**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.

PATCH — метод, направленный на частичное изменение указанного ресурса.

DELETE — метод, удаляющий заданный ресурс.

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

JWT (JSON Web Token) — это открытый стандарт для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.

JS (JavaScript) — мультипарадигменный язык программирования.

TS (TypeScript) — язык программирования, позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript.

JSX – расширение языка JavaScript.

DTO (Data Transfer Object) — один из шаблонов проектирования, используется для передачи данных между подсистемами приложения.

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.

ORM (Object-Relational Mapping) — технология программирования, связывающая базы данных с концепциями объектно-ориентированных языков программирования.

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.

CLI — Интерфейс командной строки.

**ВВЕДЕНИЕ**

Глобальные ограничения, с которыми столкнулись представители отечественного искусства из сферы музыки, театра, художественного промысла, поставили в приоритет вопрос о большей принципиальной важности популяризации и распространения культурного наследия, в том числе не только физические, но и нематериальные ценности.

Просветительская деятельность в области творчества, одним из основных способов осуществления которой является посещение учреждений культурно одухотворяющих, могла лишиться достаточного влияния без вовремя разработанных мероприятий, направленных на цифровизацию представления объектов классического и современного искусства.

Национальный Проект «Культура», разработанный в соответствии с указом Президента Российской Федерации от 7 мая 2018 года № 204 «О национальных целях и стратегических задачах развития Российской Федерации на период до 2024 года», включает в себя три федеральных проекта: «Культурная среда», «Творческие люди» и «Цифровая культура».

Федеральный проект «Культурная среда» направлен на создание и реконструирование культурно-образовательных и музейных комплексов, включающие в себя концертные залы, театральные, музыкальные, хореографические и другие творческие школы, а также выставочные пространства.

Проект «Творческие люди» направлен на продвижение талантливой молодежи в сфере музыкального искусства, а также обеспечение поддержки добровольческих движений, в том числе в сфере сохранения культурного наследия народов Российской Федерации.

«Цифровая культура» включает в себя задачи по организации виртуальных концертных залов не менее чем в 500 городах Российской Федерации и обеспечению широкого внедрения цифровых технологий в культурное пространство страны.

Выполненная дипломная работа в рамках Национального Проекта «Культура», относящаяся к Федеральному проекту «Цифровая культура», грезит не только увеличить число обращений к цифровым ресурсам.

Целью веб-сервиса является формирование пространства выявления и поддержки талантов Российской Федерации, предоставление всестороннего духовно-нравственного развития посредством доступа к качественному интернет-контенту, а также популяризация творческих инициатив и проектов, реализованных на территории страны.

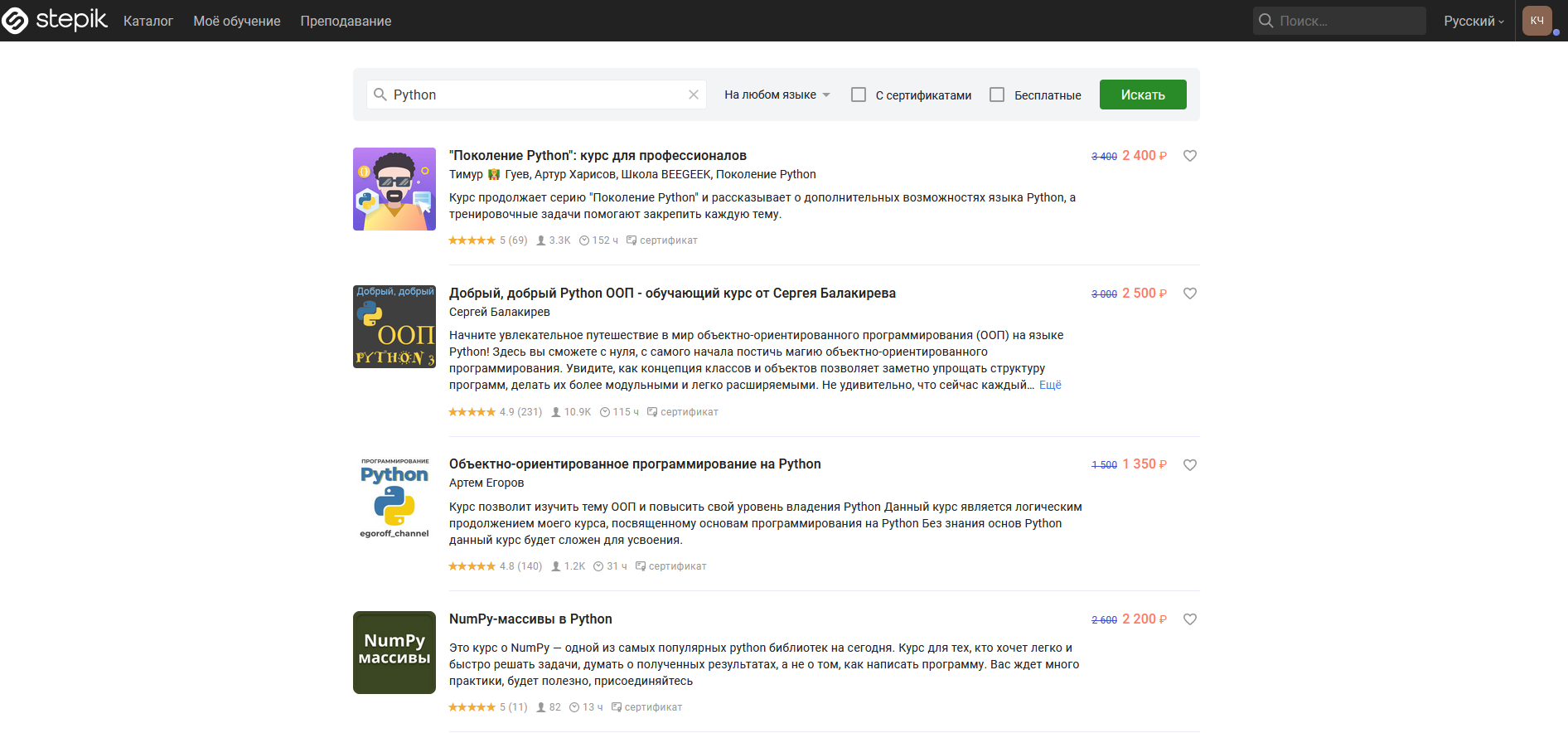
Для осуществление поставленной цели необходимо выполнить следующие задачи:

1. провести информационное исследование выбранной предметной области на выявление конкурентов-аналогов, систем со схожим функционалом;
2. 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;
3. 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;
4. разработать основной функционал программного продукта для организации виртуального пространства просмотра контента;
5. провести тестирование программного продукта;
6. выполнить технико-экономическое обоснование программного продукта;
7. подготовить документационный материал по процессу работы над проектом;
8. обосновать выпускную квалификационную работу.
9. **ПОСТАНОВКА ЗАДАЧИ**
   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.

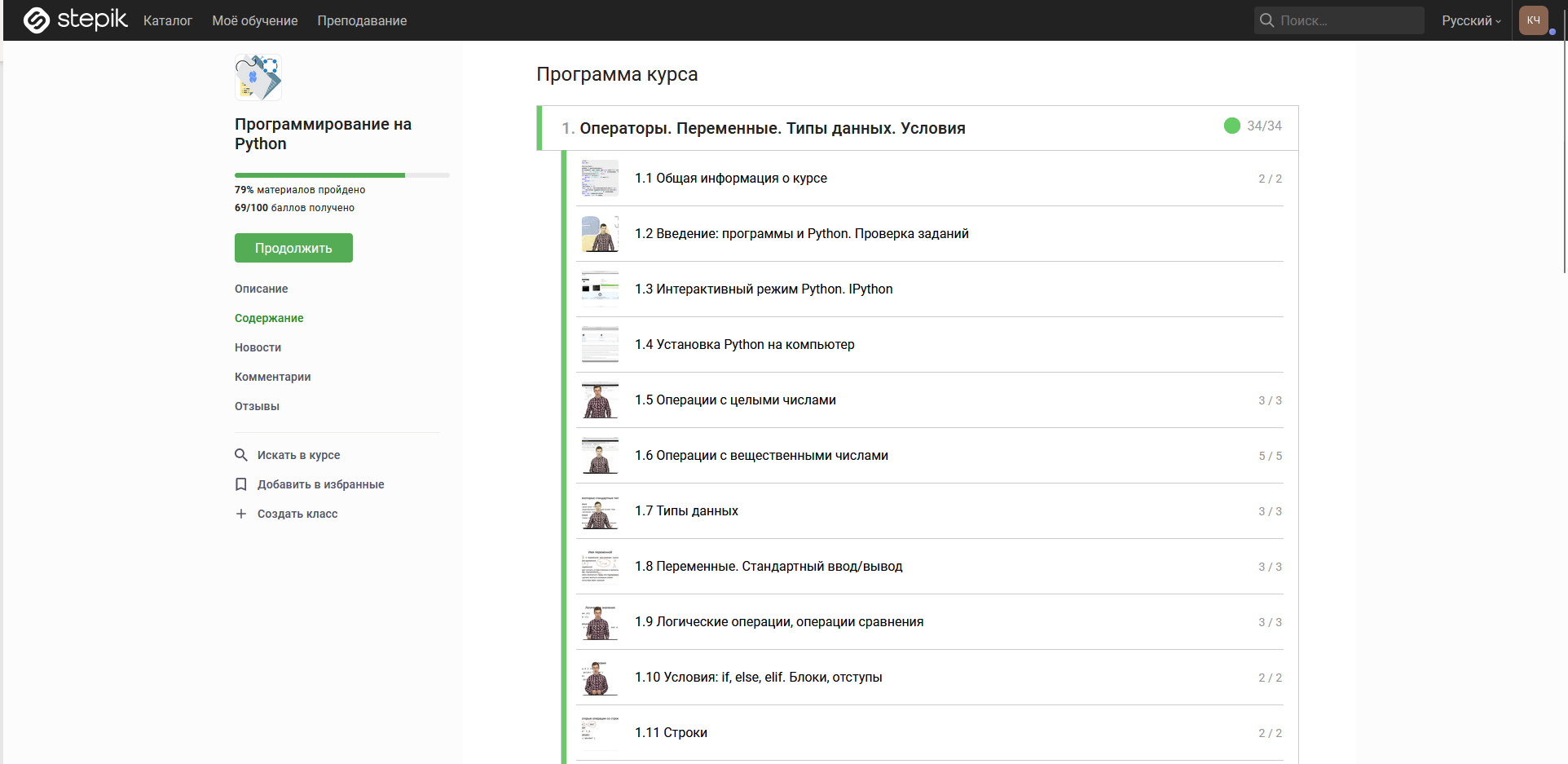
В «Stepik» имеется большое количество курсов связанных с Python



*Рис. 1.1 курсы python на stepik*

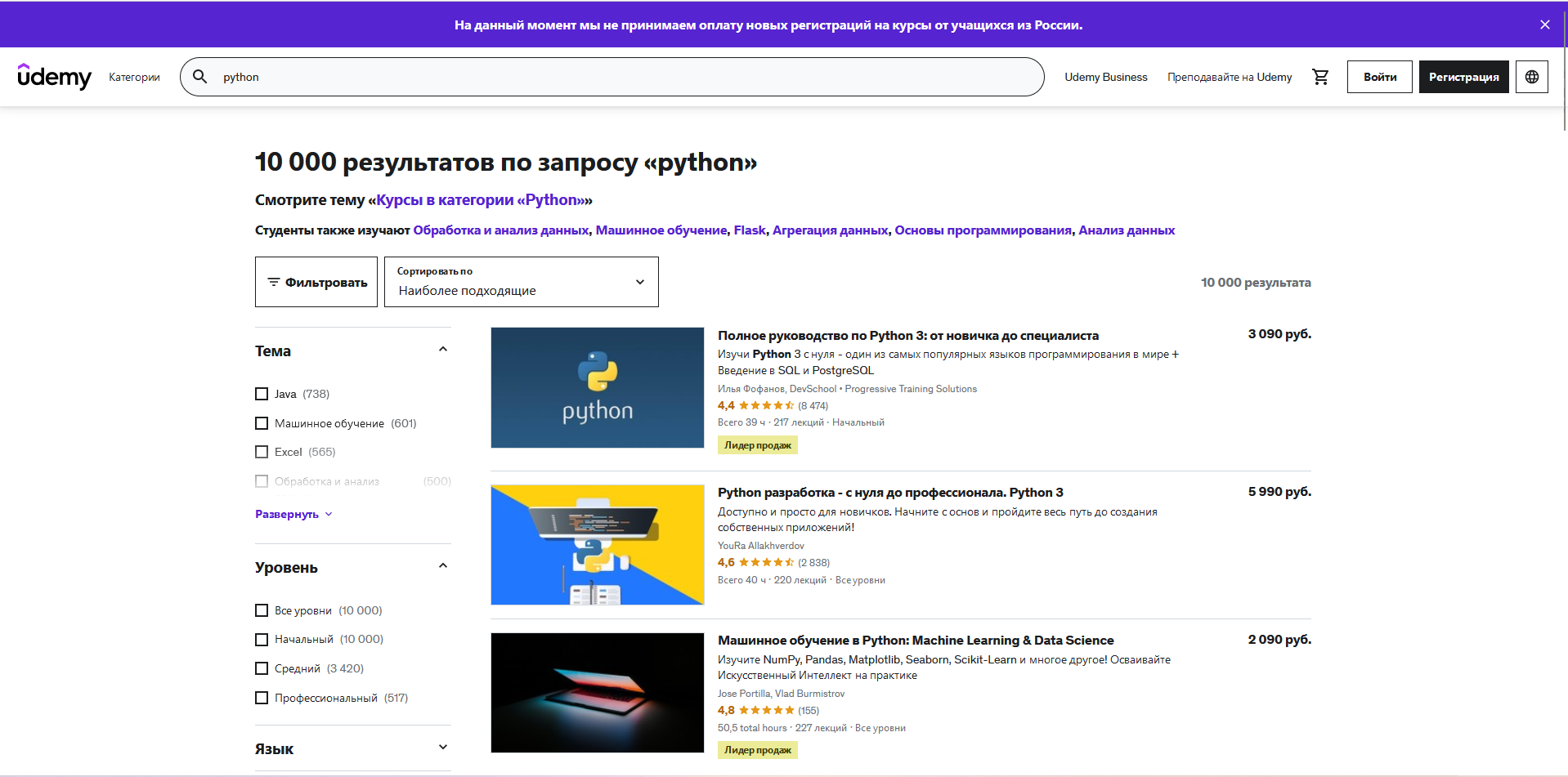
Данный ресурс предоставляет функционал по:

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



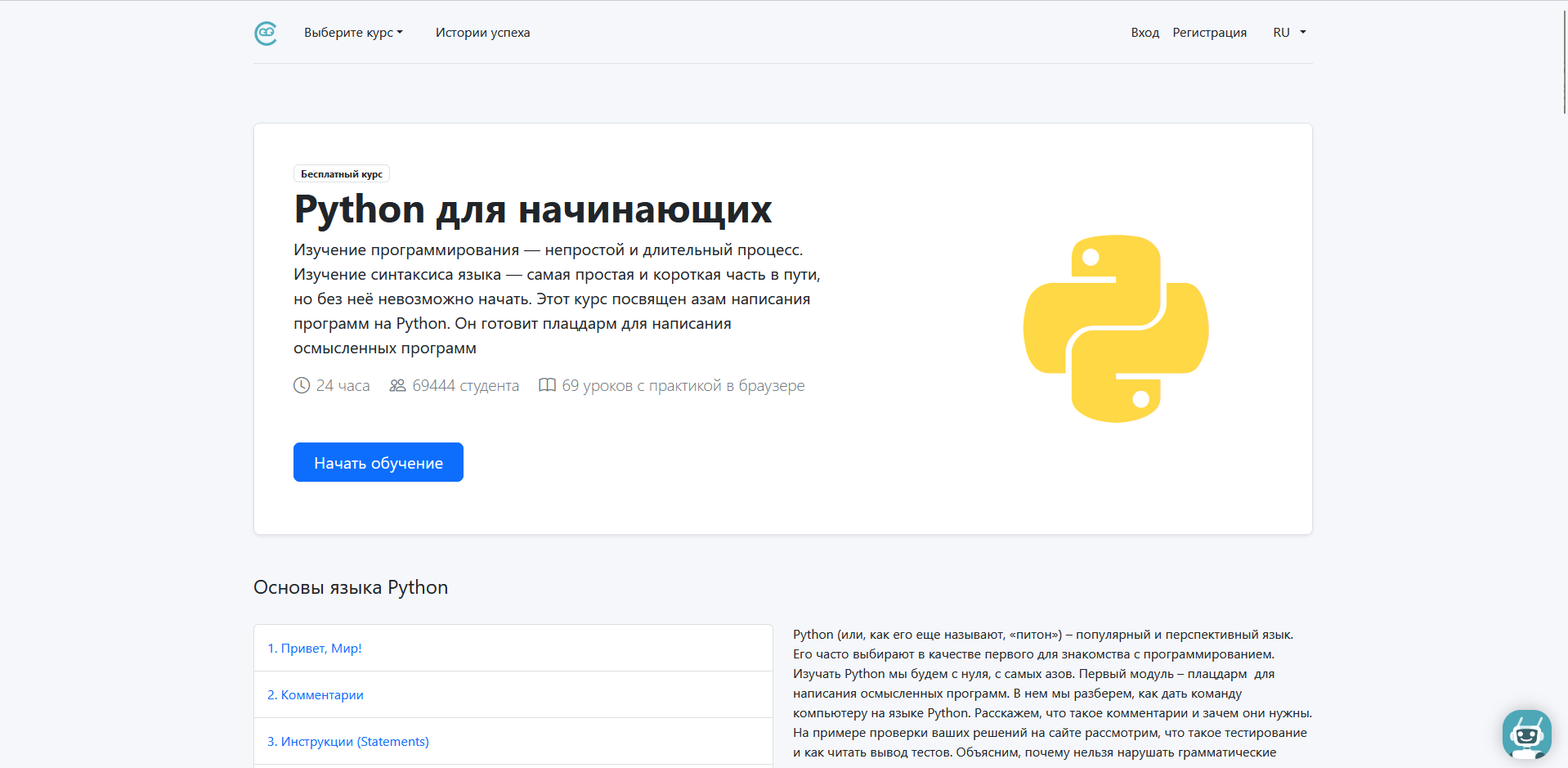
*Рис. 1.2 Просмотр прогресса*

1. **«Udemy.com»** — 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. 1.3 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. **Формирование требований к программному продукту**
      1. **Бизнес-требования**

Спроектированный продукт направлен людей, желающих изучить язык программирования «Python»

Целевая аудитория — пользователи ПК, интересующиеся сферой программирования.  
Веб-сервис дает возможность начать изучение программирования на python с нуля.

Разрабатываемый проект должен содержать в себе достаточное количество информации о языке, а также соответствующих практических задач. К немаловажной части функционала можно отнести возможность посетителя отслеживать процесс обучения, просматривать историю задач, а также изучать или делится решением с другими пользователями.

Веб-приложение должно быть совместимо со всеми популярными видами гаджетов — обладать адаптивностью под экраны телефонов и планшетов, а также мониторов компьютерных устройств.

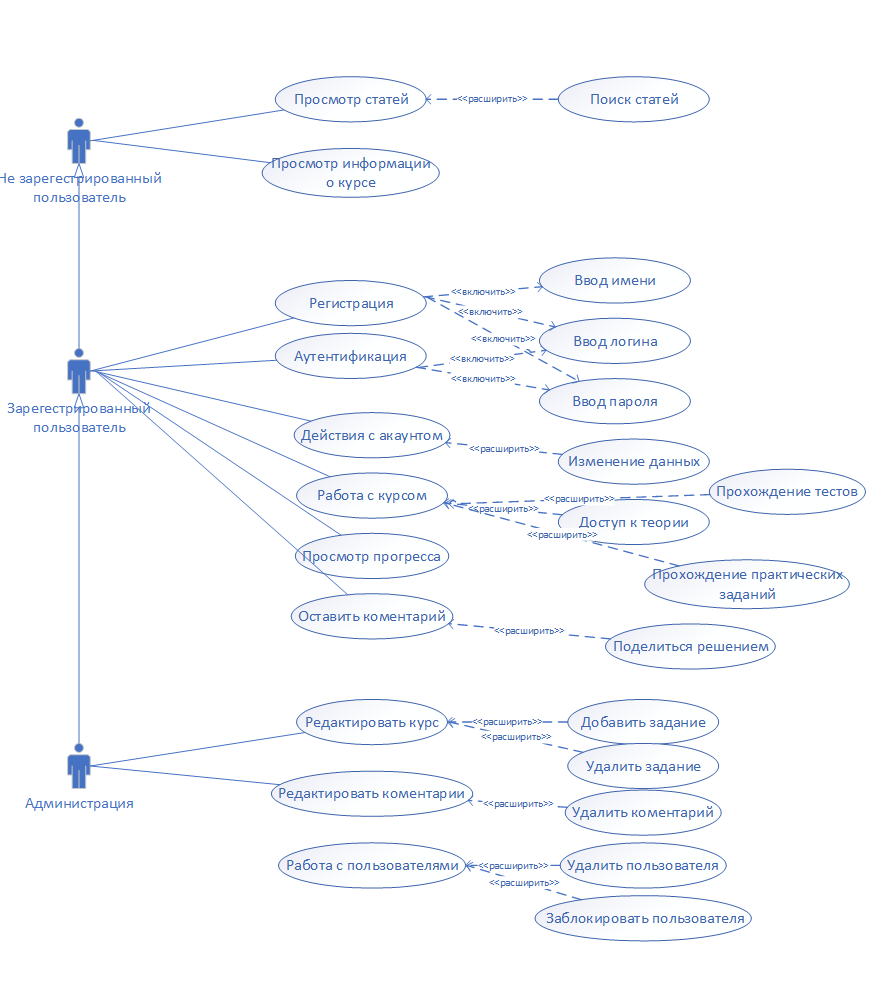
Эффективность программного продукта заключена в достижении рационального соотношения между затратами на создание программы и целевыми эффектами, получаемыми при ее функционировании.

Требования по безопасности хранения персональных данных пользователя портала будут строго соблюдены.

К главным же рискам системы можно отнести: опасность совершения DDOS-атак на портал, приводящих к перегрузке сайта и временном отсутствии доступа к источнику данных, а также недоверие пользователей в виде заботы о сохранении целостности и конфиденциальности персональных данных.

* + 1. **Пользовательские требования**

На рисунке 1.4 представлена диаграмма вариантов использования проектируемого веб-сервиса с ролью пользователя:



*Рис. 1.4 Диаграмма вариантов использования*

К возможностям незарегистрированного пользователя относится следующий функционал:

**Просмотр информации о курсе:**

На главной странице находится общая информация о курсе, а также краткое оглавления глав и уроков:

**Просмотр статей:**

В отдельной вкладке есть возможность просмотра краткого содержания различных статей, связанных с курсом и языком программирования.

**Просмотр статьи:**

При нажатии на статью есть возможность полностью ознакомиться с ней, а также оставить комментарий.

**Поиск статей:**

На сайте у посетителя есть возможность найти определенную статью.

Для этого: предусмотрено поле поиска в «шапке» сайта, куда можно ввести название статьи, для поиска её в базе данных проекта;

В случае, если поисковый результат есть в базе, система выдаст релевантный результат, иначе сообщит пользователю об отсутствии данного.

**Регистрация в веб-приложении:**

Кнопка входа в систему находится либо в элементе header (от англ. заголовок, шапка) веб-сервиса, либо в меню приложения, которое открывается при нажатии на кнопку с иконкой бургер-меню. После клика по кнопке посетитель перенаправляется на страницу с формой для входа, состоящей из двух полей ввода: почты и пароля. Если у пользователя нет аккаунта, необходимо пройти по ссылке в заголовке, перенаправляющей на форму регистрации в системе, состоящую из трех полей: псевдонима, электронной почты и пароля. Данные формы наделены определенными критериями-характеристиками, которые необходимо соблюсти при регистрации.

В случае, если посетитель не авторизован, при попытке взаимодействия с курсом, либо с данными личного кабинета, система будет также перенаправлять на форму входа в приложение.

Возможности зарегистрированного и не аутентифицированного пользователя, помимо всех возможностей незарегистрированного пользователя (за исключением регистрации), включают в себя следующие возможности:

**Аутентификация в web-приложении:**

При нажатии на ссылку «Мой профиль», пользователь будет перенаправлен на страницу аутентификации, для выполнения которой необходимо ввести электронную почту, указанную при регистрации, и пароль. При положительном результате валидации введенных данных аутентификация пройдет успешно.

**Восстановление пароля:**

В случае, если пользователь был зарегистрирован в веб-приложении, но забыл свой пароль, у него есть возможность его восстановить, используя для этого email. На странице авторизации, при нажатии на ссылку «Забыли пароль?», пользователя переносит на форму ввода электронной почты с кнопкой «Подтвердить». После выполнения действия происходит отправка сообщения на указанный адрес пользователя. При переходе по ссылке в письме пользователь сможет ввести новый пароль в соответствии с нормами безопасности.

**Работа с курсом:**

Где находится

При переходе на страницу курса, отображается текущая статистика по курсу (пройденные задания, сохраненные решения, оставленные комментарии, сколько осталось пройти).

При выборе определенного урока, он открывается, каждый урок содержит: теорию, тесты и практические задания для решения.

После успешного решения задания есть возможность оставить комментарий или поделится решением.

Если урок пройден до конца (решены все тесты и выполнены практические задания) урок помечается как пройденный

**Возможности администрации:**

У администрации имеется личный кабинет, где есть возможность редактировать курс, создавать новые уроки, редактировать и удалять уже существующие.

У администрации есть доступ к функционалу связанному с модерацией, создание статей, удаление комментариев, блокировка и удаление пользователей.

* + 1. **Функциональные требования**

Web-сервис должен строиться на основании трех функциональных ролей, определяемых до и после регистрации на сайте:

1. посетитель — не зарегистрированный пользователь системы;
2. пользователь — авторизованный клиент с доступом к дополнительному функционалу сервиса;
3. администратор — сотрудник, наделенный правами управления взаимодействия пользователя с сервисом.

Web-сервис включает в себя следующий функционал:

1. регистрация / авторизация на портале с использованием электронной почты и пароля;
2. управление учетной записью авторизованного клиента — возможность изменить персональные данные
3. осуществление поиска интересующего материала соответствующее поле поиска;
4. взаимодействие с курсом:
   * 1. доступ к теории;
     2. решение тестов;
     3. выполнение практических задач;
     4. комментирование статей и заданий;

5. администрирование курса

1. Редактировать уроки;

2. модерация статей и комментариев;

3. работа с пользователями

* + 1. **Нефункциональные требования**

Для размещения серверной части web-приложения необходим web-сервер с поддержкой технологии asp.net версии >12.18.0, файловый сервер и сервер базы данных postgreSQL.

Данные должны передаваться между серверной и клиентской частью приложения по протоколу HTTPS в формате JSON.

Для работы с клиентской частью web-приложения необходимо современный браузер, с поддержкой HTML5, CSS 3 и ECMAScript 2015 (ES6). Например:

1. Google Chrome;
2. Yandex браузер;
3. Opera;
4. Opera GX;
5. Safari.
   * 1. **Ограничения**

**Ограничения на ввод электронной почты:**

Имя пользователя может в себе содержать:

1. латиницу;
2. цифры;
3. знаки! # $ % & ' \* + — / =? ^ \_ ` { | } ~
4. точку, за исключением первого и последнего знака, которая не может повторяться.

Имя хоста состоит из компонентов, разделенных точкой и суффиксов (домены первого уровня). Длина компонента — не более 63 символов.

Компоненты имени хоста могут содержать:

1. латиницу;
2. цифры;
3. дефисы (кроме начала и конца компонента).

**Ограничения на загрузку файла аватара:**

1. размер файла меньше 5 Mb;
2. формат файла: .jpg/.jpeg/.png

**Ограничения на ввод псевдонима:**

1. длина должна быть не более 10 символов;

**Ограничения на ввод пароля:**

1. пароль должен содержать только латинские буквы и цифры;
2. длина должна быть не менее 6 символов.
   * 1. **Требования к интерфейсам**

Для хранения данных необходимо использовать сервер базы данных, СУБД PostgreSQL и файловый сервер. Информация в базе данных должна выбираться, изменяться, добавляться с помощью Web API.

Web API должен быть реализован с помощью NestJS (фреймворк NodeJS - программная платформа, основанная на движке V8, превращающая JavaScript из узкоспециализированного языка в язык общего назначения) на языке TypeScript.

Front-end должен быть выполнен с помощью ReactJS. Front-end не должен иметь непосредственного доступа к базе данных, вся информация должна поступать через Web API. Проект разрабатывается по технологии SPA, подразумевающий загрузку всех страниц сервиса в единственном HTML-документе. Сама архитектура front-end части заключается в компонентном подходе: элемент создается однажды и переиспользуется в различных частях продукта.

* 1. **Программные средства разработки**

Для разработки Web-приложения использовались следующие программные средства:

1. интерфейс пользователя проектировался в графическом редакторе Figma;
2. для составления диаграмм использовался продукт MS Visio 2022;
3. для написания и тестирования front-end составляющей использовался следующий стек:
   * 1. Visual Studio 2022
     2. JavaScript XML;
     3. FireFox
     4. GitHub.
   1. **Аппаратные средства разработки**

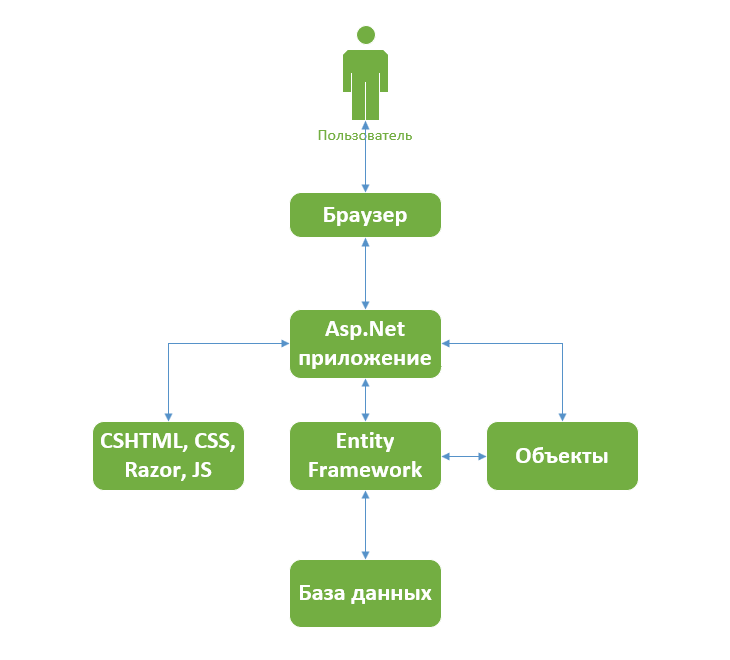
Для разработки веб-приложения использовался ПК со следующими характеристиками:

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**
   1. **Архитектура системы**

Для обеспечения масштабируемости решения и возможности выноса ресурсоемких задач на сторону сервера выбрана трехуровневая архитектура (рис. 2.1), состоящая из:

1. уровня представления (клиент) — представляет из себя интерфейс пользователя, который реализуется с помощью HTML, CSS, библиотеки ReactJS и работает через браузер. На этом уровне располагается простейшая бизнес-логика. Взаимодействует с уровнем приложения через вызовы Web API;
2. уровня приложения (сервер) — здесь с помощью NestJS, фреймворка для Node.js, реализуется основная бизнес-логика приложения;
3. уровня данных — состоит из базы данных с информацией, сервера баз данных и файлового сервера.

Преимущество трехуровневой архитектуры с реализацией Web API — уровень представления не зависит от используемых на уровне приложения технологий.



*Рис. 2.1 Архитектурная модель системы*

* + 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**

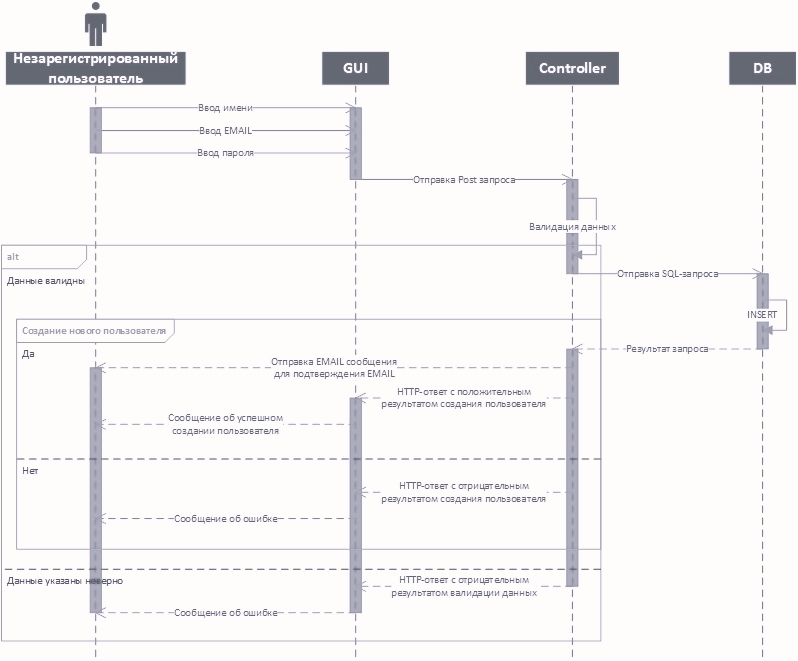
Уровень данных представлен сервером баз данных PostgreSQL и файловым сервером Object Storage от Yandex.Cloud.

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.

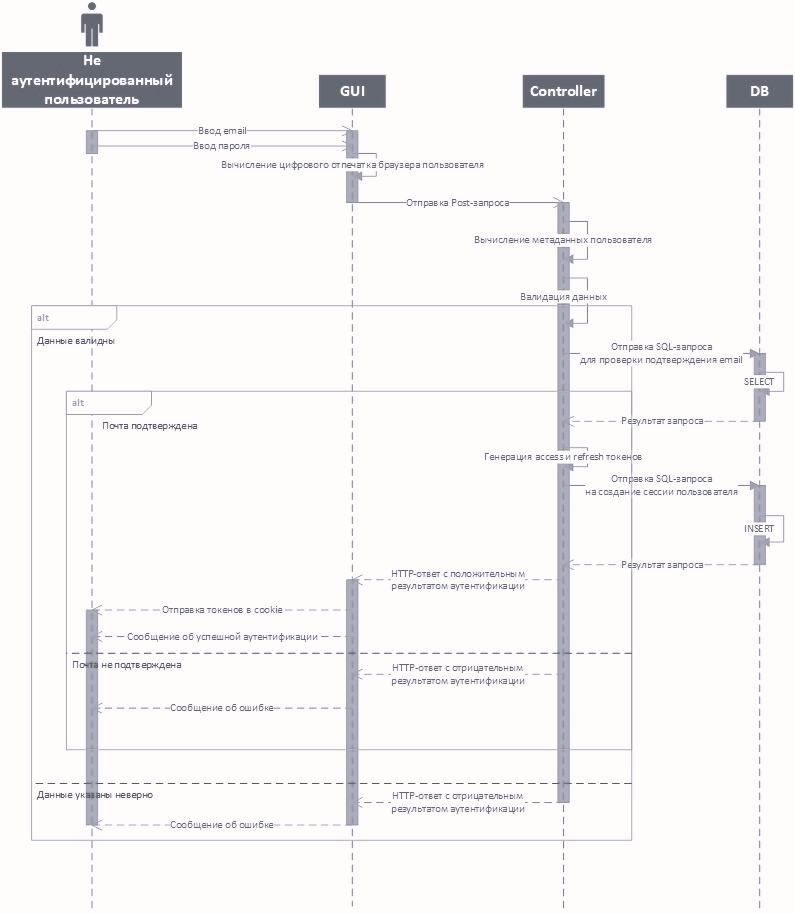
* 1. **Моделирование основных сценариев системы idef, uml**
     1. **Диаграммы последовательности**

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).



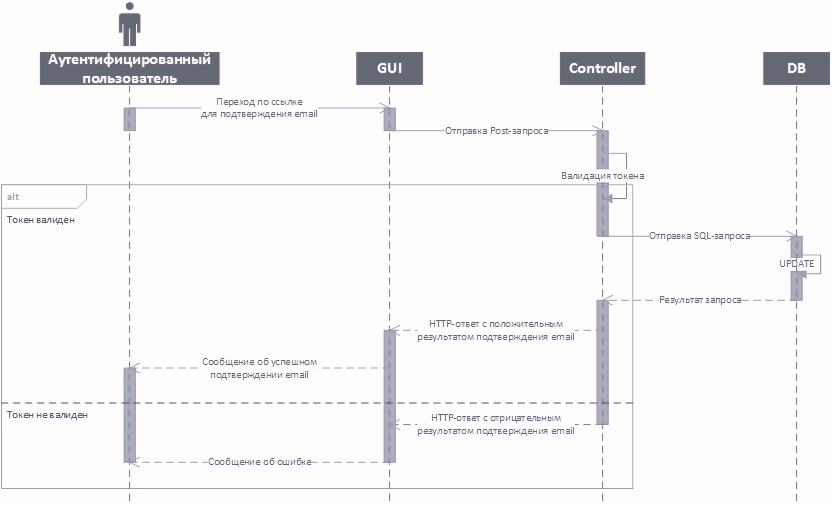
*Рис. 2.2 Диаграмма последовательности прецедента «Регистрация»*

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).



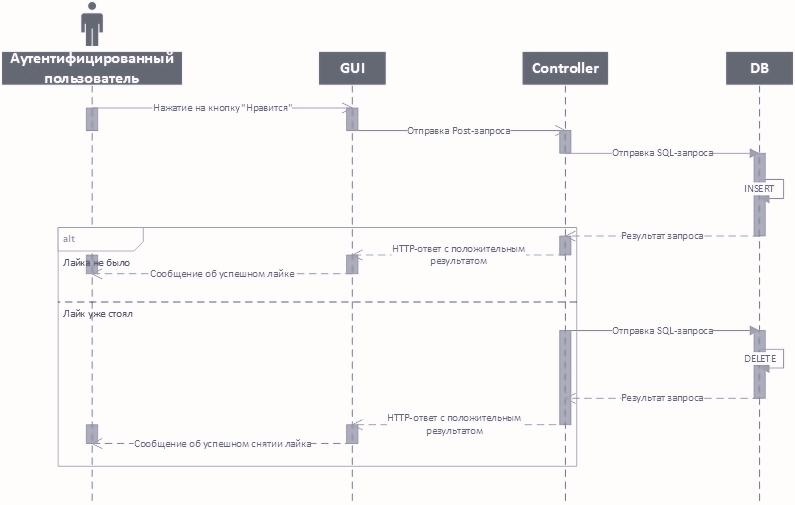
*Рис. 2.3 Диаграмма последовательности прецедента «Аутентификация»*

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).



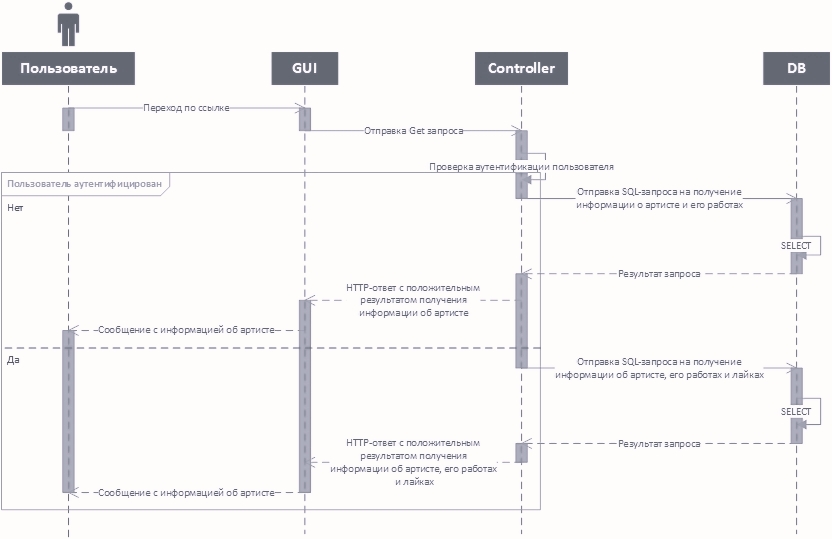
*Рис. 2.4 Диаграмма последовательности прецедента «Подтверждение email»*

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).



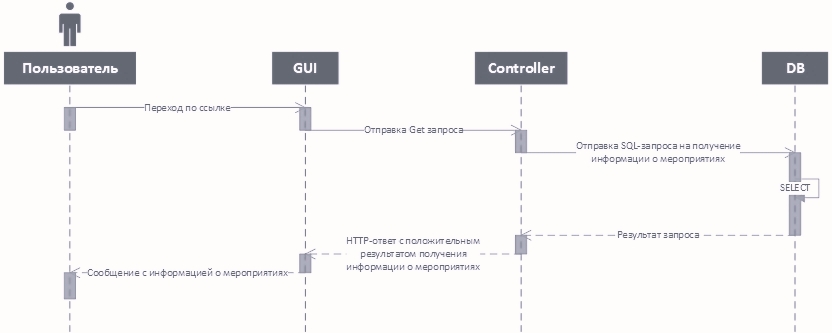
*Рис. 2.5 Диаграмма последовательности прецедента функции «Нравится»*

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).

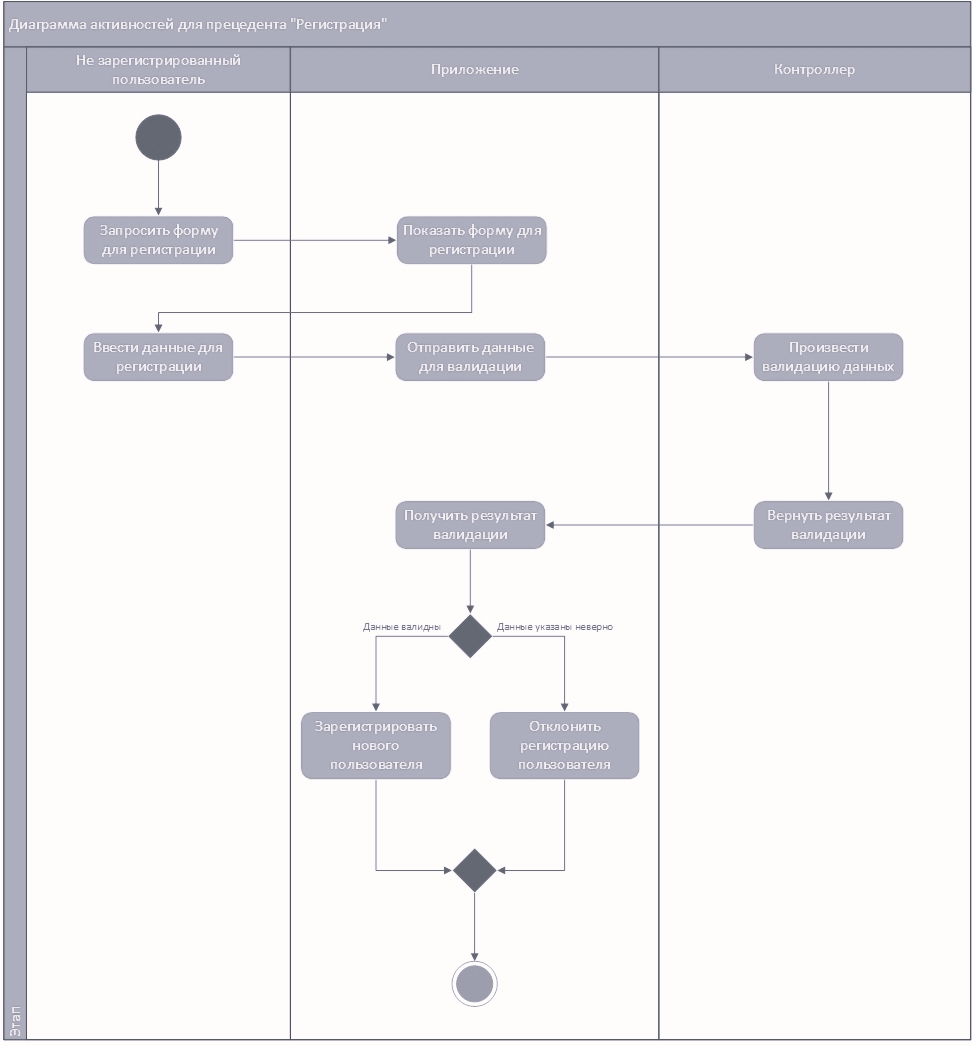


*Рис. 2.7 Диаграмма последовательности прецедента «Получение информации о мероприятиях»*

Диаграммы последовательности других прецедентов строятся по той же схеме:

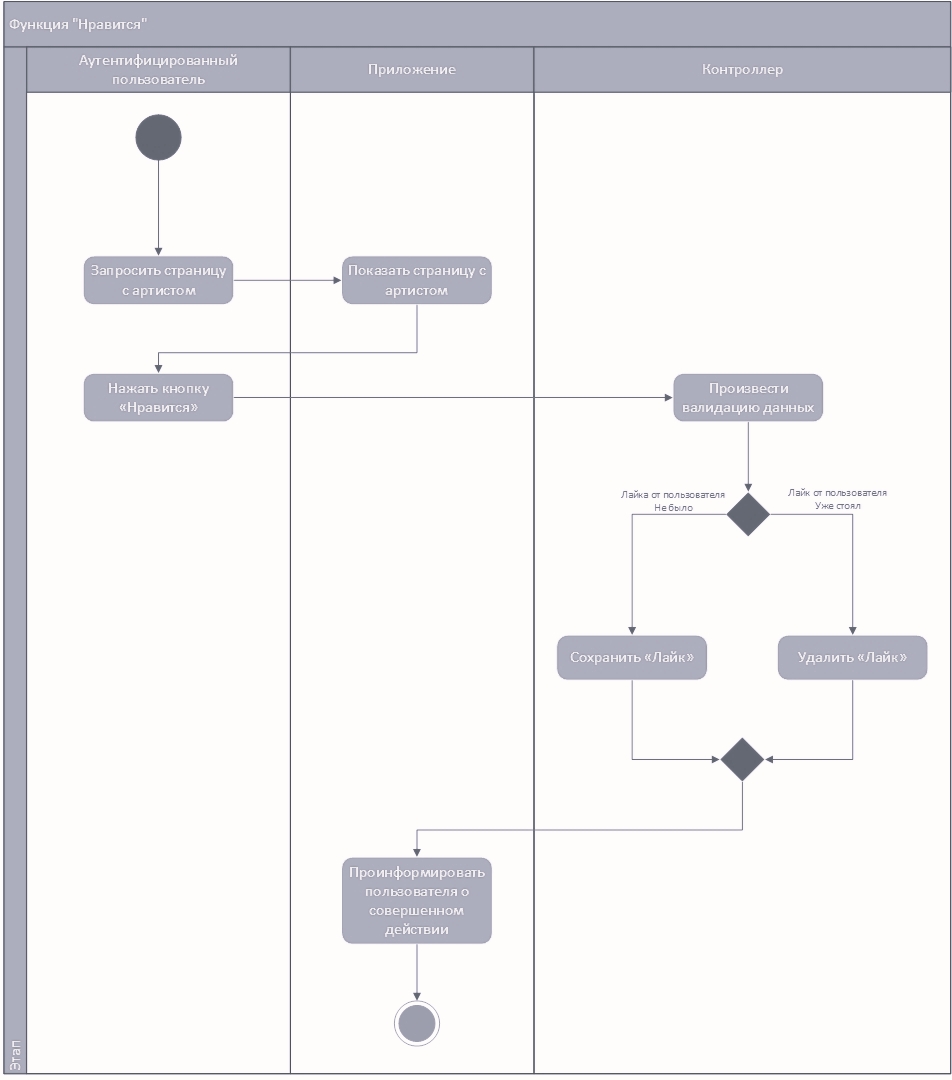
1. пользователь взаимодействует с GUI;
2. GUI отправляет HTTP-запросы на сервер;
3. на уровне приложения (controller) происходит обработка информации, обращения к БД и файловому серверу;
4. пользователю отправляется ответ или сообщение об ошибке.
   * 1. **Диаграммы активностей**

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»*

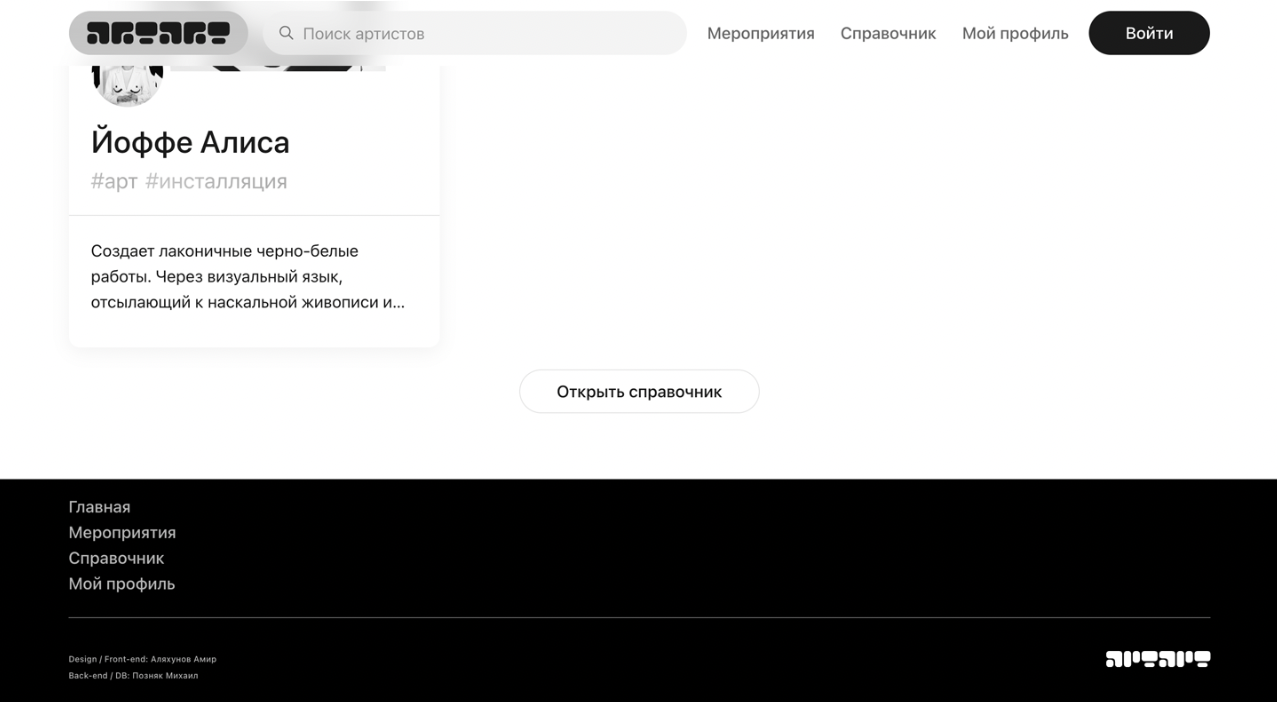
Диаграммы активностей других прецедентов строятся по той же схеме:

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

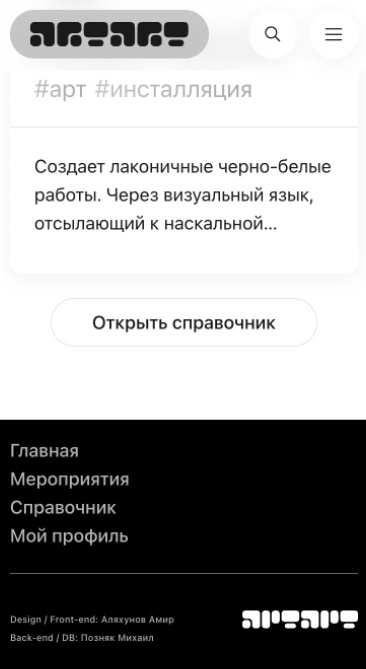
Интерфейс веб-сервиса должен быть адаптивным, корректно отображаться и функционировать на устройствах различного типа, с различным разрешением и форм-фактором экрана: монитор, планшет, смартфон. Визуальная часть программного продукта разрабатывалась в графическом редакторе Figma.

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*



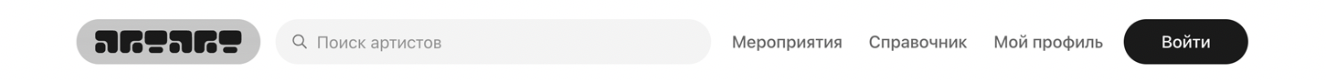
*Рис. 2.11 Общее представление веб-сервиса — Мобильная версия*

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.