**ОГЛАВЛЕНИЕ**

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ7**

**ВВЕДЕНИЕ8**

**1. ПОСТАНОВКА ЗАДАЧИ10**

1.1. Анализ предметной области10

1.2. Формирование требований к программному продукту14

1.2.1. Бизнес-требования14

1.2.2. Пользовательские требования15

1.2.3. Функциональные требования17

1.2.4. Нефункциональные требования17

1.2.5. Ограничения18

1.2.6. Требования к интерфейсам18

1.3. Программные средства разработки18

1.4. Аппаратные средства разработки19

**2. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ20**

2.1. Архитектура системы20

2.1.1. Web API20

2.1.2. Уровень представления22

2.1.3. Уровень приложения22

2.1.4. Уровень данных23

2.2. Моделирование основных сценариев системы idef, uml23

2.2.1. Диаграммы последовательности23

2.2.2. Диаграммы активностей28

2.3. Проектирование графического интерфейса пользователя30

2.3.1. Заголовок (header)31

2.3.1.1. Поле активации формы поиска32

2.3.1.2. Кнопка активации мобильной формы поиска33

2.3.1.3. Навигационная панель35

2.3.1.4. Мобильное меню36

2.3.2. Главная страница37

2.3.3. Страница авторизации39

2.3.4. Страница восстановления пароля40

2.3.5. Страница регистрации43

2.3.6. Личный кабинет пользователя45

2.3.7. Страница изменения профиля47

2.3.8. Страница «Справочник»48

2.3.9. Карточка творческого деятеля49

2.3.10. Страница «Мероприятия»50

2.3.11. Карточка мероприятия51

2.3.12. Ошибка «404»52

2.4. Проектирование базы данных53

2.4.1. Структура и описание таблиц53

2.4.1.1. Таблица user53

2.4.1.2. Таблица person54

2.4.1.3. Таблица art54

2.4.1.4. Таблица event55

2.4.1.5. Таблица tag55

2.4.1.6. Таблица refresh\_session56

2.4.1.7. Таблица person\_likes56

2.4.1.8. Таблица person\_tags56

2.4.2. ER-диаграмма57

**3. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА58**

3.1. Структурные принципы разработки клиентской части программного продукта58

3.2. Разработка пользовательского интерфейса62

3.3. Разработка основного функционала65

**4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА68**

4.1. История изменений68

4.2. Терминология68

4.3. Стратегия тестирования68

4.4. Определение объектов тестирования68

4.5. Описание процесса тестирования69

**5. РАЗРАБОТКА ТЕХНОЛОГИЧЕСКОЙ ДОКУМЕНТАЦИИ73**

5.1. Руководство пользователя73

5.2. Руководство администратора79

5.2. Руководство разработчика81

**6. МЕРОПРИЯТИЯ ПО ТЕХНИКЕ БЕЗОПАСНОСТИ И ОХРАНЕ ТРУДА82**

6.1. Техника безопасности на рабочем месте82

6.2. Правила охраны труда для программиста84

**7. ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА85**

**ЗАКЛЮЧЕНИЕ89**

**СПИСОК ИСТОЧНИКОВ ИНФОРМАЦИИ90**

**ПРИЛОЖЕНИЕ 1. ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ** **91**

**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 инновационной 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

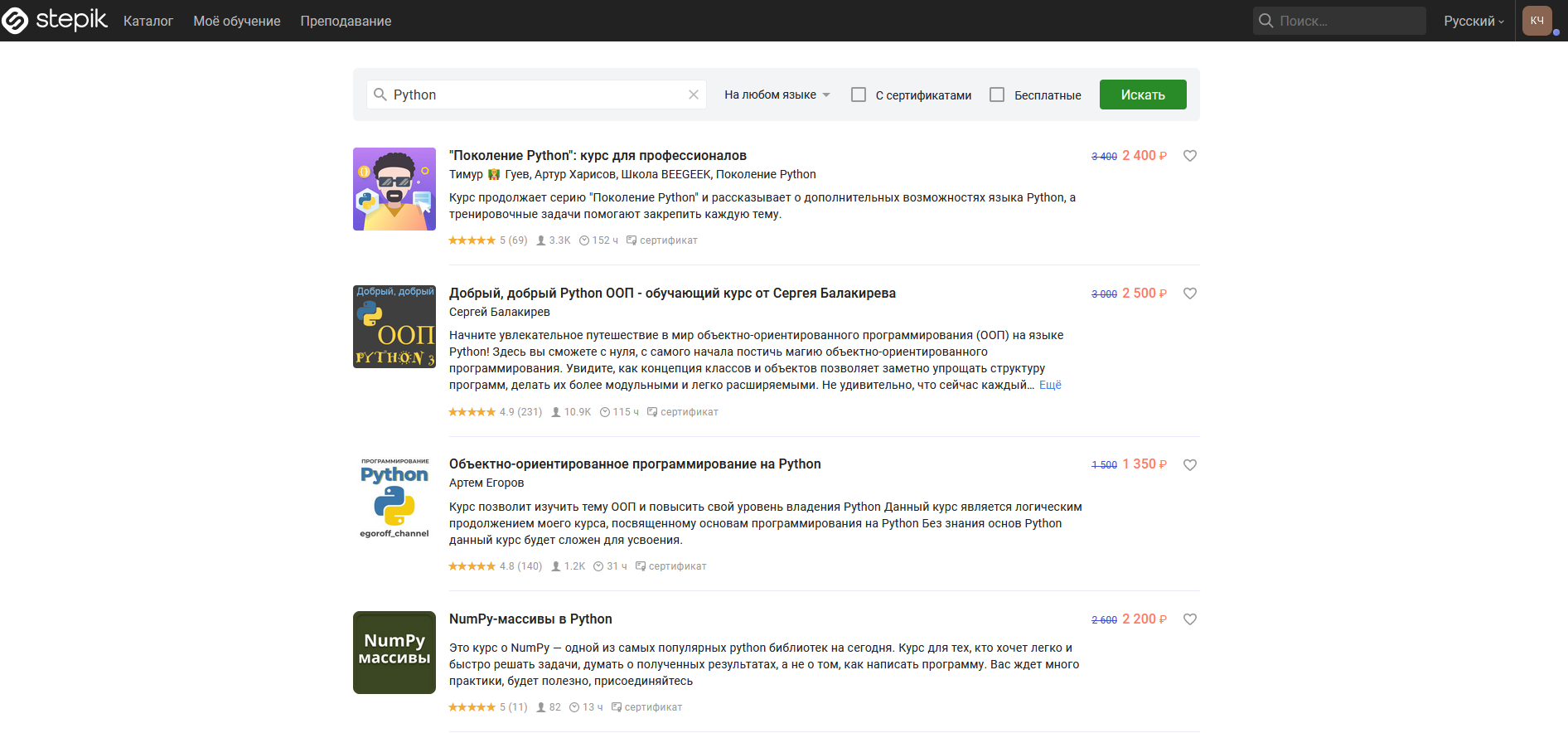


Рис. 1.1 курсы python на stepik

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).

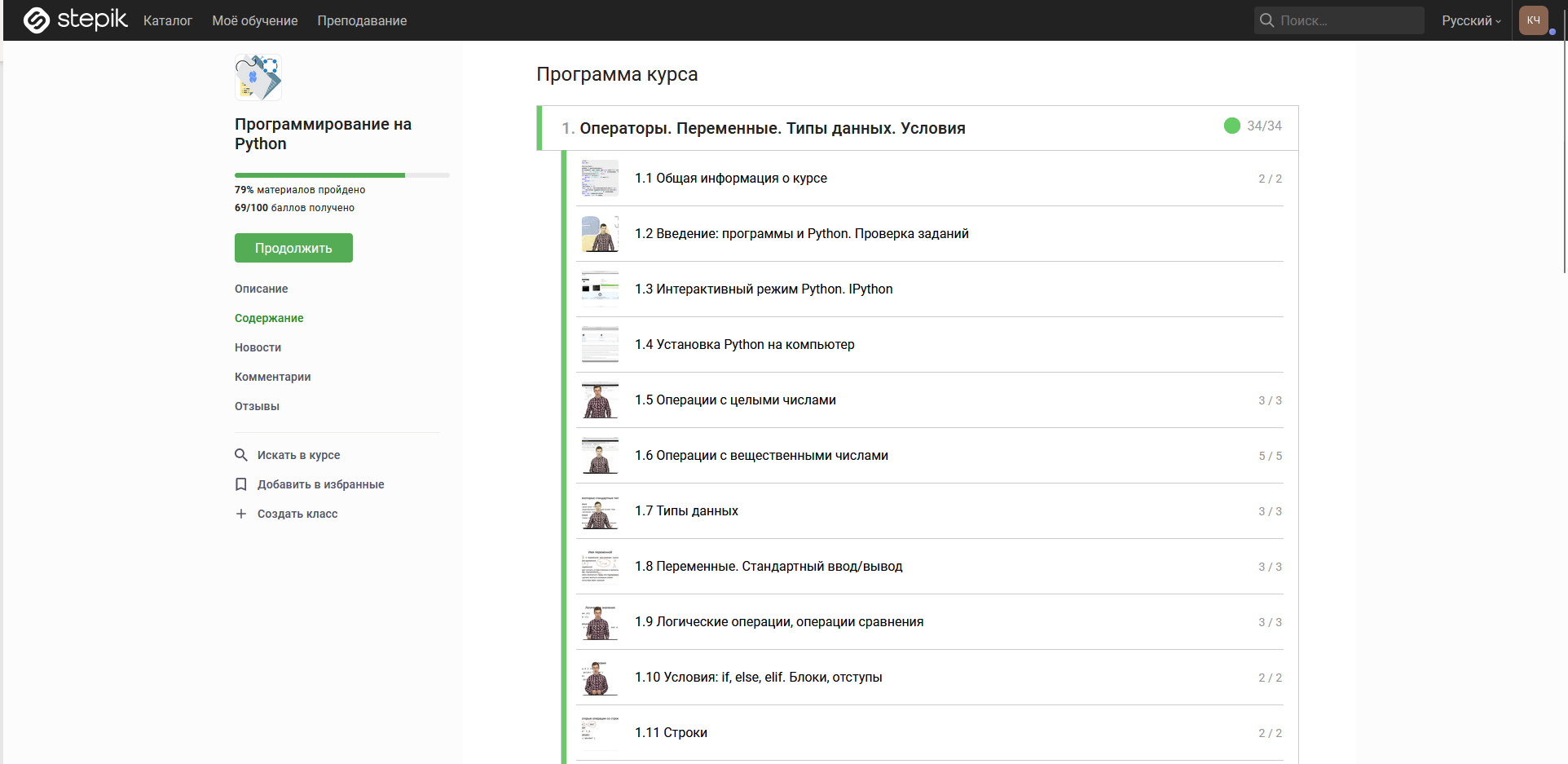


Рис. 1.2 Просмотр прогресса

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

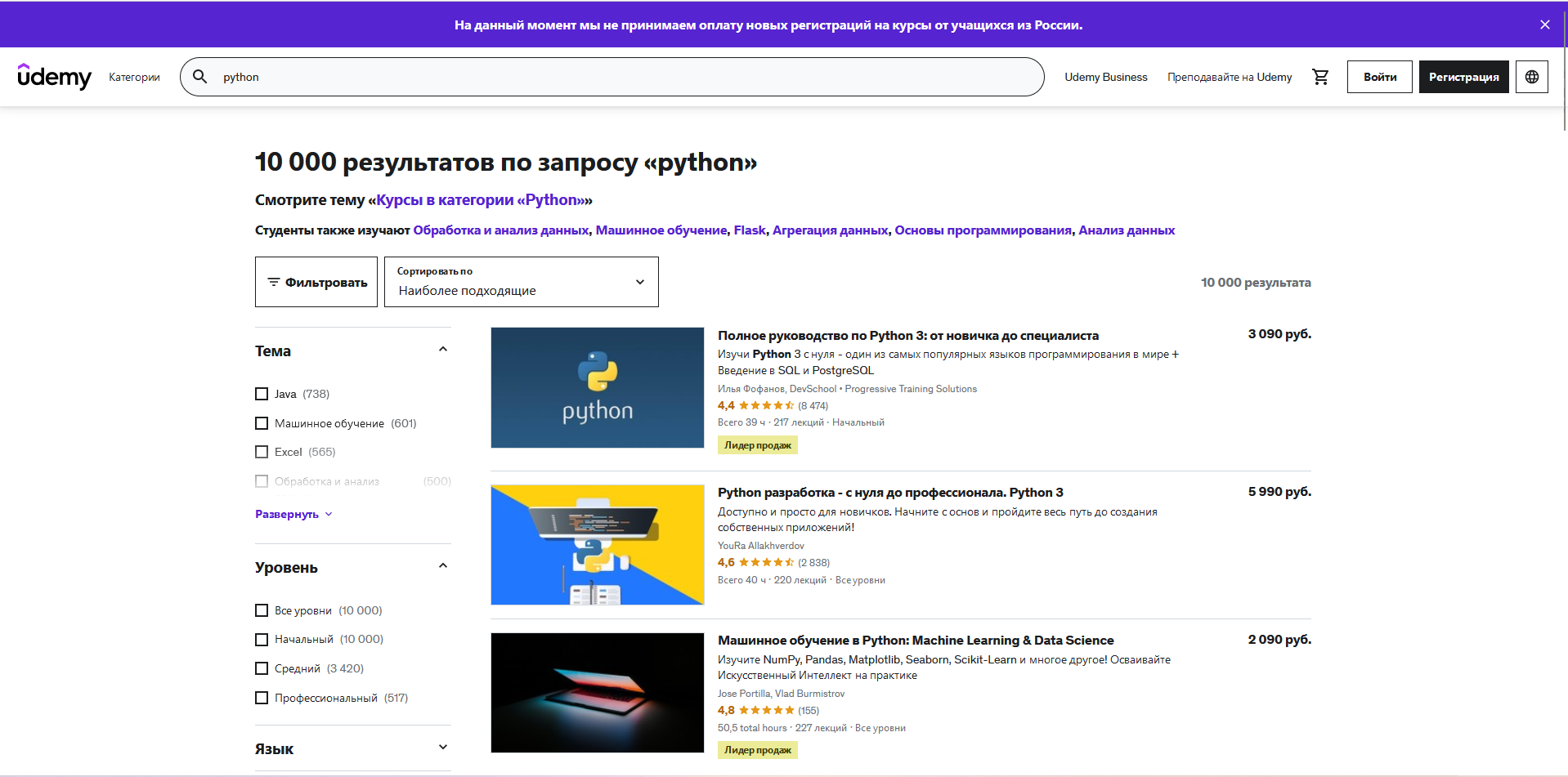


Рис. 1.3 курсы python на Undemy

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

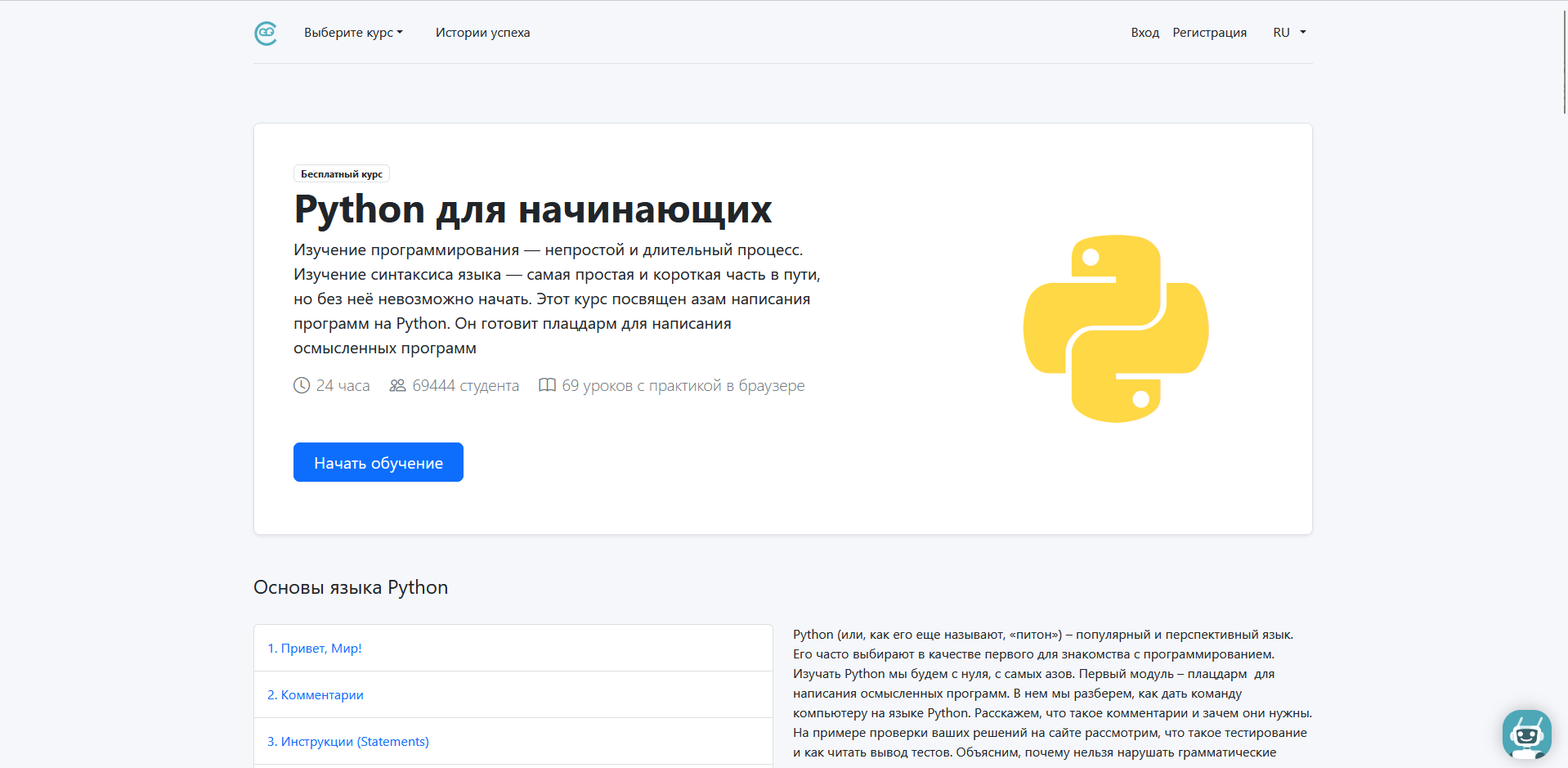


Рис. 1.4 курс python на code-basics.com

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

1.4.1 Бизнес-требования

Функциональность:

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

Безопасность:

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

Масштабируемость и производительность:

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

Адаптивность и доступность:

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

1.4.2 Пользовательские требования

Регистрация и авторизация:

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

Обучение и учебные материалы:

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

Мониторинг и оценка прогресса:

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

Доступность и адаптивность:

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

1.4.3. Функциональные требования

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

1.4.4. Нефункциональные требования

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

1.4.5. Ограничения

Бюджетные ограничения:

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

Ограничения функциональности:

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

Ограничения безопасности:

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

Ограничения по платформе и технологиям:

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

## 1.5. Программные средства разработки

Для разработки будут использоваться следующие программные средства:

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

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

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

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

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

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

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

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

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

## 1.6. 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

2.1.1 Общая структура проекта

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

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

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

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

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

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

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

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

Общая архитектура представлена на рисунке 2.1

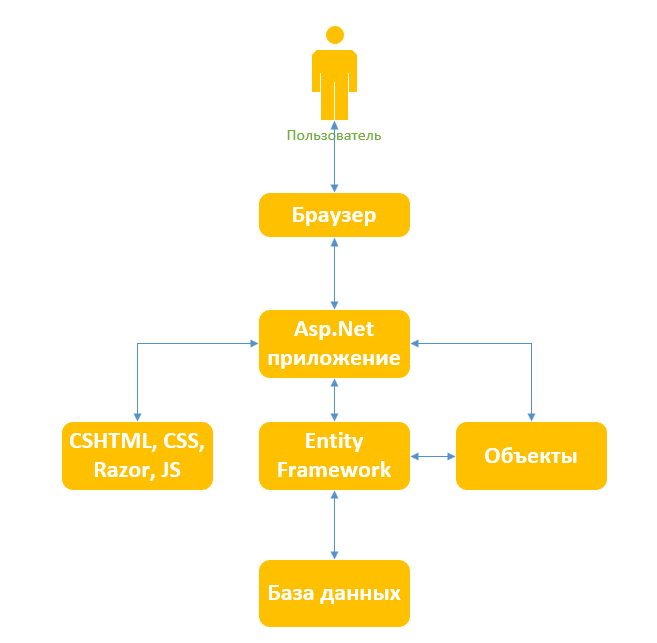


Рисунок 2.1 Общая схема взаимодействия объектов

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

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

* + 1. Диаграммы классов:

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

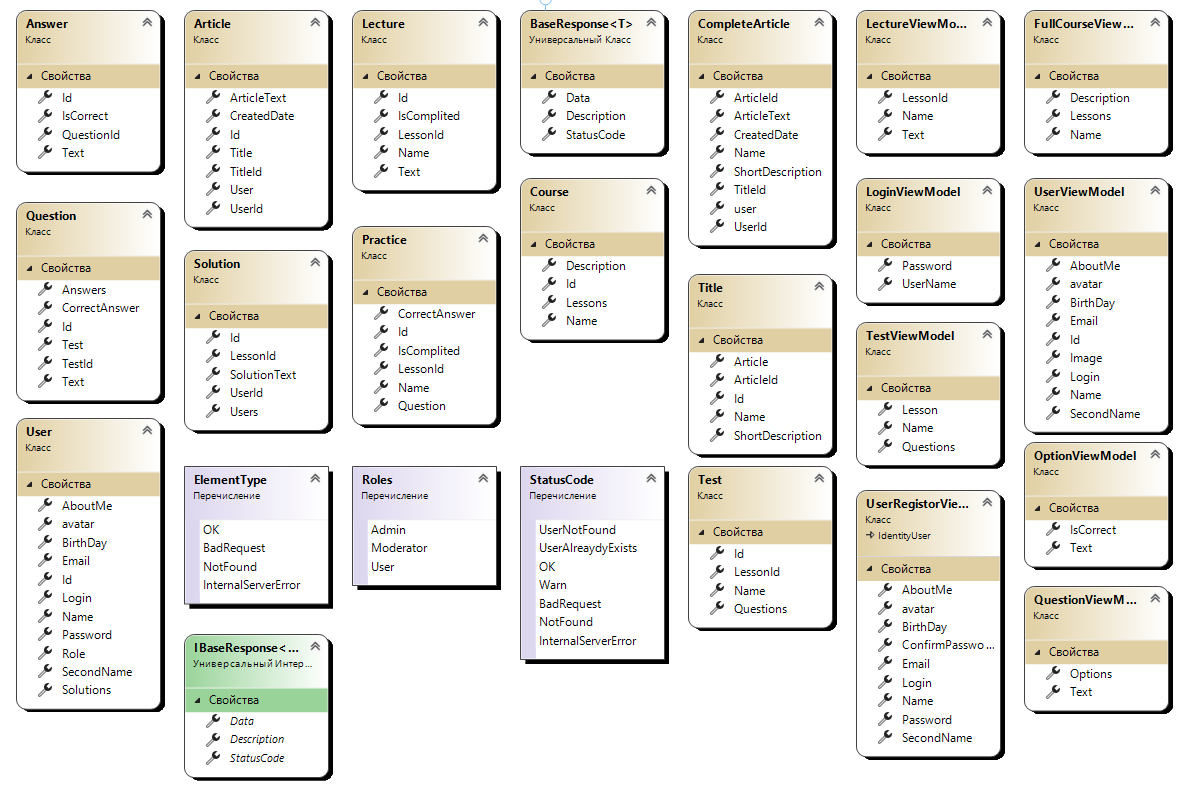


Рисунок 2.2 Диаграмма классов библиотеки Domain

В библиотеке Domain содержатся следующие модели:

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

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

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

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

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

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

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

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

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

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

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

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

Вот описания вью моделей, используемых в приложении:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

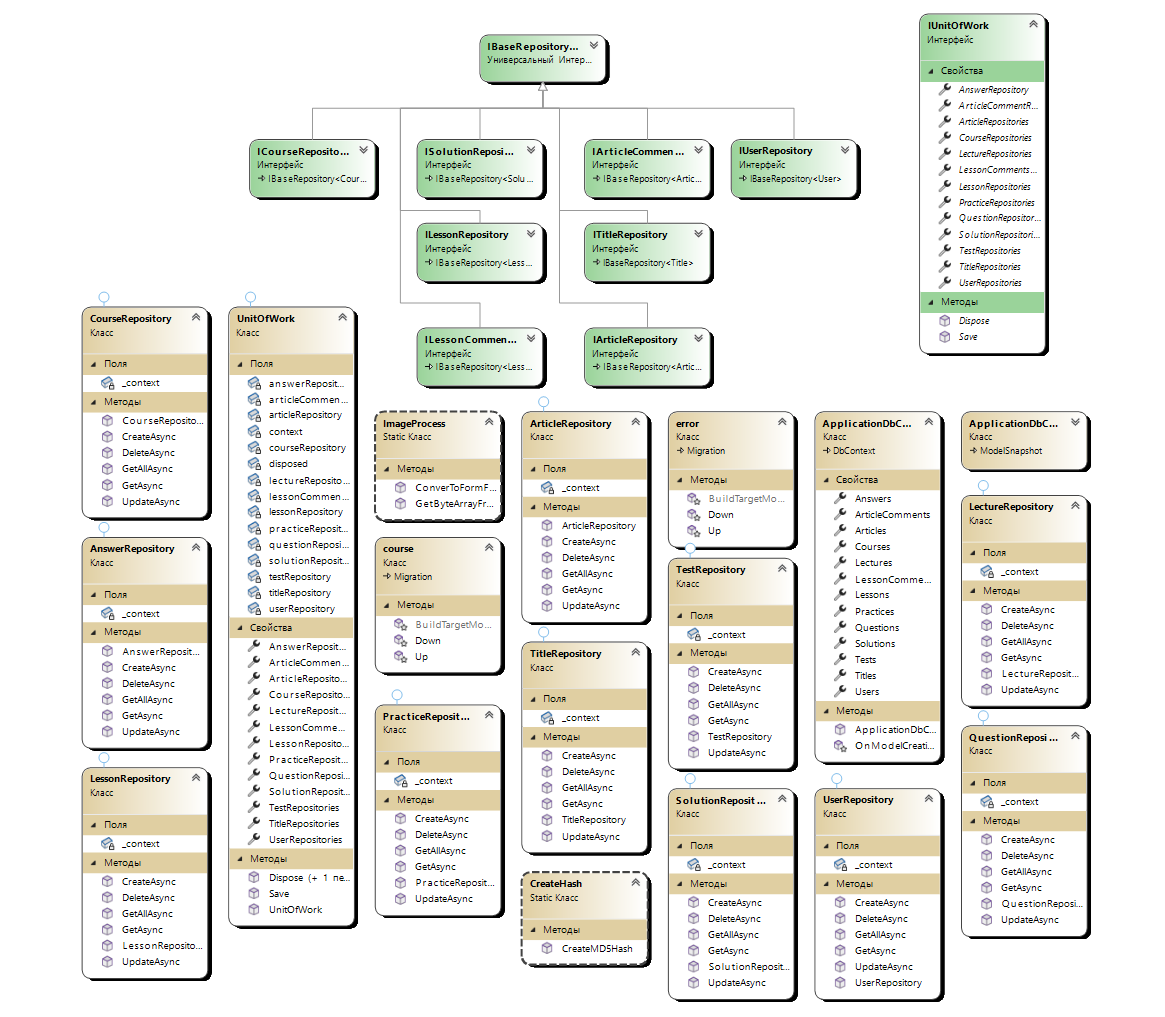


Рисунок 2.3 Диаграмма классов библиотеки Domain

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

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

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

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

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

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

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

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

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

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

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

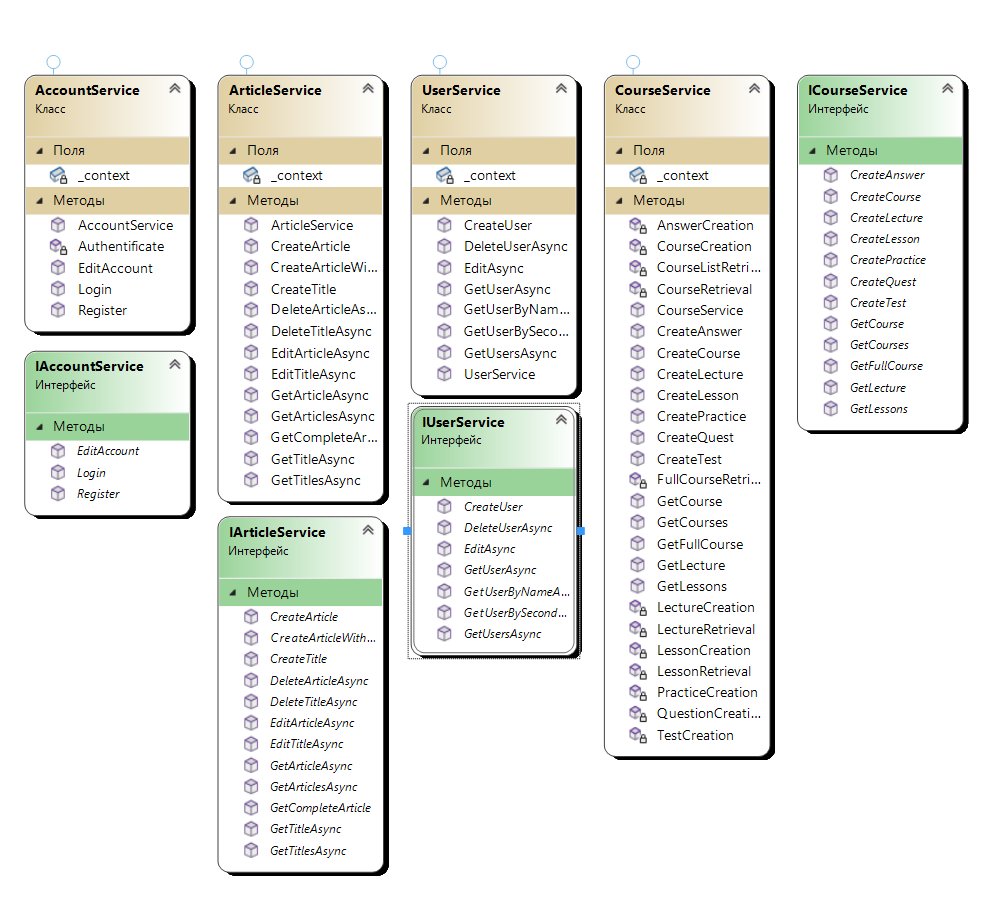


Рисунок 2.4 Диаграмма классов библиотеки Service

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

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

## 2.2. Моделирование основных сценариев системы

2.2.1 Система регистрации пользователя

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

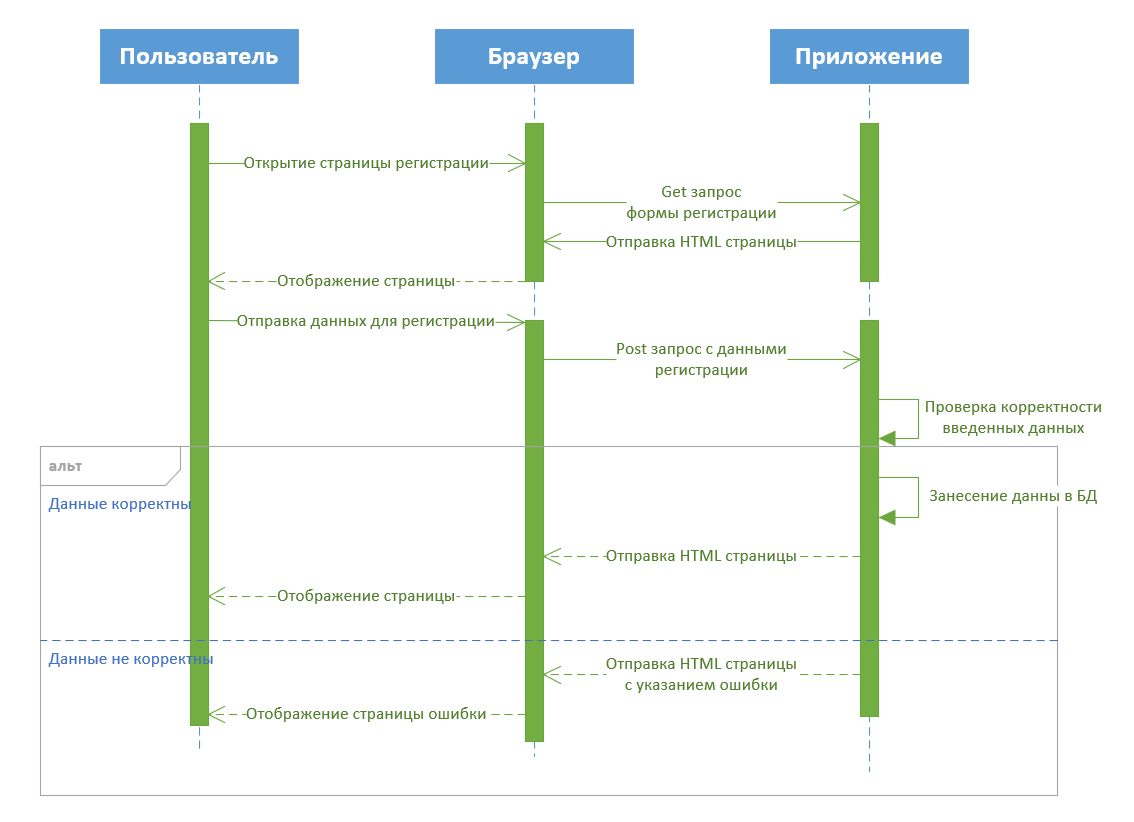


Рисунок 2.5 Диаграмма последовательности регистрации

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

Actors:

Пользователь: Человек, который желает зарегистрироваться в системе.

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

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

Шаги и взаимодействия:

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

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

2.2.2 Система создания курса

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

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

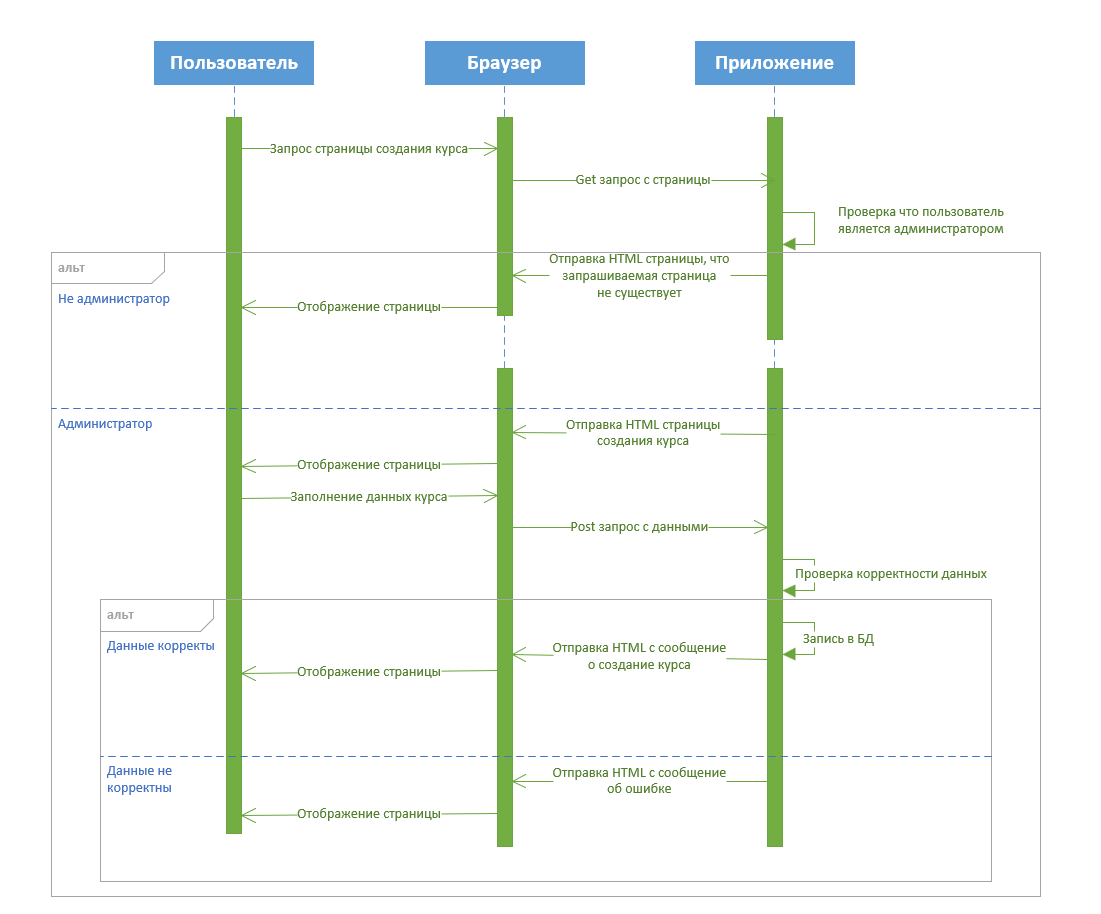


Рисунок 2.6 Диаграмма последовательности создания курс

Actors:

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

Шаги и взаимодействия:

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

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

2.2.3 Система прохождения теста

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

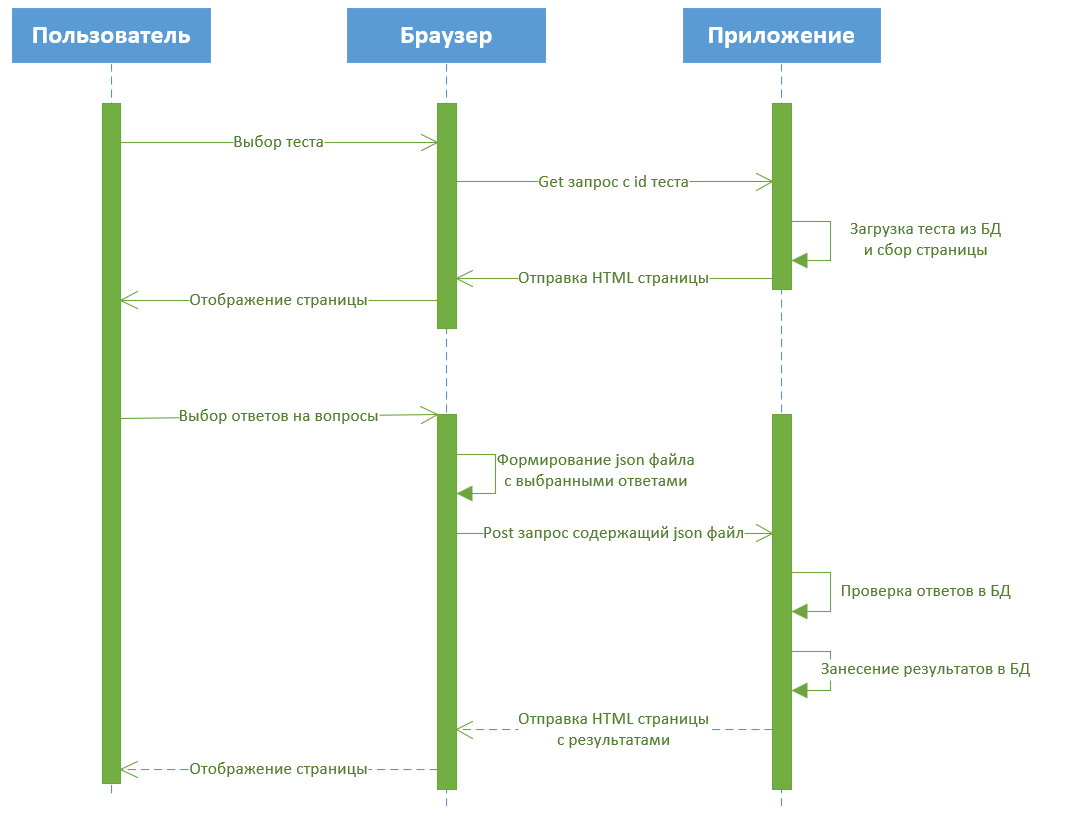


Рисунок 2.6 Диаграмма последовательности прохождение теста

Actors:

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

Шаги и взаимодействия:

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

2.2.4 Диаграмма деятельности создания урока

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

Ниже приведено описание элементов, которые представлены на диаграмме:

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

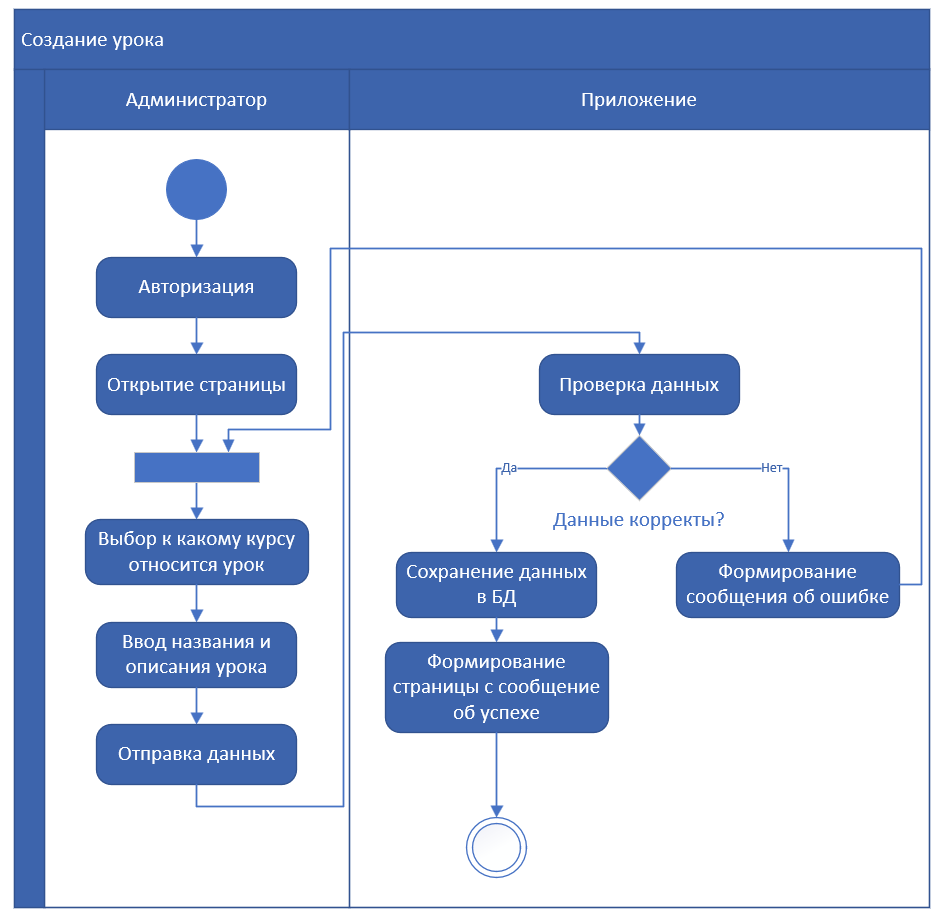


Рисунок 2.7 создание урока

Процесс создания урока представлен на рисунке 2.7.

2.2.5 Диаграмма деятельности создания теста

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

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

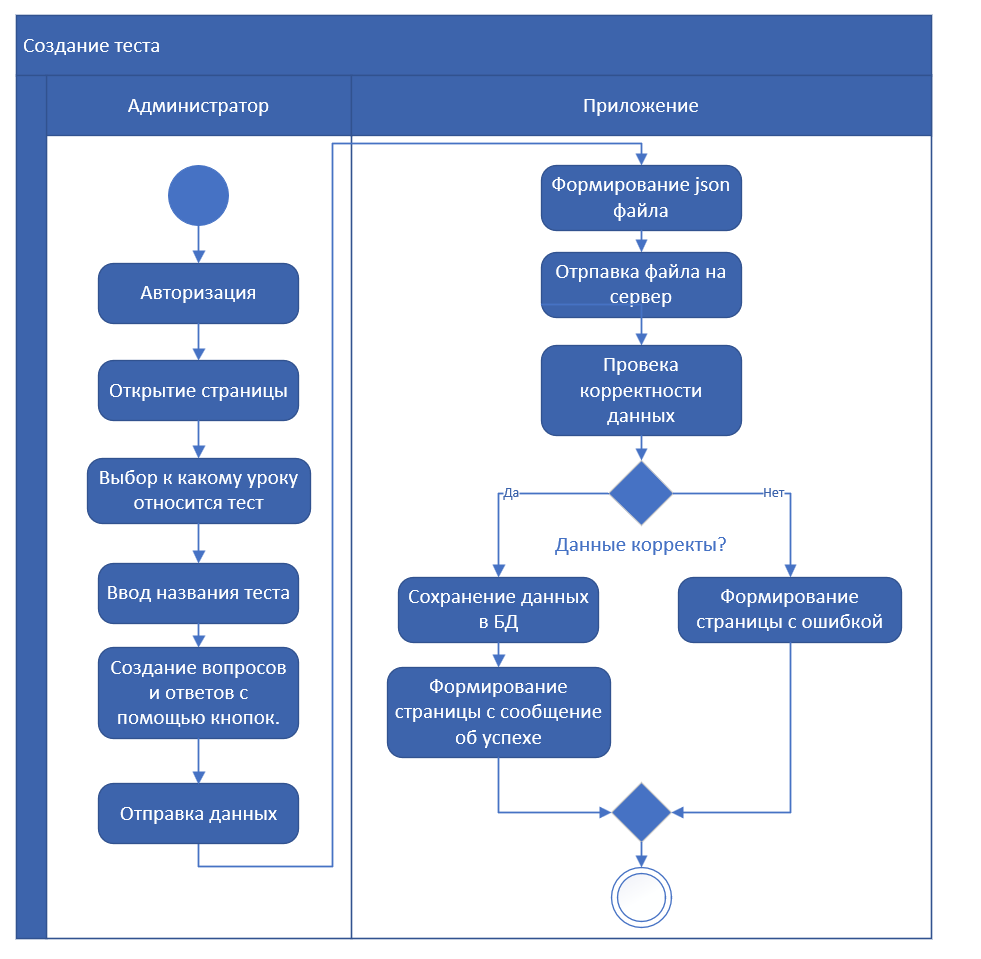


Рисунок 2.8 создание теста

Пользователь начинает процесс создания теста следующим образом:

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

2.2.6 Диаграмма деятельности: регистрация пользователя

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

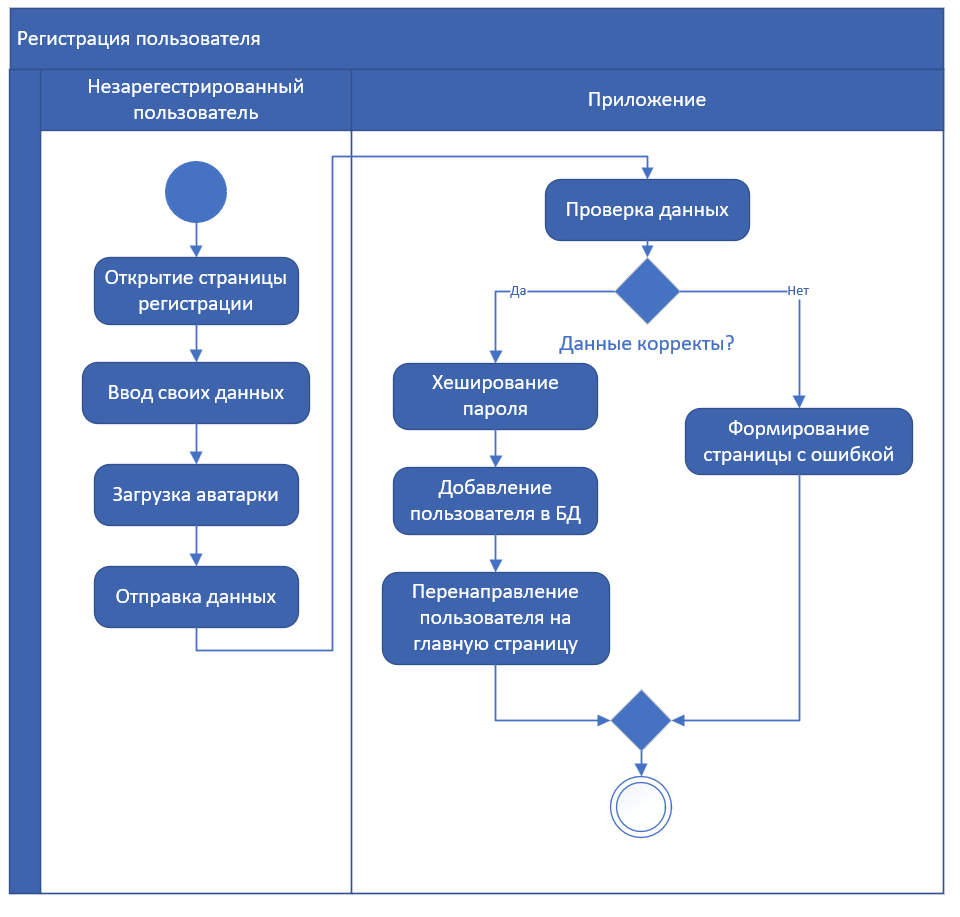


Рисунок 2.9 Регистрация пользователя

Шаги для регистрации пользователя:

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

2.2.7 Проектирование вариантов использования

Для проектирования вариантов использования был выбран инструмент Microsoft visio.

Не зарегистрированный пользователь может выполнять следующие действия:

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

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

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

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

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

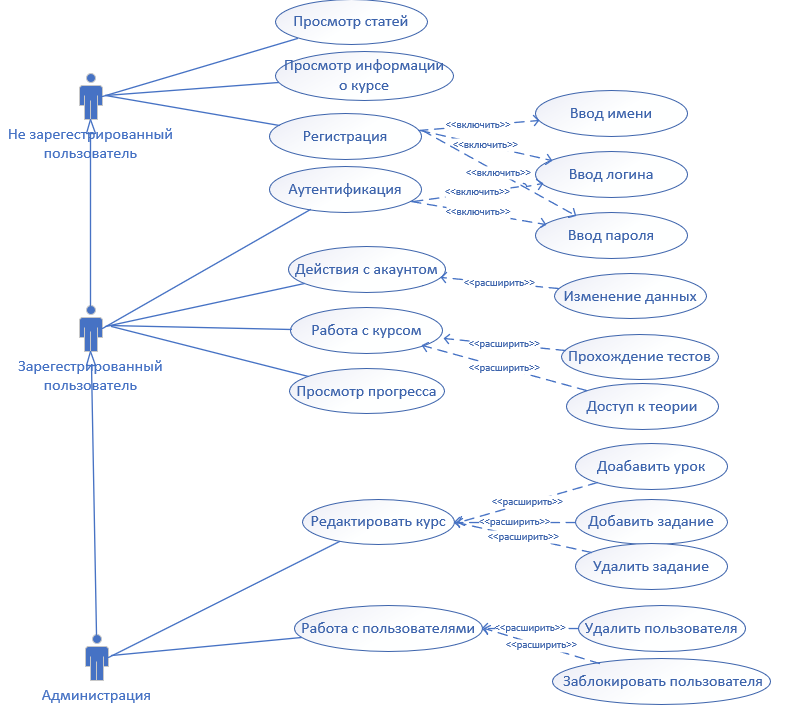


Рисунок 2.10 варианты использования

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

## 2.2. Проектирование графического интерфейса пользователя

* + 1. **Web API**

Для создания Web API была выбрана архитектура REST (Representational state transfer), в которой каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

REST основывается на протоколе HTTP. Действие над данными задается с помощью методов: GET (получить), POST (добавить, изменить, удалить), PATCH (обновить), DELETE (удалить).

Обмен данными происходит с помощью HTTP-сообщений, которые делятся на запросы (от клиента к серверу) и ответы (от сервера к клиенту).

**Структура HTTP-запроса:**

1. стартовая строка — описывает метод запроса (GET, POST, PATCH, DELETE) и цель запроса (URL запрашиваемого ресурса) и версию HTTP;
2. набор HTTP-заголовков — определяет запрос и состоит из основных заголовков (относятся к сообщению в целом), заголовков запроса (уточняют запрос) и заголовков сущности (относятся к телу запроса);
3. пустая строка, которая информирует о том, что вся мета-информация отправлена;
4. тело запроса, содержащее пересылаемые данные. Наличие тела и его размер определяют стартовая строка и заголовки HTTP.

**Структура HTTP-ответа:**

1. стартовая строка — описывает версию HTTP, статус ответа (код состояния и пояснение);
2. набор HTTP-заголовков — определяет запрос и состоит из основных заголовков (относятся к сообщению в целом), заголовков ответа (сообщают дополнительную информацию о сервере) и заголовков сущности (относятся к телу ответа);
3. пустая строка, которая информирует о том, что вся мета-информация отправлена;
4. тело запроса, содержащие пересылаемые данные. Наличие тела и его размер определяют стартовая строка и заголовки HTTP.

Формат передачи данных — JSON (текстовый формат обмена данными, основанный на JavaScript). Данные могут передаваться как JSON-объект и как JSON-массив.

JSON-объект — это множество пар «ключ:значение», которое заключено в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.

Значения ключа могут быть любыми:

1. число;
2. строка;
3. массив;
4. другой объект.

Внутри JSON-массива могут также могут быть числа, строки, JSON-объекты, а также данные разных типов.

Все передаваемые данные следуют 4 правилам Well Formed JSON:

1. данные записаны как пары «ключ:значение»;
2. пары отделяются друг от друга запятой;
3. JSON-объект находится внутри фигурных скобок «{}»;
4. JSON-массив находится внутри квадратных скобок «[]»;

**HTTP-ответ от сервера** может содержать следующие статусы:

1. 200 OK — «Успешно». Запрос успешно обработан;
2. 201 Created — «Создано». Запрос успешно выполнен, в результате создан ресурс;
3. 304 Not Modified — «Не модифицировано». Запрошенный ресурс не был изменен и можно продолжать использовать кэшированную версию ответа;
4. 401 Unauthorized — «Неавторизованно». Для получения положительного ответа необходима аутентификация;
5. 403 Forbidden — «Запрещено». Нет прав доступа на получение данных;
6. 404 Not Found — «Не найдено». Запрашиваемый ресурс не найден;
7. 500 Internal Server Error — «Внутренняя ошибка сервера». На стороне сервера произошла ошибка;
8. 504 Gateway Timeout — «Тайм-аут шлюза». Не удалось получить ответ в вовремя
   * 1. **Уровень представления**

Разработка клиент-составляющей проекта велась на библиотеке ReactJS и написана на расширении JSX с использованием таких дополнительных технологий, как react-router-dom, axios, react-cookie и fingerprintJS.

**ReactJS** — библиотека языка JS для разработки пользовательский интерфейсов программных продуктов.

**JSX** — расширение языка JS, позволяющее использовать HTML-разметку непосредственно в JavaScript-коде.

**react-router-dom** — библиотека, которая позволяет организовать навигационную часть программного продукта: с ее помощью оформляется код перемещения/движения пользователя по виртуальному пространству, не перезагружая страницу при этом.

**axios** — библиотека, через которую появляется возможность взаимодействия с технологией Web API.

**react-cookie** — библиотека для взаимодействия с cookie-файлами пользователя.

**fingerprintJS** — библиотека, позволяющая получать данные о сессии пользователя; используется для дополнительной защиты персональных данных.

* + 1. **Уровень приложения**

Уровень приложения реализован с помощью NestJS (фреймворк Node.js) и написан на языке TypeScript.

**NestJS** — фреймворк Node.js, построенный на идеях IoC/DI, модульного дизайна и декораторов.

**Node.js** — это программная платформа, которая основана на движке V8, и превращает JavaScript из узкоспециализированного языка в язык общего назначения.

**TypeScript** — язык программирования, позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript. В TypeScript реализована статическая типизация, которая отсутствует в JS.

Большое количество функционала реализовано с помощью декораторов, что существенно повышает удобство разработки, читаемость кода.

**Декоратор в NestJS** — структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Они позволяют декларативно описывать логику, модифицировать поведение классов, их свойств, аргументов и методов.

Приложение состоит из главного модуля и второстепенных модулей, каждый из которых выполняет определенные задачи.

Внутри модуля могут присутствовать следующие классы:

1. module;
2. controller;
3. service;
4. множество dto;
5. множество entity.

**Module** — класс, в котором описаны импортируемые в данный модуль другие модули, а также используемые в данном модуле entity.

**Controller** — класс, в котором определены уникальные идентификаторы ресурсов, находящихся в модуле. Контроллер разбирает URL строку запроса, определяет какой ресурс каким методом запрошен (GET, POST, PATCH, DELETE) и выделяет переданные параметры. В случае, если указанный ресурс найден, данные передаются в методы, находящиеся в классах Service.

Переданные параметры проходят валидацию, проверку на соответствие условиям, с помощью находящихся внутри каждого модуля DTO (data transfer object).

**Service** — класс, который состоит из функций, каждая из которых реализует обработку данных. В данном классе происходит взаимодействие с уровнем данных (базой данных и файловым сервером). В конечном итоге возвращается результат обработки.

**DTO** — классы, которые используются для передачи данных между подсистемами приложения. Валидация данных в DTO происходит с помощью декораторов, определенных в библиотеке class-validator:

1. @IsEmpty() — значение должно быть пустым;
2. @IsNotEmpty() — значение не должно быть пустым;
3. @IsOptional() — значение не обязательно должно быть (при отсутствии значения последующие декораторы игнорируются);
4. @IsEmail() — значение должно быть валидным email;
5. @IsString() — значение должно быть строковым;
6. @IsUrl() — значение должно быть унифицированным указателем ресурса;
7. @IsDate() — значение должно быть датой;
8. @IsInt() — значение должно быть целым числом;
9. @IsArray() — значение должно быть массивом;
10. @IsObject() — значение должно быть объектом;
11. @ArrayNotEmpty() — в массиве должен быть как минимум один элемент;
12. @Length(min, max) — длина строкового значения в пределах, указанных в параметрах;
13. @MinLength() — минимальная длина строкового значения;
14. @MaxLength() — максимальная длина строкового значения;
15. @Matches() — строковое значение соответствует шаблону.

**Entity** — сущности, которые определяют структуру базы данных, свойства полей, связи между полями. Сущности описываются в виде класса, один класс определяет структуру одной таблицы и ее связей с другими таблицами, структура которых может быть описана в entities других модулей.

* + 1. **Уровень данных**

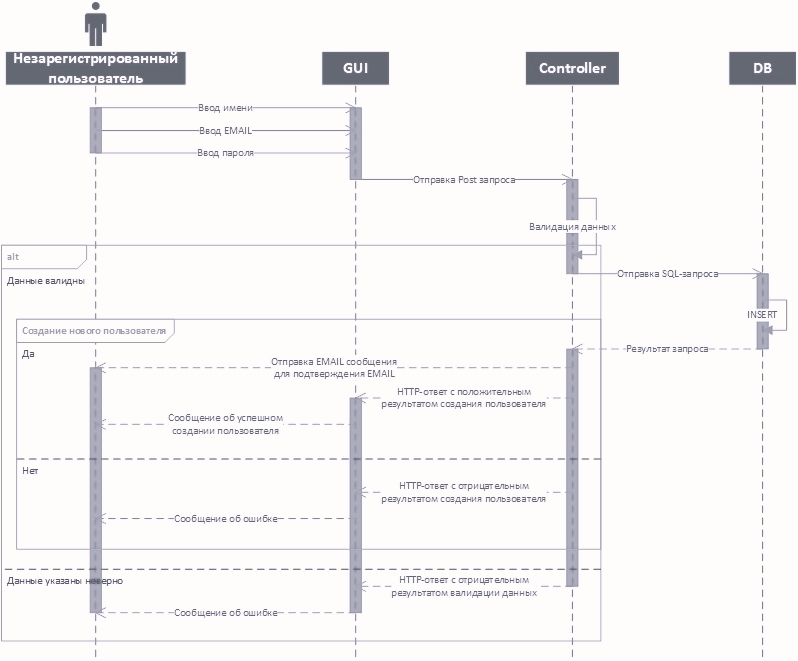
Уровень данных представлен сервером баз данных PostgreSQL и файловым сервером Object Storage от Yandex.Cloud.

В базе данных для обеспечения целостности данных применяются первичные и внешние ключи в таблицах.

В файловом сервере Object Storage от Yandex.Cloud создана папка «artart», в которой хранятся файлы, необходимые для проекта. Взаимодействие с файловым сервером происходит посредством API AWS S3.

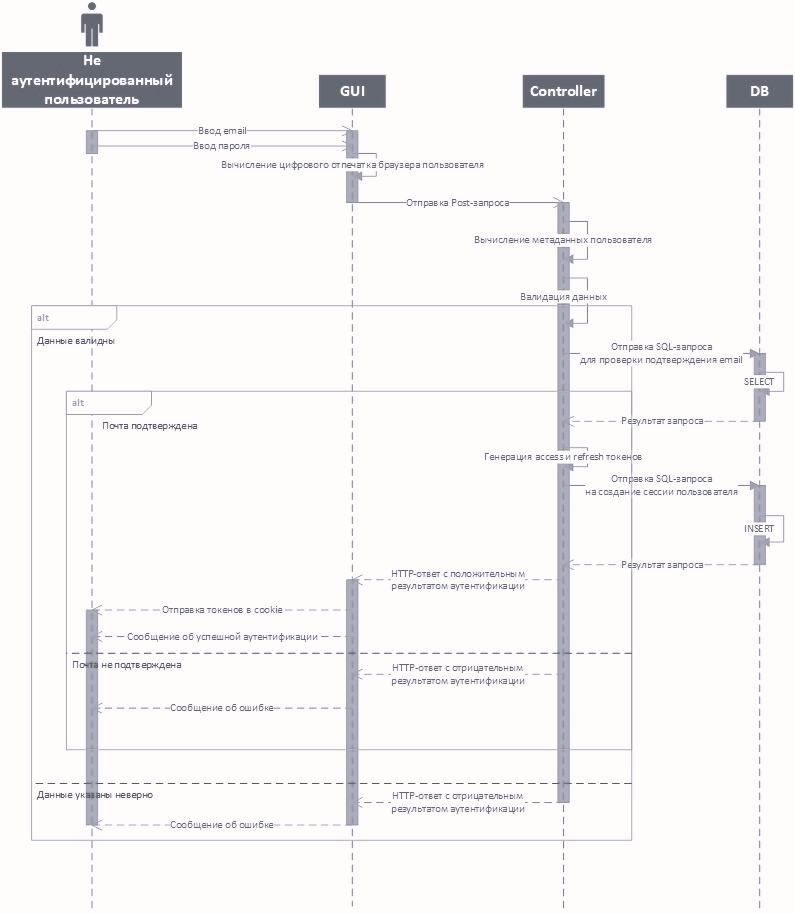
* 1. **Моделирование основных сценариев системы idef, uml**
     1. **Диаграммы последовательности**

На рисунке 2.2 изображена диаграмма последовательности для прецедента «Регистрация». На диаграмме представлены следующие актеры: незарегистрированный пользователь, GUI (графический интерфейс пользователя), controller (уровень приложения) и DB (база данных).



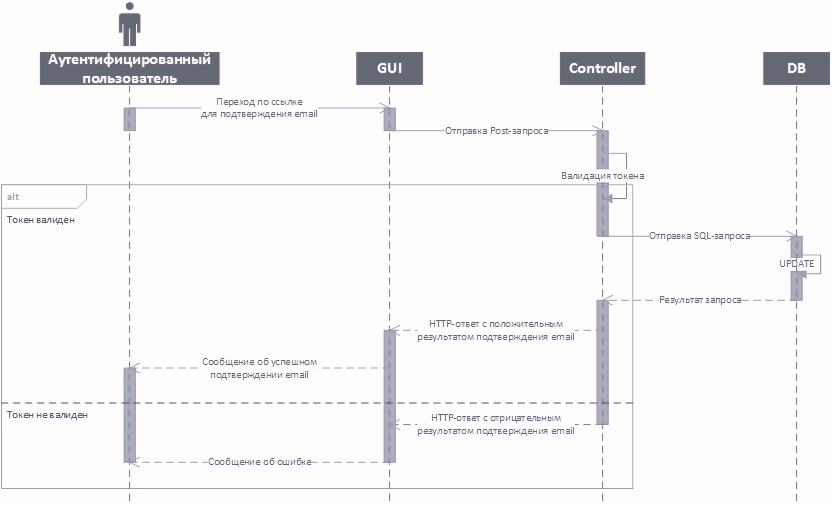
*Рис. 2.2 Диаграмма последовательности прецедента «Регистрация»*

На рисунке 2.3 изображена диаграмма последовательности для прецедента «Аутентификация». На диаграмме представлены следующие актеры: не аутентифицированный пользователь, GUI (графический интерфейс пользователя), controller (уровень приложения) и DB (база данных).



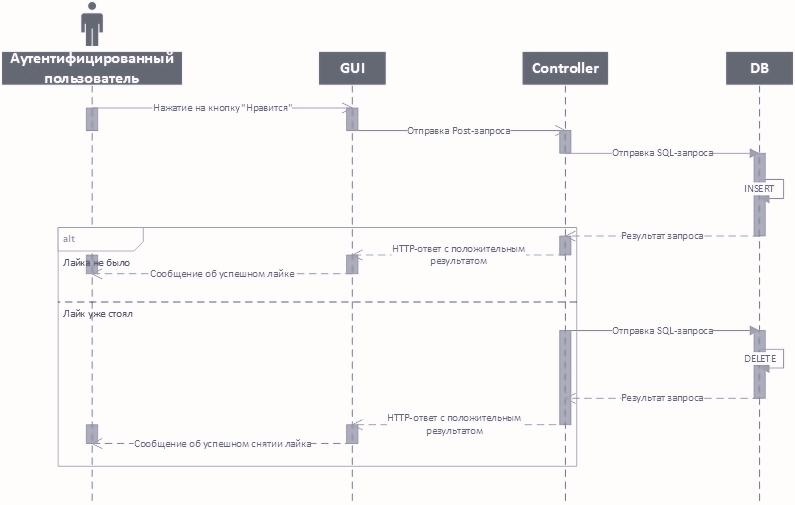
*Рис. 2.3 Диаграмма последовательности прецедента «Аутентификация»*

На рисунке 2.4 изображена диаграмма последовательности для прецедента «Подтверждение email». На диаграмме представлены следующие актеры: аутентифицированный пользователь, GUI (графический интерфейс пользователя), controller (уровень приложения) и DB (база данных).



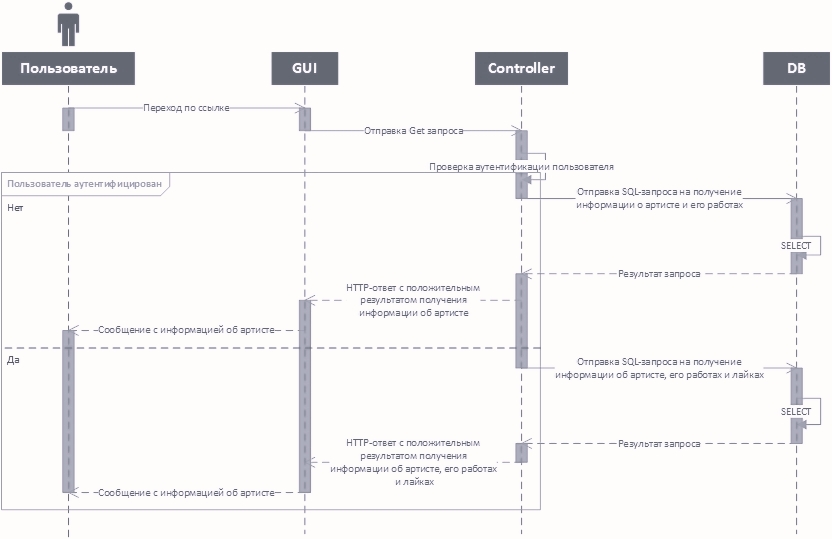
*Рис. 2.4 Диаграмма последовательности прецедента «Подтверждение email»*

На рисунке 2.5 изображена диаграмма последовательности для прецедента функции «Нравится». На диаграмме представлены следующие актеры: аутентифицированный пользователь, GUI (графический интерфейс пользователя), controller (уровень приложения) и DB (база данных).



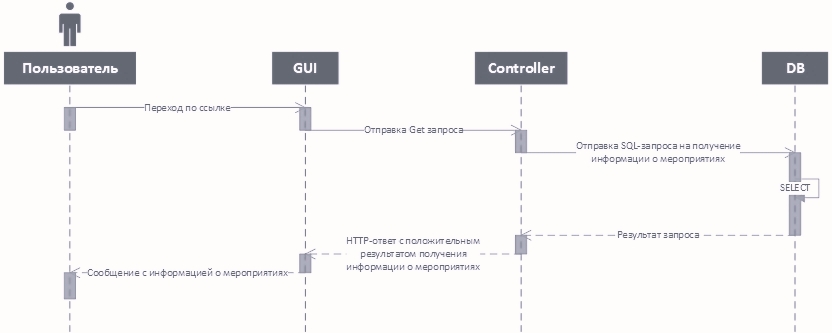
*Рис. 2.5 Диаграмма последовательности прецедента функции «Нравится»*

На рисунке 2.6 изображена диаграмма последовательности для прецедента «Получение информации об одном артисте». На диаграмме представлены следующие актеры: пользователь, GUI (графический интерфейс пользователя), controller (уровень приложения) и DB (база данных).



*Рис. 2.6 Диаграмма последовательности прецедента «Получение информации об одном артисте»*

На рисунке 2.7 изображена диаграмма последовательности для прецедента «Получение информации о мероприятиях». На диаграмме представлены следующие актеры: пользователь, GUI (графический интерфейс пользователя), controller (уровень приложения) и DB (база данных).

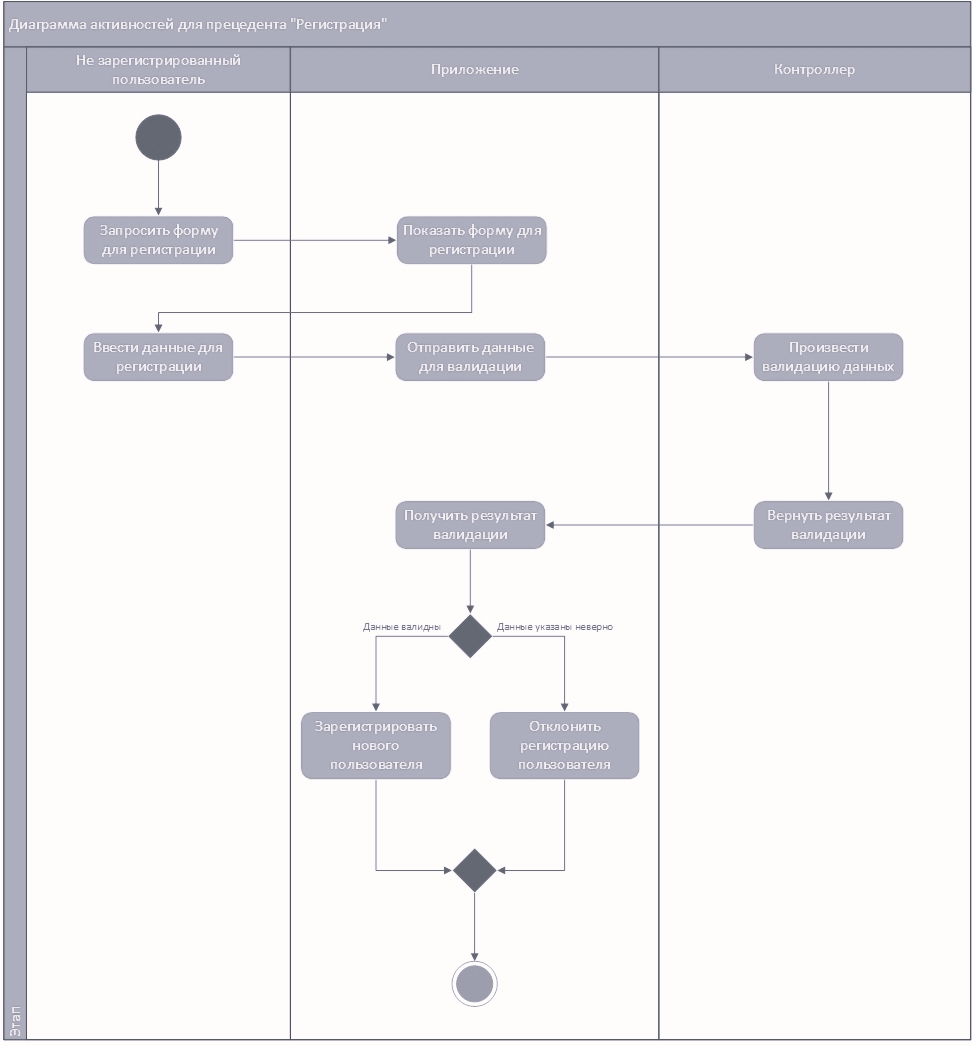


*Рис. 2.7 Диаграмма последовательности прецедента «Получение информации о мероприятиях»*

Диаграммы последовательности других прецедентов строятся по той же схеме:

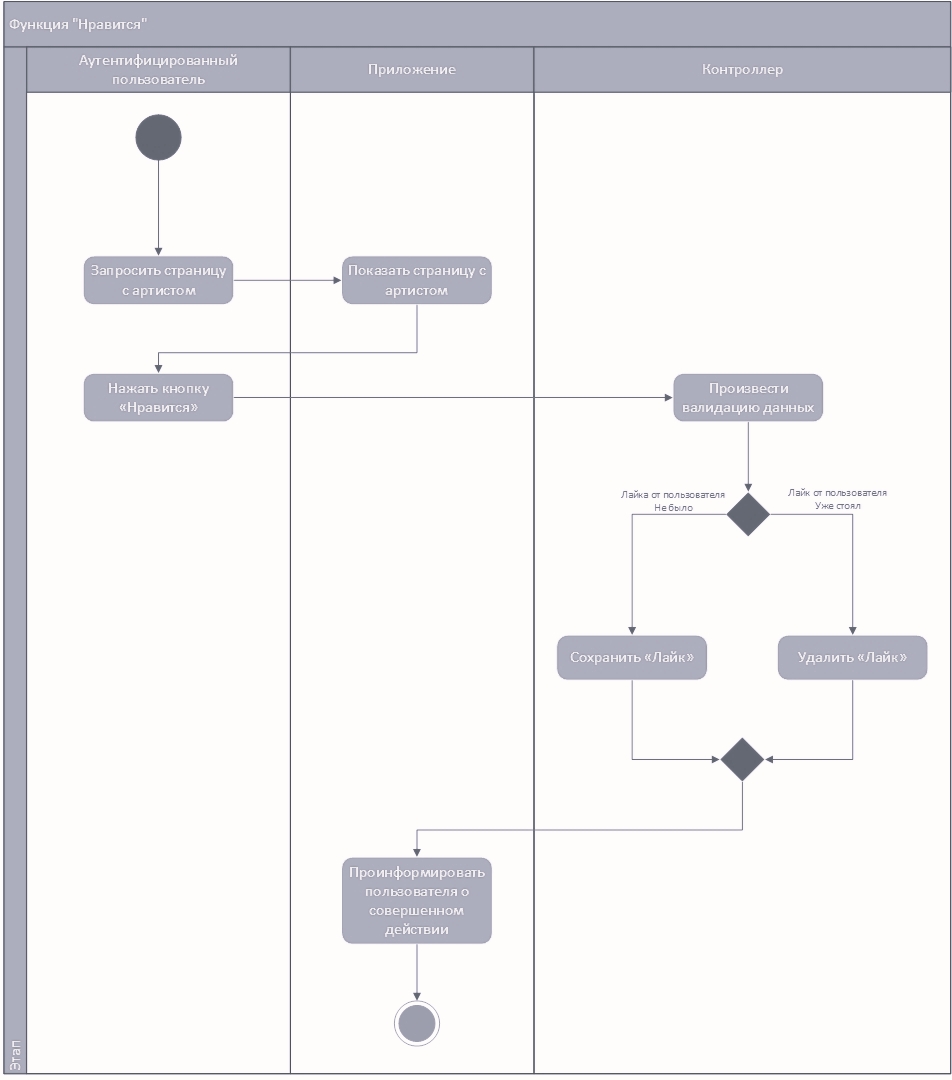
1. пользователь взаимодействует с GUI;
2. GUI отправляет HTTP-запросы на сервер;
3. на уровне приложения (controller) происходит обработка информации, обращения к БД и файловому серверу;
4. пользователю отправляется ответ или сообщение об ошибке.
   * 1. **Диаграммы активностей**

На рисунке 2.8 изображена диаграмма активностей для прецедента «Регистрация». На диаграмме представлены следующие объекты: незарегистрированный пользователь, приложение и контроллер.



*Рис. 2.8 Диаграмма активностей прецедента «Регистрация»*

На рисунке 2.9 изображена диаграмма активностей для прецедента функции «Нравится». На диаграмме представлены следующие объекты: аутентифицированный пользователь, приложение и контроллер.

****

*Рис. 2.9 Диаграмма активностей прецедента функции «Нравится»*

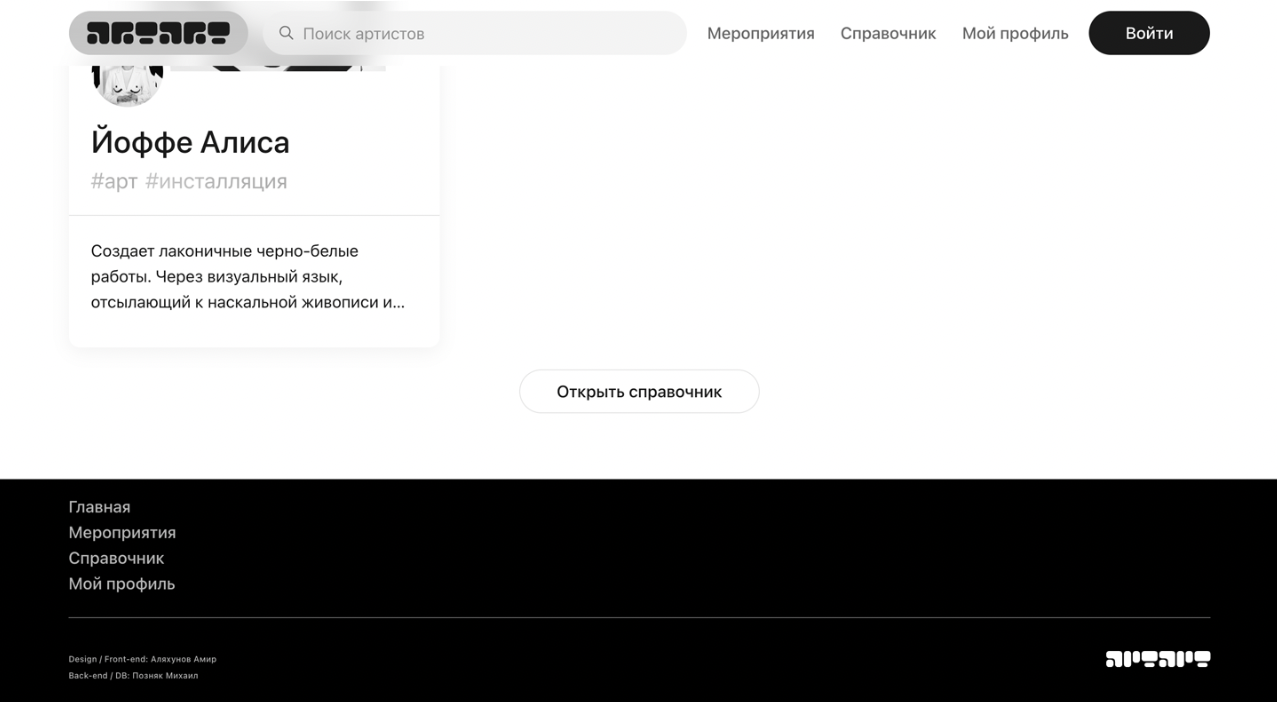
Диаграммы активностей других прецедентов строятся по той же схеме:

1. пользователь взаимодействует с приложением;
2. необходимые вычисления происходят в контроллере.
   1. **Проектирование графического интерфейса пользователя**

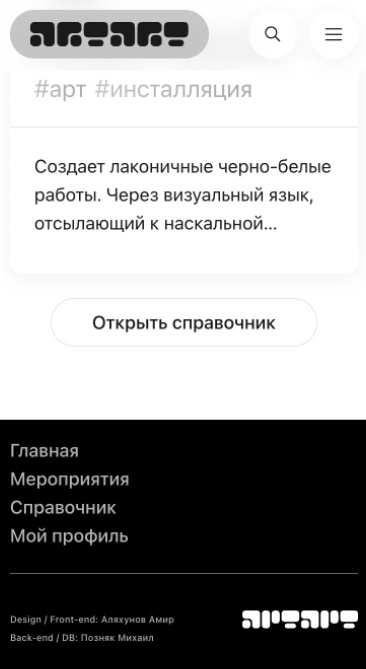
Интерфейс веб-сервиса должен быть адаптивным, корректно отображаться и функционировать на устройствах различного типа, с различным разрешением и форм-фактором экрана: монитор, планшет, смартфон. Визуальная часть программного продукта разрабатывалась в графическом редакторе Figma.

Фундамент веб-сайта (рис. 2.10-2.11) состоит из:

1. заголовка (header);
2. тела приложения (body), в котором расположен основной контент сервиса, динамически изменяющийся в зависимости от положения пользователя в системе;
3. подвала (footer), дублирующий в себе все ссылки на страницы пространства, а также дополнительную информацию о приложении.



*Рис. 2.10 Общее представление веб-сервиса — Десктоп версия*



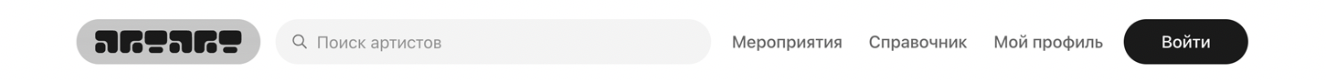
*Рис. 2.11 Общее представление веб-сервиса — Мобильная версия*

Исходя из этого можно заключить, что весь интерфейс построен на трех основных блоках.

* + 1. **Заголовок (header)**

Header приложения (рис. 2.12-2.13), в зависимости от адаптированного режима, включает в себя:

1. логотип проекта, который также является ссылкой на домашнюю (главную) страницу веб-сервиса;
2. поле активации формы поиска/кнопку активации формы поиска артистов в приложении;
3. навигационную панель, состоящую из кнопки входа-выхода из аккаунта, ссылок на страницы веб-сервиса/кнопку открытия меню приложения.



*Рис. 2.12 Header — Десктопная версия*



*Рис. 2.13 Header — Мобильная версия*

Благодаря адаптивности веб-приложения появляется возможность представления одного и того же функционала в разном виде, более удобном в применении на разных типах устройств.

* 1. **Проектирование базы данных** 
     1. **Структура и описание таблиц**

Информация, отображаемая и обрабатываемая веб-сервисом, хранится в базе данных PostgreSQL.

* + - 1. **Таблица user**

Содержит информацию о пользователях веб-приложении. Структура таблицы, а также типы данных полей приведены на рисунке 2.46.

*Рис. 2.46 Таблица user*

Назначение полей:

1. id — уникальный идентификатор пользователя, генерируется автоматически, используется для связи с другими таблицами;
2. name — имя пользователя, должно быть уникальным;
3. email — email пользователя, должен быть уникальным;
4. hash — зашифрованный пароль пользователя;
5. createdAt — дата создания записи;
6. updatedAt — дата обновления записи;
7. role — роль пользователя (user или admin);
8. userpic — ссылка на аватар пользователя;
9. isEmailConfirmed — поле, которое определяет подтвержден ли пользователем email.
   * + 1. **Таблица person**

Содержит информацию об артистах. Структура таблицы, а также типы данных полей приведены на рисунке 2.47.

*Рис. 2.47 Таблица person*

Назначение полей:

1. id — уникальный идентификатор артиста, генерируется автоматически, используется для связи с другими таблицами;
2. fullname — полное имя артиста;
3. pseudonym — псевдоним артиста;
4. description — описание артиста;
5. personpic — ссылка на аватар артиста;
6. createdAt — дата создания записи;
7. updatedAt — дата обновления записи;
8. views — счетчик просмотров карточки артиста;
9. likes — счетчик лайков карточки артиста;
10. previewWork — ссылка на вторую картинку артиста;
11. socNetworks — ссылки на соцсети артиста.
    * + 1. **Таблица art**

Содержит информацию о работах артистов. Структура таблицы, а также типы данных полей приведены на рисунке 2.48.

*Рис. 2.48 Таблица art*

Назначение полей:

1. id — уникальный идентификатор работы, генерируется автоматически, используется для связи с другими таблицами;
2. title — название работы;
3. description — описание работы;
4. personid — id автора работы;
5. pic — ссылки на изображения работы в формате фото;
6. video — ссылки на изображения работы в видео формате.
   * + 1. **Таблица event**

Содержит информацию о мероприятиях. Структура таблицы, а также типы данных полей приведены на рисунке 2.49

*Рис. 2.49 Таблица event*

Назначение полей:

1. id — уникальный идентификатор мероприятия, генерируется автоматически, используется для связи с другими таблицами;
2. title — название мероприятия;
3. description — описание мероприятия;
4. createdAt — дата создания записи;
5. updatedAt — дата последнего обновления записи,;
6. pics — картинки мероприятия;
7. startDate — дата начала проведения мероприятия;
8. endDate — дата окончания проведения мероприятия;
9. place — место проведения мероприятия.