserRepository userRepository;

private QuestionRepository questionRepository;

public UnitOfWork(ApplicationDbContext context)

{

this.context = context;

}

public AnswerRepository AnswerRepository

{

get

{

if(answerRepository == null)

{

answerRepository = new AnswerRepository(context);

}

return answerRepository;

}

}

public ArticleCommentRepository ArticleCommentRepository

{

get

{

if (articleCommentRepository == null)

{

articleCommentRepository = new ArticleCommentRepository(context);

}

return articleCommentRepository;

}

}

public ArticleRepository ArticleRepositories

{

get

{

if (articleRepository == null)

{

articleRepository = new ArticleRepository(context);

}

return articleRepository;

}

}

public CourseRepository CourseRepositories

{

get

{

if (courseRepository == null)

{

courseRepository = new CourseRepository(context);

}

return courseRepository;

}

}

public LectureRepository LectureRepositories

{

get

{

if (lectureRepository == null)

{

lectureRepository = new LectureRepository(context);

}

return lectureRepository;

}

}

public LessonCommentRepository LessonCommentsRepositories

{

get

{

if (lessonCommentsRepositories == null)

{

lessonCommentsRepositories = new LessonCommentRepository(context);

}

return lessonCommentsRepositories;

}

}

public LessonRepository LessonRepositories

{

get

{

if (lessonRepository == null)

{

lessonRepository = new LessonRepository(context);

}

return lessonRepository;

}

}

public SolutionRepository SolutionRepositories

{

get

{

if (solutionRepository == null)

{

solutionRepository = new SolutionRepository(context);

}

return solutionRepository;

}

}

public TitleRepository TitleRepositories

{

get

{

if (titleRepository == null)

{

titleRepository = new TitleRepository(context);

}

return titleRepository;

}

}

public UserRepository UserRepositories

{

get

{

if (userRepository == null)

{

userRepository = new UserRepository(context);

}

return userRepository;

}

}

public QuestionRepository QuestionRepositories

{

get

{

if (questionRepository == null)

{

questionRepository = new QuestionRepository(context);

}

return questionRepository;

}

}

public TestRepository TestRepositories

{

get

{

if (testRepository == null)

{

testRepository = new TestRepository(context);

}

return testRepository;

}

}

public PracticeRepository PracticeRepositories

{

get

{

if (practiceRepository == null)

{

practiceRepository = new PracticeRepository(context);

}

return practiceRepository;

}

}

public void Save()

{

context.SaveChanges();

}

private bool disposed = false;

public virtual void Dispose(bool disposing)

{

if (!disposed)

{

if (disposing)

{

context.Dispose();

}

disposed = true;

}

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

}

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());

}

}

public class AccountService : IAccountService

{

private IUnitOfWork \_context;

/// <summary>

/// Конструктор для dependence injactive

/// </summary>

/// <param name="context">Контекст</param>

public AccountService(IUnitOfWork context)

{

\_context = context;

}

/// <summary>

/// Аутентификация пользователя

/// </summary>

/// <param name="model">Вью модель логина пользователя</param>

/// <returns>Стандартный ответ с результатом операции</returns>

public async Task<IBaseResponse<ClaimsIdentity>> Login(LoginViewModel model)

{

try

{

var user = await \_context.UserRepositories.GetAllAsync().FirstOrDefaultAsync(x=>x.Login == model.UserName);

if (user == null)

{

return new BaseResponse<ClaimsIdentity>()

{

Description = "Пользователь не найден",

StatusCode = Domain.Enum.StatusCode.BadRequest

};

}

if (user.Password != CreateHash.CreateMD5Hash(model.Password))

{

return new BaseResponse<ClaimsIdentity>()

{

Description = "Неверный пароль",

StatusCode = Domain.Enum.StatusCode.BadRequest

};

}

var result = Authentificate(user);

return new BaseResponse<ClaimsIdentity>()

{

Data= result,

StatusCode = Domain.Enum.StatusCode.OK,

Description = "Успешная атуентификация"

};

}

catch (Exception ex)

{

return new BaseResponse<ClaimsIdentity>()

{

Description = $"[AccountService] Login(LoginViewModel model): {ex.Message}",

StatusCode = Domain.Enum.StatusCode.InternalServerError,

};

}

}

/// <summary>

/// Регистрация нового пользователя

/// </summary>

/// <param name="model">Вью модель пользователя</param>

/// <returns>Возвращает стандартный ответ с резульатом операции</returns>

public async Task<IBaseResponse<ClaimsIdentity>> Register(UserRegistorViewModel model)

{

try

{

var user = await \_context.UserRepositories.GetAllAsync().FirstOrDefaultAsync(x => x.Login == model.Login);

if (user != null)

{

return new BaseResponse<ClaimsIdentity>()

{

StatusCode = Domain.Enum.StatusCode.UserAlreaydyExists,

Description = "Пользователь с таким логином уже существует"

};

}

byte[] avatarFromFile = null;

using(var binaryReader = new BinaryReader(model.avatar.OpenReadStream()))

{

avatarFromFile = binaryReader.ReadBytes((int)model.avatar.Length);

}

user = new Domain.Entity.User()

{

Name = model.Name,

Email = model.Email,

AboutMe = model.AboutMe,

BirthDay = model.BirthDay,

Login = model.Login,

Password = CreateHash.CreateMD5Hash(model.Password),

Role = Domain.Enum.Roles.User,

SecondName = model.SecondName,

avatar = avatarFromFile,

};

await \_context.UserRepositories.CreateAsync(user);

var result = Authentificate(user);

return new BaseResponse<ClaimsIdentity>()

{

Data = result,

Description = "Пользователь создан",

StatusCode = Domain.Enum.StatusCode.OK

};

}

catch (Exception ex)

{

return new BaseResponse<ClaimsIdentity>()

{

Description = $"[AccountService] Register(UserRegistorViewModel model): {ex.Message}",

StatusCode = Domain.Enum.StatusCode.InternalServerError,

};

}

}

/// <summary>

/// редактирование аккаунта

/// </summary>

/// <param name="model">Вью модель</param>

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

public async Task<BaseResponse<bool>> EditAccount(UserViewModel model)

{

try

{

var user = await \_context.UserRepositories.GetAllAsync().FirstOrDefaultAsync(x => x.Login == model.Login);

if (user == null)

{

return new BaseResponse<bool>()

{

StatusCode = Domain.Enum.StatusCode.UserNotFound,

Description = "Пользователь с таким логином уже существует",

Data = false

};

}

user = new Domain.Entity.User()

{

Id= model.Id,

Name = model.Name,

Email = model.Email,

AboutMe = model.AboutMe,

BirthDay = model.BirthDay,

Login = model.Login,

SecondName = model.SecondName,

};

if (model.avatar != null)

{

byte[] avatarFromFile = null;

using (var binaryReader = new BinaryReader(model.avatar.OpenReadStream()))

{

avatarFromFile = binaryReader.ReadBytes((int)model.avatar.Length);

}

user.avatar= avatarFromFile;

}

await \_context.UserRepositories.UpdateAsync(user);

return new BaseResponse<bool>()

{

Data = true,

Description = "Пользователь изменен",

StatusCode = Domain.Enum.StatusCode.OK

};

}

catch (Exception ex)

{

return new BaseResponse<bool>()

{

Description = $"[AccountService] EditAccount(UserRegistorViewModel model): {ex.Message}",

StatusCode = Domain.Enum.StatusCode.InternalServerError,

};

}

}

/// <summary>

/// Метод аутентификации пользователя

/// </summary>

/// <param name="user">Пользователь</param>

/// <returns></returns>

private ClaimsIdentity Authentificate(User user)

{

var claims = new List<Claim>()

{

new Claim(ClaimsIdentity.DefaultNameClaimType, user.Login),

new Claim(ClaimsIdentity.DefaultRoleClaimType, user.Role.ToString())

};

return new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType,

ClaimsIdentity.DefaultRoleClaimType);

}

}

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

};

}

}

}

public class CourseService : ICourseService

{

private readonly IUnitOfWork \_context;

public CourseService(IUnitOfWork context)

{

\_context = context;

}

public async Task<IBaseResponse<Course>> GetCourse(int id)

{

try

{

Course course = await CourseRetrieval(id);

if (course != null)

{

return new BaseResponse<Course>

{

StatusCode = StatusCode.OK,

Data = course

};

}

else

{

return new BaseResponse<Course>

{

StatusCode = StatusCode.NotFound,

Description = "Course not found."

};

}

}

catch (Exception ex)

{

return new BaseResponse<Course>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while getting the course: {ex.Message}"

};

}

}

public async Task<IBaseResponse<Lecture>> GetLecture(int id)

{

try

{

var lecture = await LectureRetrieval(id);

if (lecture != null)

{

return new BaseResponse<Lecture>

{

StatusCode = StatusCode.OK,

Data = lecture

};

}

else

{

return new BaseResponse<Lecture>

{

StatusCode = StatusCode.NotFound,

Description = "Lecture not found."

};

}

}

catch (Exception ex)

{

return new BaseResponse<Lecture>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while getting the lecture: {ex.Message}"

};

}

}

public async Task<IBaseResponse<FullCourseViewModel>> GetFullCourse(int id)

{

try

{

FullCourseViewModel fullCourses = await FullCourseRetrieval(id);

if (fullCourses != null)

{

return new BaseResponse<FullCourseViewModel>

{

StatusCode = StatusCode.OK,

Data = fullCourses

};

}

else

{

return new BaseResponse<FullCourseViewModel>

{

StatusCode = StatusCode.NotFound,

Description = "Full courses not found."

};

}

}

catch (Exception ex)

{

return new BaseResponse<FullCourseViewModel>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while getting the full courses: {ex.Message}"

};

}

}

public async Task<IBaseResponse<List<Course>>> GetCourses()

{

try

{

List<Course> courses = await CourseListRetrieval();

if (courses != null && courses.Any())

{

return new BaseResponse<List<Course>>

{

StatusCode = StatusCode.OK,

Data = courses

};

}

else

{

return new BaseResponse<List<Course>>

{

StatusCode = StatusCode.NotFound,

Description = "Courses not found."

};

}

}

catch (Exception ex)

{

return new BaseResponse<List<Course>>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while getting the courses: {ex.Message}"

};

}

}

public async Task<IBaseResponse<List<Lesson>>> GetLessons()

{

try

{

var lessons = await LessonRetrieval();

if (lessons != null)

{

return new BaseResponse<List<Lesson>>

{

StatusCode = StatusCode.OK,

Data = lessons

};

}

else

{

return new BaseResponse<List<Lesson>>

{

StatusCode = StatusCode.NotFound,

Description = "Lessons not found."

};

}

}

catch (Exception ex)

{

return new BaseResponse<List<Lesson>>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while getting the lessons: {ex.Message}"

};

}

}

public async Task<IBaseResponse<bool>> CreateCourse(Course course)

{

try

{

await CourseCreation(course);

return new BaseResponse<bool>

{

StatusCode = StatusCode.OK,

Data = true

};

}

catch (Exception ex)

{

return new BaseResponse<bool>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while creating the course: {ex.Message}"

};

}

}

public async Task<IBaseResponse<bool>> CreateLesson(Lesson lesson)

{

try

{

await LessonCreation(lesson);

return new BaseResponse<bool>

{

StatusCode = StatusCode.OK,

Data = true

};

}

catch (Exception ex)

{

return new BaseResponse<bool>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while creating the lesson: {ex.Message}"

};

}

}

public async Task<IBaseResponse<bool>> CreateLecture(LectureViewModel lecture)

{

try

{

await LectureCreation(lecture);

return new BaseResponse<bool>

{

StatusCode = StatusCode.OK,

Data = true

};

}

catch (Exception ex)

{

return new BaseResponse<bool>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while creating the lecture: {ex.Message}"

};

}

}

public async Task<IBaseResponse<bool>> CreateTest(Test test)

{

try

{

await TestCreation(test);

return new BaseResponse<bool>

{

StatusCode = StatusCode.OK,

Data = true

};

}

catch (Exception ex)

{

return new BaseResponse<bool>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while creating the test: {ex.Message}"

};

}

}

public async Task<IBaseResponse<bool>> CreatePractice(Practice practice)

{

try

{

await PracticeCreation(practice);

return new BaseResponse<bool>

{

StatusCode = StatusCode.OK,

Data = true

};

}

catch (Exception ex)

{

return new BaseResponse<bool>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while creating the practice: {ex.Message}"

};

}

}

public async Task<IBaseResponse<bool>> CreateQuest(Question question)

{

try

{

await QuestionCreation(question);

return new BaseResponse<bool>

{

StatusCode = StatusCode.OK,

Data = true

};

}

catch (Exception ex)

{

return new BaseResponse<bool>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while creating the question: {ex.Message}"

};

}

}

public async Task<IBaseResponse<bool>> CreateAnswer(Answer answer)

{

try

{

await AnswerCreation(answer);

return new BaseResponse<bool>

{

StatusCode = StatusCode.OK,

Data = true

};

}

catch (Exception ex)

{

return new BaseResponse<bool>

{

StatusCode = StatusCode.InternalServerError,

Description = $"An error occurred while creating the answer: {ex.Message}"

};

}

}

private async Task<Course> CourseRetrieval(int id)

{

var course = await \_context.CourseRepositories.GetAsync(id);

return course;

}

private async Task<FullCourseViewModel> FullCourseRetrieval(int id)

{

var course = await \_context.CourseRepositories.GetAsync(id);

var lessons = \_context.LessonRepositories.GetAllAsync().Where(c => c.CourseId == id).ToList();

var fullCourse = new FullCourseViewModel()

{

Name = course.Name,

Description = course.Description,

Lessons = course.Lessons,

};

return fullCourse;

}

private async Task<List<Course>> CourseListRetrieval()

{

var courses = \_context.CourseRepositories.GetAllAsync();

return courses.ToList();

}

private async Task<List<Lesson>> LessonRetrieval()

{

var lessons = \_context.LessonRepositories.GetAllAsync();

return lessons.ToList();

}

private async Task<Lecture> LectureRetrieval(int id)

{

var lecture = await \_context.LectureRepositories.GetAsync(id);

return lecture;

}

private async Task CourseCreation(Course course)

{

await \_context.CourseRepositories.CreateAsync(course);

}

private async Task LectureCreation(LectureViewModel lecture)

{

var newLecture = new Lecture()

{

Text = lecture.Text,

Name = lecture.Name,

LessonId = lecture.LessonId,

IsComplited = false

};

await \_context.LectureRepositories.CreateAsync(newLecture);

}

private async Task LessonCreation(Lesson lesson)

{

await \_context.LessonRepositories.CreateAsync(lesson);

}

private async Task TestCreation(Test test)

{

await \_context.TestRepositories.CreateAsync(test);

}

private async Task PracticeCreation(Practice practice)

{

await \_context.PracticeRepositories.CreateAsync(practice);

}

private async Task QuestionCreation(Question question)

{

await \_context.QuestionRepositories.CreateAsync(question);

}

private async Task AnswerCreation(Answer answer)

{

await \_context.AnswerRepository.CreateAsync(answer);

}

}

public class UserService : IUserService

{

private readonly IUnitOfWork \_context;

/// <summary>

/// Конструктор для dependence injactive

/// </summary>

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

public UserService(IUnitOfWork context)

{

\_context = context;

}

//CREATE

/// <summary>

/// Создание пользователя

/// </summary>

/// <param name="user">Вью модель пользователя</param>

/// <param name="imageData">Аватарка пользоватя</param>

/// <returns>Асинхронный ответ в виде bool значения</returns>

public async Task<IBaseResponse<bool>> CreateUser(UserViewModel user, byte[] imageData)

{

var response = new BaseResponse<bool>();

try

{

var newUser = new User()

{

AboutMe = user.AboutMe,

BirthDay = user.BirthDay,

Email = user.Email,

Login = user.Login,

avatar = imageData,

Name = user.Name,

SecondName = user.SecondName

};

await \_context.UserRepositories.CreateAsync(newUser);

response.Description = $"Пользователй успешно создан";

response.StatusCode = StatusCode.OK;

response.Data = true;

return response;

}

catch (Exception ex)

{

return new BaseResponse<bool>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[UserService] CreateUser(UserViewModel user): {ex.Message}"

};

}

}

//READ

/// <summary>

/// Получить список всех пользователей

/// </summary>

/// <returns>Стандартный ответ со списком всех пользователй</returns>

public async Task<IBaseResponse<IEnumerable<User>>> GetUsersAsync()

{

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

try

{

var users = await \_context.UserRepositories.GetAllAsync().ToListAsync();

if (users.Count == 0)

{

baseResponse.StatusCode = StatusCode.Warn;

baseResponse.Description = $"Найдено 0 элементов";

return baseResponse;

}

else

{

baseResponse.StatusCode = StatusCode.OK;

baseResponse.Description = $"Найдено {users.Count} пользователей";

baseResponse.Data = users;

return baseResponse;

}

}

catch (Exception ex)

{

return new BaseResponse<IEnumerable<User>>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[UserService] GetUsersAsync(): {ex.Message}"

};

}

}

/// <summary>

/// Получить пользователя по id

/// </summary>

/// <param int="id">id пользователя</param>

/// <returns>Стандартный ответ c пользователй c заданным id </returns>

public async Task<IBaseResponse<User>> GetUserAsync(int id)

{

var response = new BaseResponse<User>();

try

{

var user = await \_context.UserRepositories.GetAsync(id);

if (user == null)

{

response.StatusCode = StatusCode.NotFound;

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

return response;

}

else

{

response.StatusCode= StatusCode.OK;

response.Description = $"Пользователь {user.Name} с id {id} найден";

response.Data = user;

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<User>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[UserService] GetUserAsync(): {ex.Message}"

};

}

}

/// <summary>

/// Получить список всех пользователей по имени

/// </summary>

/// <param string="Name">имя пользователя</param>

/// <returns>Стандартный ответ со списком всех пользователй с заданным именем</returns>

public async Task<IBaseResponse<IEnumerable<User>>> GetUserByNameAsync(string name)

{

var response = new BaseResponse<IEnumerable<User>>();

try

{

var users = await \_context.UserRepositories.GetAllAsync().Where(x=>x.Name == name).ToListAsync();

if (users.Count==0)

{

response.StatusCode = StatusCode.NotFound;

response.Description = $"Пользователи с именем {name} не найдены в БД";

return response;

}

else

{

response.StatusCode = StatusCode.OK;

response.Description = $"Найдено {users.Count} пользователей с именем {name}";

response.Data = users;

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<IEnumerable<User>>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[UserService] GetUserByNameAsync(string name): {ex.Message}"

};

}

}

/// <summary>

/// Получить список всех пользователей по фамилии

/// </summary>

/// <param string="secondName">фамилия пользователя</param>

/// <returns>Стандартный ответ со списком всех пользователй с заданной фамилией</returns>

public async Task<IBaseResponse<IEnumerable<User>>> GetUserBySecondNameAsync(string secondName)

{

var response = new BaseResponse<IEnumerable<User>>();

try

{

var users = await \_context.UserRepositories.GetAllAsync().Where(x=>x.SecondName==secondName).ToListAsync();

if (users.Count == 0)

{

response.StatusCode = StatusCode.NotFound;

response.Description = $"Пользователи с фамилией {secondName} не найдены в БД";

return response;

}

else

{

response.StatusCode = StatusCode.OK;

response.Description = $"Найдено {users.Count} пользователей с фамилией {secondName}";

response.Data = users;

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<IEnumerable<User>>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[UserService] GetUserBySecondNameAsync(string secondName): {ex.Message}"

};

}

}

//UPDATE

/// <summary>

/// Изменить поля пользователя

/// </summary>

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

/// <param name="model">модель пользователя</param>

/// <returns>Измененный пользователь</returns>

public async Task<IBaseResponse<User>> EditAsync(int id, UserViewModel model)

{

var response = new BaseResponse<User>();

try

{

var user = await \_context.UserRepositories.GetAsync(id);

if (user == null)

{

response.StatusCode = StatusCode.NotFound;

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

return response;

}

else

{

user.Name= model.Name;

user.Email= model.Email;

user.AboutMe = model.AboutMe;

user.SecondName= model.SecondName;

user.BirthDay= model.BirthDay;

user.Login= model.Login;

await \_context.UserRepositories.UpdateAsync(user);

response.StatusCode = StatusCode.OK;

response.Description = $"Пользователь изменён";

response.Data = user;

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<User>()

{

StatusCode = StatusCode.BadRequest,

Description = $"[UserService] EditAsync(int id, UserViewModel model): {ex.Message}"

};

}

}

//DELETE

/// <summary>

/// Удалить пользователя по id

/// </summary>

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

/// <returns>стандартный ответ с данными : true в случае успеха false в случае неудачи</returns>

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

{

var response = new BaseResponse<bool>();

try

{

var user = await \_context.UserRepositories.GetAsync(id);

if (user == null)

{

response.StatusCode = StatusCode.NotFound;

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

response.Data = false;

return response;

}

else

{

response.StatusCode = StatusCode.OK;

response.Description = $"Пользователь {user.Name} с id {id} удален";

response.Data = true;

await \_context.UserRepositories.DeleteAsync(id);

return response;

}

}

catch (Exception ex)

{

return new BaseResponse<bool>()

{

StatusCode = StatusCode.BadRequest,

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

Data = false

};

}

}

}

public class AccountController : Controller

{

private readonly IAccountService \_service;

private readonly IUserService \_userService;

private readonly IMapper \_mapper;

/// <summary>

/// Конструктор для dependence injactive

/// </summary>

/// <param name="context">Контекст</param>

public AccountController(IAccountService service, IUserService userService, IMapper mapper)

{

\_service = service;

\_userService = userService;

\_mapper = mapper;

}

/// <summary>

/// Отобразить окно регистрации

/// </summary>

/// <returns>Представление Register</returns>

[HttpGet]

public IActionResult Register() => View();

/// <summary>

/// Отправить данные для попытки регистрации

/// </summary>

/// <param name="model">Вью модель регистрации пользователя</param>

/// <returns>Перенаправялет на домашнию страницу</returns>

[HttpPost]

public async Task<IActionResult> Register(UserRegistorViewModel model)

{

if (ModelState.IsValid)

{

var response = await \_service.Register(model);

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

{

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(response.Data));

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

}

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

}

return View(model);

}

/// <summary>

/// Отобразить окно входа на сайт

/// </summary>

/// <returns>Представление Login</returns>

[HttpGet]

public IActionResult Login() => View();

/// <summary>

/// Отправить данные для попытки входа

/// </summary>

/// <param name="model">Логин вью модель</param>

/// <returns>Перенаправялет на домашнию страницу</returns>

[HttpPost]

public async Task<IActionResult> Login(LoginViewModel model)

{

if (ModelState.IsValid)

{

var response = await \_service.Login(model);

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

{

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(response.Data));

var userModel = \_userService.GetUsersAsync().Result.Data.FirstOrDefault(x => x.Login == model.UserName);

var userViewModel = new UserViewModel()

{

Login = userModel.Login,

Email = userModel.Email,

Id = userModel.Id,

Name = userModel.Name,

SecondName = userModel.SecondName

};

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

}

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

}

return View(model);

}

/// <summary>

/// Выход из аккаунта

/// </summary>

/// <returns>Перенаправялет на домашнию страницу</returns>

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

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

}

/// <summary>

/// Отобразить окно входа на сайт

/// </summary>

/// <returns>Представление Login</returns>

[HttpGet]

public IActionResult Profile()

{

if (User.Identity.IsAuthenticated)

{

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

var userViewModel = \_mapper.Map<UserViewModel>(user);

return View(userViewModel);

}

return View();

}

/// <summary>

/// Изменение профиля пользователя

/// </summary>

/// <param name="model">Вью модель пользователя</param>

/// <returns></returns>

[HttpPost]

public async Task<IActionResult> Profile(UserViewModel model)

{

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

if (user != null)

{

model.Email = user.Email;

model.Login= user.Login;

model.Image = user.avatar;

model.Id = user.Id;

if (ModelState.IsValid)

{

var response = await \_service.EditAccount(model);

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

{

return RedirectToAction("Index", "Home", model.Login);

}

else

{

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

}

}

}

return View(model);

}

}

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 });

}

}

public class CourseController : Controller

{

private readonly ICourseService \_service;

public CourseController(ICourseService service)

{

\_service = service;

}

[HttpGet]

public async Task<IActionResult> Main(int id)

{

var course = await \_service.GetFullCourse(id);

return View(course.Data);

}

[HttpGet]

public async Task<IActionResult> Lecture(int id)

{

var lecture = await \_service.GetLecture(id);

var viewLecture = new LectureViewModel()

{

Name = lecture.Data.Name,

Text = lecture.Data.Text,

};

return View(viewLecture);

}

[HttpGet]

public async Task<IActionResult> CreateTest()

{

var lessons = await \_service.GetLessons();

ViewBag.Lessons = new SelectList(lessons.Data, "Id", "Name");

return View();

}

[HttpGet]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> CreateLecture()

{

var lessons = await \_service.GetLessons();

ViewBag.Lessons = new SelectList(lessons.Data, "Id", "Name");

return View();

}

[HttpPost]

public async Task<IActionResult> CreateTest([FromBody] TestViewModel test)

{

var testEntity = new Test()

{

LessonId = test.Lesson,

Name = test.Name

};

await \_service.CreateTest(testEntity);

foreach (var questionViewModel in test.Questions)

{

var correctAnswer = new StringBuilder();

foreach (var answer in questionViewModel.Options)

{

correctAnswer.Append(answer.ToString());

}

var questionEntity = new Question

{

TestId = testEntity.Id,

Text = questionViewModel.Text,

CorrectAnswer = correctAnswer.ToString()

};

await \_service.CreateQuest(questionEntity);

foreach (var optionViewModel in questionViewModel.Options)

{

var optionEntity = new Answer

{

QuestionId = questionEntity.Id,

Text = optionViewModel.Text,

IsCorrect = optionViewModel.IsCorrect

};

await \_service.CreateAnswer(optionEntity);

}

}

return Ok();

}

[HttpPost]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> CreateLecture(LectureViewModel lecture)

{

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

lecture.Text= TxtInfo;

if (ModelState.IsValid)

{

var response = await \_service.CreateLecture(lecture);

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

{

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

}

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

}

return View(lecture);

}

[HttpPost]

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

{

if (folder == null)

{

folder = Path.GetRandomFileName();

}

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

if (!Directory.Exists(folderPath))

{

Directory.CreateDirectory(folderPath);

}

var filePath = "";

foreach (IFormFile photo in Request.Form.Files)

{

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

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

{

photo.CopyToAsync(stream);

}

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

}

return Json(new { url = filePath });

}

}

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

public async Task<IActionResult> Index(UserViewModel viewModel)

{

return View(viewModel);

}

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 });

}

}

public class UserController : Controller

{

private readonly IUserService \_service;

public UserController(IUserService service)

{

\_service = service;

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpGet]

public async Task<IActionResult> Save(int id)

{

if (id == 0)

{

return View();

}

var response = await \_service.GetUserAsync(id);

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

{

return View(response.Data);

}

return Redirect("Error");

}

[HttpPost]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Save(UserViewModel model)

{

if (ModelState.IsValid)

{

if (model.Id == 0)

{

byte[] imageData;

using (var binaryReader = new BinaryReader(model.avatar.OpenReadStream()))

{

imageData= binaryReader.ReadBytes((int)model.avatar.Length);

}

await \_service.CreateUser(model, imageData);

}

else

{

await \_service.EditAsync(model.Id, model);

}

}

return RedirectToAction("Error");

}

[HttpPost]

public IActionResult Register(string name, string secondName, DateTime Birthday, string login, string email, string password, string aboutMe, byte[] avatar)

{

return Redirect("./RegisterSuccess");

}

[HttpGet]

public IActionResult RegisterSuccess()

{

return View();

}

[HttpGet]

public async Task<IActionResult> GetUsers()

{

var response = await \_service.GetUsersAsync();

return View();

}

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 });

}

}

ПРИЛОЖЕНИЕ 2

ОБРАЗОВАТЕЛЬНЫЙ СЕРВИС ПО ИЗУЧЕНИЮ ЯЗЫКА ПРОГРАМИРОВАНИЯ PYTHON

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Санкт Петербург, 2023

**ВВЕДЕНИЕ**

**Область применения**

Образовательный сервис по обучению Python имеет широкую область применения и может быть полезным для различных групп пользователей. Вот некоторые области, где такой сервис может быть использован:

1. Учебные заведения: Образовательные учреждения, включая школы, колледжи и университеты, могут использовать сервис по обучению Python для предоставления студентам возможности изучать программирование и разработку приложений. Он может быть включен в учебные планы и использоваться как дополнительный ресурс для самообразования.
2. Профессиональное обучение: Курсы и тренинги по Python могут быть организованы для профессионалов, которые хотят расширить свои навыки программирования или переквалифицироваться в области разработки программного обеспечения. Образовательный сервис предлагает структурированные материалы, задания, практические проекты и оценку, что способствует освоению навыков Python.
3. Самообучение: Люди, желающие самостоятельно изучить Python, могут воспользоваться образовательным сервисом. Он предоставляет доступ к обучающим материалам, интерактивным урокам, заданиям и практическим проектам, позволяя пользователям изучать язык программирования в своем собственном темпе.
4. Веб-разработка: Python широко используется в веб-разработке, и образовательный сервис может сосредоточиться на этой области. Пользователи могут изучать основы веб-разработки, включая фреймворки Django и Flask, создание веб-приложений, работу с базами данных и другие связанные темы.

**Краткое описание возможностей**

Образовательный сервис позволяет:

● Изучать теоретический материал;

● Проходить тестирование;

● Читать различные тематические статьи

**Уровень подготовки пользователя**

Для успешного использования системы необходимо обладать определенным уровнем подготовки. Минимальные требования для пользователей интернета могут варьироваться в зависимости от их конкретных потребностей и целей. Однако, в целом, пользователь должен быть знаком с основными навыками и знаниями, чтобы эффективно взаимодействовать с сетью. Эти навыки включают:

* + - 1. Операционные навыки: Пользователь должен быть способен работать с компьютером или мобильным устройством, уметь их включать и выключать, а также управлять основными функциями и настройками.
      2. Знание терминологии: Пользователь должен быть знаком с основными терминами и понятиями, связанными с использованием интернета. Например, он должен понимать значения таких терминов, как браузер, поисковая система, ссылка, веб-сайт и другие, чтобы правильно воспринимать и использовать информацию в сети.
      3. Безопасность в интернете: Пользователь должен знать основные правила безопасности в интернете и соблюдать их. Это включает умение распознавать потенциально опасные ссылки, использование надежных паролей, защиту личных данных и прочие меры предосторожности для защиты от онлайн-угроз.

Соблюдение этих требований позволит пользователям интернета наиболее эффективно и безопасно использовать систему и извлекать пользу из своего онлайн-присутствия.

**Подготовка к работе**

Для работы с системой необходимо установить браузер, и иметь подключение к серверу (локальному или удаленному).

В случае отсутствия учетной записи, смотрите раздел “РЕКОМЕНДАЦИИ ПО ОСВОЕНИЮ”, пункт “регистрация”.

**Описание операций**

Регистрация

1. Неавторизованном пользователем необходимо перейти на любую страницу сайта, где пользователя переадресует на страницу входа в систему. Необходимо выбрать пункт “Перейти к регистрации”.

2. На странице регистрации будут доступны следующие поля ввода:

- ФИО;

- Почта;

- о себе;

- аватарка;

- пароль;

- подтверждение пароля

На данной странице необходимо заполнить поля, и нажать кнопку “Зарегистрироваться”, после чего на почту указанную в поле

3. После успешного ввода пароля, в системе создастся новый аккаунт, а пользователя автоматически перейдет на страницу авторизации.

Авторизация

1. На странице авторизации доступны 2 поля для ввода:

- Логин

- Пароль

После ввода почты и пароля, станет активной кнопка “Войти”.

2. В случае успешной авторизации, пользователь автоматически перейдет на главную страницу.

3. В случае ошибки авторизации, она будет отображена пользователю.

**Работа с сайтом**

1. Чтение блога

На главной странице нажать кнопку «Блог» также в этот раздел можно попасть с любой другой страницы через навигационное меню сайта, расположенное в шапке сайта. После перехода будет отображена страница, содержащая список статей и их краткое описание. При нажатии на заголовок статьи будет осуществлен переход к полной версии статьи

2. Изучение лекций

На главной странице нажать кнопку «Обучение» также в этот раздел можно попасть с любой другой страницы через навигационное меню, расположенное в шапке сайта. После перехода будет отображена страница, на которой показано содержимое уроков, из этого списка выбрать интересующую лекцию и нажать на её название. После прочтения лекции можно нажать кнопку завершить, и она будет помечена как прочитанная.

3. Прохождение тестов

На главной странице нажать кнопку «Обучение» также в этот раздел можно попасть с любой другой страницы через навигационное меню, расположенное в шапке сайта. После перехода будет отображена страница, на которой показано содержимое уроков, из этого списка выбрать интересующий тест и нажать на его название. После прохождения теста и нажатия кнопки отправить будет получен результат тестирования и тест пометится как пройденный.

**Аварийные ситуации**

Если возникают проблемы с техническими средствами, следует принять следующие меры:

1. Для длительных отказов технических средств, рекомендуется обновить страницу браузера и очистить куки. Для этого можно выполнить следующие действия: на любой странице во вкладке с сервисом, нажмите комбинацию клавиш CTRL+F5, чтобы перезапустить страницу с очисткой куки.
2. При обнаружении ошибок сети, стоит проверить работоспособность роутера, состояние Wi-Fi или подключение через порт. Если отсутствует подключение, рекомендуется обратиться к системным администраторам для устранения неисправностей.
3. Если возникают другие ошибки, можно воспользоваться "инспектором" в браузере. Во вкладке "Сеть" отключите опцию "кэширование запросов". После выполнения этого действия следует обновить страницу.

При соблюдении данных рекомендаций возможно решить проблемы с техническими средствами и продолжить пользоваться сервисом без значительных помех.

**Рекомендации по освоению**

Если учетной записи пользователя в системе нет, её можно создать.

1. Регистрация. Для регистрации, на странице входа необходимо нажать на ссылку “перейти к регистрации”. На отразившейся странице регистрации необходимо ввести вашу электронную почту и нажать “Зарегистрироваться

2. Вход. Для входа вам необходимо ввести логин и пароль. В случае возникновения ошибки входа, она будет отображаться текстом под формой входа. В случае ввода корректного пароля и почты, пользователь совершит вход в систему и попадает на страницу с списком проектов.

ПРИЛОЖЕНИЕ 3

ОБРАЗОВАТЕЛЬНЫЙ СЕРВИС ПО ИЗУЧЕНИЮ ЯЗЫКА ПРОГРАМИРОВАНИЯ PYTHON

РУКОВОДСТВО АДМИНИСТРАТОРА

Санкт-Петербург, 2023

**Назначение программы**

Образовательный сервис по обучению Python имеет широкую область применения и может быть полезным для различных групп пользователей. Вот некоторые области, где такой сервис может быть использован:

1. Учебные заведения: Образовательные учреждения, включая школы, колледжи и университеты, могут использовать сервис по обучению Python для предоставления студентам возможности изучать программирование и разработку приложений. Он может быть включен в учебные планы и использоваться как дополнительный ресурс для самообразования.
2. Профессиональное обучение: Курсы и тренинги по Python могут быть организованы для профессионалов, которые хотят расширить свои навыки программирования или переквалифицироваться в области разработки программного обеспечения. Образовательный сервис предлагает структурированные материалы, задания, практические проекты и оценку, что способствует освоению навыков Python.
3. Самообучение: Люди, желающие самостоятельно изучить Python, могут воспользоваться образовательным сервисом. Он предоставляет доступ к обучающим материалам, интерактивным урокам, заданиям и практическим проектам, позволяя пользователям изучать язык программирования в своем собственном темпе.
4. Веб-разработка: Python широко используется в веб-разработке, и образовательный сервис может сосредоточиться на этой области. Пользователи могут изучать основы веб-разработки, включая фреймворки Django и Flask, создание веб-приложений, работу с базами данных и другие связанные темы.

**Условия выполнения программы**

Данный сервис написан на asp.net core и требует для своего использования наличие IIS сервера

Для запуска программы необходимо установить следующий набор программ:

1. .NET Core SDK >= 7.0.0

2. .NET Core Runtime >= 7.0.0

3. IIS сервер

Минимальные аппаратные требования:

CPU: 4 ядра 2 ггц

ORM: 8 гб

HDD/SSD: 80 гб

**Сообщения администратору**

В случае возникновения определенных ошибок, рекомендуется принять следующие меры:

"Ошибка ввода данных": Это сообщение указывает на проблему записи данных в базу данных. Для решения данной проблемы можно попробовать перезапустить базу данных или проверить соединение сервера с базой данных. Если соединение отсутствует, рекомендуется перезапустить сервер.

"Содержимое страницы не найдено": Это сообщение говорит о том, что страница, на которую вы пытаетесь перейти, не существует. В таком случае, следует вернуться на главную страницу.

"Конечный сервер отверг запрос": Это сообщение указывает на проблему сети. Рекомендуется перезапустить сервер с правами администратора.

При некорректном отображении страницы или возникновении других ошибок на стороне клиента, возможной причиной может быть устаревший бандл. В таком случае рекомендуется обновить страницу с очисткой кук через комбинацию клавиш CTRL+F5 или отключить опцию "кэшировать запросы" во вкладке "Сеть" инспектора браузера.

"Некорректные настройки": Эта ошибка указывает на неправильную конфигурацию. Для ее решения необходимо обратиться к системному администратору и перенастроить конфигурационные файлы.

Для других ошибок, рекомендуется попробовать перезапустить систему.

Соблюдение этих рекомендаций поможет решить большинство проблем, связанных с ошибками и обеспечит более стабильную работу системы.