Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

**Иркутский национальный исследовательский**

**технический университет**

|  |
| --- |
| Институт информационных технологий и анализа данных |
| наименование института |

Допускаю к защите

|  |  |
| --- | --- |
| Руководитель |  |
|  | подпись |
|  | З.А. Бахвалова |
|  | И.О. Фамилия |

|  |
| --- |
|  |
| Разработка прикладного программного обеспечения |
| наименование темы |

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

|  |
| --- |
| Технологии разработки программных комплексов |

|  |
| --- |
| 1.008.00.00 ПЗ |
| обозначение документа |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил студент |  | ИСТб-19-2 |  |  |  | Ю.В. Комогорцева |
|  |  | шифр группы |  | подпись |  | И.О. Фамилия |
| Нормоконтроль |  |  |  |  |  | З.А. Бахвалова |
|  |  |  |  | подпись |  | И.О. Фамилия |
| Курсовая работа защищена с оценкой | | | |  | | |

Иркутск 2022 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

ЗАДАНИЕ

на курсовую работу

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| По курсу | Технологии разработки программных комплексов | | | | | | | | | | | | | | | | | |
| Студенту | Комогорцевой Ю.В. | | | | | | | | | | | | | | | | | |
|  | (фамилия, инициалы) | | | | | | | | | | | | | | | | | |
| Тема работы | | Разработка прикладного программного обеспечения | | | | | | | | | | | | | | | | |
| Исходные данные | | | Разработать систему, которая формирует сообщество любителей литературы. | | | | | | | | | | | | | | | |
| Рекомендуемая литература | | | | | | | |  | | | | | | | | | | |
| 1. Курсовая работа «Разработка прикладного программного обеспечения» Руководство и методические указания для студентов направления: 09.03.02 «Информационные системы и технологии». Составитель Бахвалова З.А. – Иркутск, Изд-во ИРНИТУ, 2018, 61 с. | | | | | | | | | | | | | | | | | | |
| Графическая часть на | | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ листах. | | | | | | | | | | | | | | |
| Дата выдачи задания | | | | « | 18 | » | февраля | | | | | 2022 г. | | | | |
|  | | | | | | | | | | | | | | | | | | |
| Задание получил | | | | | | | | |  | | | |  | | Ю.В. Комогорцева | | | |
|  | | | | | | | | | подпись | | | |  | | И.О. Фамилия | | | |
|  | | | | | | | | | |  |  | | |  | |  | |  |
| Дата представления работы руководителю | | | | | | | | | | « |  | | | » | |  | | 2022 г. |
|  | | | | | | | | | | | | | | | | | | |
| Руководитель курсовой работы | | | | | | | | |  | | | |  | | З.А. Бахвалова | | | |
|  | | | | | | | | | подпись | | | |  | | И.О. Фамилия | | | |

**Содержание**

[Введение 5](#_Toc105367549)

[1 Постановка задачи 6](#_Toc105367550)

[1.1 Текстовое описание предметной области 6](#_Toc105367551)

[1.2 Цель 6](#_Toc105367552)

[1.3 Словари 6](#_Toc105367553)

[1.4 Постановка задачи 8](#_Toc105367554)

[1.5 Функциональность ПО 9](#_Toc105367555)

[2 Анализ 13](#_Toc105367556)

[2.1 Модель предметной области (IDEF0) 13](#_Toc105367557)

[2.2 Диаграмма вариантов использования UML Use Case 21](#_Toc105367558)

[2.3 Концептуальная модель хранилища данных 22](#_Toc105367559)

[2.4 DFD - диаграмма потоков данных 23](#_Toc105367560)

[2.5 Описание вариантов использования 27](#_Toc105367561)

[2.5.1 Спецификация варианта использования «Регистрация». 27](#_Toc105367562)

[2.5.2 Спецификация варианта использования «Войти в личный кабинет». 28](#_Toc105367563)

[2.5.3 Спецификация варианта использования «Выйти из личного кабинета». 30](#_Toc105367564)

[2.6 Описание интерфейса 31](#_Toc105367565)

[3 Проектирование программного продукта 36](#_Toc105367566)

[3.1 Выбор и обоснование программных инструментов 36](#_Toc105367567)

[3.2 Проектирование хранилища данных 38](#_Toc105367568)

[3.3 Описание процесса авторизации 39](#_Toc105367569)

[3.4 Проектирование клиентской части разрабатываемой системы 39](#_Toc105367570)

[3.5 Проектирование серверной части разрабатываемой системы 43](#_Toc105367571)

[3.6 Реализация ВИ с учетом спроектированных классов данных и интерфейсных классов 46](#_Toc105367572)

[3.7 Фактическая организация модулей системы 47](#_Toc105367573)

[3.8 Модель развертывания продукта 48](#_Toc105367574)

[3.9 Макеты пользовательских интерфейсов 49](#_Toc105367575)

[4 Программная реализация 53](#_Toc105367576)

[5 Тестирование 60](#_Toc105367577)

[5.1 Модульное тестирование 60](#_Toc105367578)

[5.2 Интерфейсное тестирование 61](#_Toc105367579)

[5.3 Функциональное тестирование 62](#_Toc105367580)

[Заключение 64](#_Toc105367581)

[Список литературы 65](#_Toc105367582)

# Введение

Целью курсовой работы является закрепление теоретических знаний, полученных в процессе изучения курса «Технологии разработки программных комплексов» и развитие практических навыков постановки задачи, проведения анализа, проектирования и реализации в ходе решения конкретной практической задачи – разработки программного продукта на основе спиральной модели создания программного обеспечения. А также при выполнении курсовой работы должны развиться навыки самостоятельного выбора и обоснования решений возникающих проблем, базового опыта разработки программного обеспечения, работы с технической и справочной литературой.

В рамках курсовой работы решается задача создания информационной системы, которая должна сформировать сообщество любителей литературы.

Предполагается, что этапы постановки задачи и анализа станут фундаментом выпускной квалификационной работы, поэтому на данной итерации разработки будет реализована лишь часть всей системы.

# 1 Постановка задачи

## 1.1 Текстовое описание предметной области

На сегодняшний день любители литературы сталкиваются с проблемой выбора книги для прочтения среди миллиардов произведений по всему миру. Человек может зря потратить время на прочтение и деньги на покупку книги, которая ему не понравилась. Поэтому важно узнать некоторую информацию о книге до ее покупки. Так же, просто придя в магазин или библиотеку, трудно будет сделать выбор, опираясь на любимых авторов и выбранный жанр. Тем самым формируется потребность в системе, которая будет служить источником структурированной информации о книгах.

При выборе книги человек так же может прислушиваться к мнению других людей, например, ознакомиться с рецензиями или подборками книг. Но мнение других людей не найти в библиотеке или книжном магазине. В условиях современной реальности человек может узнать мнение других людей о книгах в интернете. Но большинство сайтов в интернете не контролируют отзывы людей о книгах, поэтому качество этой информации оставляет желать лучшего. Лучшим решением этой проблемы будет являться площадка обмена мнениями с контролем людей, разбирающихся в литературе.

Существуют ситуации, когда человеку необходимо сохранить информацию о книге (название, автор и т.д.) для дальнейшего прочтения или рекомендации. Существуют разные способы сохранить информацию о книге: снимок экрана, использование памяти человека, добавление в закладки браузера, запись в электронном и бумажном виде и т.д. Но все эти способы не являются удобными и надежными, так как информация не собрана в одном месте и не структурирована. Поэтому для любителя литературы не менее важно иметь место для хранения информации о важных для него книгах.

## 1.2 Цель

Система должна сформировать сообщество любителей литературы.

## 1.3 Словари

Таблица 1 – Словарь предметной области

|  |  |  |
| --- | --- | --- |
| № | Сущность | Описание |
| 1 | Пользователь | Роль пользователя, может быть авторизованный или неавторизованный. Простой пользователь, которому доступны не все функции. |
| 2 | Контент-менеджер | Роль пользователя, может быть авторизованный или неавторизованный. Пользователь, который имеет доступ к функциям обычного пользователя, а также к работе с заявками. |
| 3 | Модератор | Роль пользователя, может быть авторизованный или неавторизованный. Пользователь, который имеет доступ к функциям обычного пользователя, а также к работе с подборками и рецензиями. |

Продолжение таблицы 1 – Словарь предметной области

|  |  |  |
| --- | --- | --- |
| 4 | Учетная запись | Совокупность данных о пользователе, необходимой для его опознавания (аутентификации) и предоставления доступа к его личным данным |
| 5 | Личный кабинет | Это часть системы, которая служит для отображения личных данных пользователя. Доступен только авторизированному пользователю |
| 6 | Личные данные пользователя | Совокупность информации о деятельности пользователя в системе, которая состоит из читательского дневника, рецензий и оценок пользователя, заявок на добавление книги и т.д. |
| 7 | Административная панель | Часть системы, с помощью которой можно редактировать контент сайта. Доступна авторизованному модератору и контент-менеджеру. |
| 8 | Заявка | Совокупность информации, которую пользователь хотел бы добавить в систему. Может быть с информацией о книге, жанре, авторе. Может иметь статус: ожидает проверки, отклонена или принята. |
| 9 | Мнение о книге | Совокупность информации, которая состоит из рейтинга книги, рецензий и подборок. |
| 10 | Рейтинг книги | Составляется из оценок пользователей для книги. |
| 11 | Подборка книг | Список книг, подобранный на определённую тематику, который составляет обычный авторизованный пользователь, а проверяет и допускает к публикации модератор. |
| 12 | Читательский дневник | Совокупность списков книг пользователя, по умолчанию имеется список «читаю», «Хочу читать», «прочитал». Так же подразумевает добавление своих списков. |

Таблица 2 – Словарь по Абботу

|  |  |  |  |
| --- | --- | --- | --- |
| № | Сущность | Глагол | Пояснение глагола |
| 1 | Пользователь | Создавать | Относиться к регистрации пользователя |
| 2 | Контент-менеджер | - |  |
| 3 | Модератор | - |  |
| 4 | Учетная запись | Создавать |  |
| 5 | Личный кабинет | Просматривать | Демонстрируется по запросу пользователя |
| 6 | Личные данные пользователя | Просматривать | Демонстрируются в личном кабинете |
| 7 | Административная панель | Просматривать | Демонстрируется по запросу пользователя |

Продолжение таблицы 2 – Словарь по Абботу

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | Заявка | Отправлять, редактировать, удалять, проверять |  |
| 9 | Мнение о книге | Формировать, просматривать |  |
| 10 | Рейтинг книги | Ставить, просматривать |  |
| 11 | Подборка книг | Формировать, просматривать, проверять |  |
| 12 | Читательский дневник | Формировать, просматривать |  |

Таблица 3 – Объектно-ориентированный словарь

|  |  |  |  |
| --- | --- | --- | --- |
| № | Сущность | Данные | Методы |
| 1 | Пользователь | Email, никнейм\логин, пароль, ФИО, роль | Создавать |
| 2 | Контент-менеджер | - |  |
| 3 | Модератор | - |  |
| 4 | Учетная запись | Email, пароль, роль пользователя | Создавать |
| 5 | Личный кабинет | - |  |
| 6 | Личные данные пользователя | Email, никнейм\логин, пароль, ФИО, роль | Просматривать |
| 7 | Административная панель | - |  |
| 8 | Заявка | Дата и время отправки, информация из заявки, статус | Отправлять, редактировать, удалять, проверять |
| 9 | Мнение о книге | - |  |
| 10 | Рейтинг книги | Число | Ставить, просматривать |
| 11 | Подборка книг | Название, описание, список книг | Формировать, просматривать, проверять |
| 12 | Читательский дневник | Списки читательского дневника | Формировать, просматривать |

## 1.4 Постановка задачи

Что бы система могла быть источником информации о книгах пользователю необходимо иметь возможность просматривать информацию о книге, просматривать информацию об авторе и его книгах, просматривать информацию о жанрах, находить книгу и автора через поисковую строку. Кроме просмотра информации, необходимо иметь возможность информацию добавлять, редактировать и удалять. Подразумевается, что пользователь может только отправить заявку на добавление книги, автора или жанра. А проверять эти заявки, редактировать контент и наполнять систему информацией, связанной с книгами, авторами и жанрами будет контент-менеджер.

Пользователи должны иметь возможность обмениваться мнением о книге с помощью рейтинга книги, рецензий и подборок книг. Так же пользователь может использовать мастер рекомендаций, который находит наиболее подходящие книги для конкретного пользователя на основе всей информации в системе и заданных параметров. Неавторизированный пользователь может только просматривать рейтинг книги, рецензии к ней и подборки книг. Авторизированный пользователь может поставить книге оценку из которой формируется рейтинг книги. Авторизированный пользователь может так же составить подборку книг и написать рецензию. Однако для улучшения качества информации, подборка книг или рецензия должны пройти проверку. Проверку осуществляют модераторы и контент-менеджеры.

Авторизированный пользователь должен иметь возможность вести читательский дневник, с помощью которого он сможет отслеживать процесс чтения и сохранять важные для него книги в собственные списки. Стандартно пользователь в личном кабинете имеет 3 списка: «Хочу читать», «Читаю», «Прочитал». Одна книга не может быть одновременно сразу в этих трех списках. Так же пользователь может создать свой список, например для сохранения книг одной тематики, автора или жанра.

Исходя из поставленных задач можно выделить следующие роли пользователей: пользователь, контент-менеджер и модератор. Каждый из них может быть как авторизованным, так и не авторизованным. Исходя из этого становится очевидным что пользователь должен иметь возможность пройти авторизацию для доступа к информации в личном кабинете и для появления возможности делиться мнением. Контент-менеджер и модератор после авторизации могут переключаться между системой и административной панелью.

## 1.5 Функциональность ПО

Система должна предоставлять незарегистрированному пользователю возможность зарегистрироваться в системе. Для этого пользователю необходимо указать: электронную почту, псевдоним, ФИО и пароль. На одну электронную почту может быть зарегистрирован только один пользователь. В это время в системе создается учетная запись этого пользователя.

Система должна позволять пользователю пройти процедуру авторизации, для предоставления пользователю прав на выполнение различных функций. Для авторизации необходимо быть зарегистрированным в системе, и заполнить информацию: электронная почта пользователя и пароль. Процесс авторизации для всех видов пользователей является одинаковым. Для обычного авторизированного пользователя открыт доступ в личный кабинет, привязанный к этому пользователю. Однако после авторизации для модератора и контент-менеджера будет отображаться система с возможностью переключения на административную панель. То есть контент-менеджер и модератор совмещают в себе роли авторизированного пользователя и свои собственные. Система должна предоставлять возможность авторизованному контент-менеджеру и модератору просматривать административную панель, где предоставляется возможность выполнять те действия, которые недоступны обычному авторизованному пользователю.

Система должна предоставлять пользователю возможность выходить из аккаунта, которая может понадобиться пользователю если он зашел не со своего устройства.

Система должна предоставлять пользователю возможность ознакомиться с документацией пользователя.

Функционал для объектов системы **книги, жанры, авторы** следует объединить, потому что они очень схожи и их отличие заключается лишь в информации, которая с ними связана. На само выполнение функций это не влияет.

Система должна предоставлять пользователю возможность просматривать список книг, который может быть получен при поиске, при просмотре жанра. Так же необходимо что бы система предоставляла возможность пользователю сортировать и фильтровать список книг.

Система должна позволять пользователю искать книги по всем параметрам. В результате поиска система либо выдает список найденных книг, либо сообщает об отсутствии книг по заданному поиску. Система должна позволять пользователю искать автора по ФИО. В результате поиска система либо выдает список найденных авторов, либо сообщает об отсутствии авторов по заданному поиску.

При просмотре списка книг, система должна предоставлять возможность пользователю выбрать книгу и посмотреть информацию о ней. Система должна при выполнении функции просмотра книги показывать рейтинг книги и рецензии, написанные к этой книге. Система должна предоставлять пользователю возможность посмотреть информацию об авторе: ФИО, биография, дата рождения и смерти, фото автора и книги которые он написал. Система должна предоставлять пользователю возможность посмотреть информацию о жанре: описание жанра и список книг этого жанра.

Система должна предоставить авторизованному пользователю возможность отправить заявку на добавление книги/автора/жанра, если они не найдены. В заявке авторизованному пользователю должен указать по возможности все свойства книги, жанра или автора, которые он знает. Так же система должна предоставить авторизованному пользователю возможность посмотреть отправленную заявку, редактировать и удалить. Заявка на добавление может иметь статусы: ожидает проверки, отклонена или принята. Редактировать и удалить отправленную заявку возможно только до момента проверки этой заявки.

Система должна позволять авторизованному контент-менеджеру проверить заявку на добавление книги, показав ему всю информацию из заявки. Результатом проверки пользователь изменяет статус заявки с ожидает проверки на отклонена или принята. В случае принятия заявки пользователь должен выполнить функцию добавление книги/автора/жанра, если заявка отклонена, то пользователь должен указать почему. Система должна позволять авторизованному контент-менеджеру добавить книгу, если он укажет название, жанры, автор, описание, серия, издательство, картинка обложки, номер ISBN, год издания, язык. Система должна позволять авторизованному контент-менеджеру добавить автора, если этот пользователь укажет ФИО, биография, дата рождения и смерти, фото автора и книги которые он написал. Система должна позволять авторизованному контент-менеджеру добавить жанр, если он укажет название, список поджанров и описание жанра.

Система должна позволять пользователю редактировать информацию о книге/авторе/жанре и удалять их.

Для того что бы система могла быть площадкой обмена мнениями с контролем людей, разбирающихся в литературе, необходимо сделать возможность **оценивать книгу, писать к ней рецензии и составлять подборки книг**.

Система должна показывать пользователю при просмотре все подборки книг в виде списка. У каждой подборки должно отображаться название и краткое описание. Помимо просмотра списка подборок, необходимо просматривать и одну выдранную подборку. Система должна показывать пользователю информацию о подборке: название, краткое описание, книги входящие в подборку.

Система должна позволять авторизованному пользователю создать и отправить на проверку подборку книг, если он укажет название подборки, описание подборки и книги которые он бы хотел туда включить. После отправления авторизованный пользователь должен иметь возможность посмотреть отправленную на проверку подборку. Система должна показывать авторизованному пользователю в личном кабинете отправленные заявки. Заявка на добавление может иметь статусы: ожидает проверки, отклонена или принята. Кроме просмотра необходимо добавить возможность редактировать и удалять подборку, до момента ее проверки.

Система должна позволять авторизованному модератору проверить заявку на добавление подборки, показав ему всю информацию из заявки. Результатом проверки модератор изменяет статус заявку с ожидает проверки на отклонена или принята. В случае принятия заявки, модератор должен выполнить функцию добавление подборки, если заявка отклонена, то пользователь должен указать почему.

При выставлении оценки книге авторизованный пользователь может выразить свое мнение о книге, выбрав значение от 1 до 10. Если пользователь не добавлял эту книгу в свой читательский дневник, то система уточнит у пользователя, уверен ли он что готов оценить книгу, не прочитав ее. Рейтинг книги формируется из ее оценок. Система должна предоставлять пользователю возможность посмотреть списки книг с наибольшим рейтингом по каждому жанру.

При просмотре книги авторизованный пользователь может написать и отправить на проверку рецензию на эту книгу. Для этого ему нужно написать саму рецензию, заголовок и выбрать тип рецензии (положительная, отрицательная и нейтральная) и указать имеет ли она спойлеры. Система должна показывать авторизованному пользователю в личном кабинете отправленные рецензии, каждую из которых можно так же просмотреть. Рецензия может иметь статусы: ожидает проверки, отклонена или опубликована. Система должна предоставлять авторизованному пользователю возможность редактировать и удалять рецензию, до момента проверки этой рецензии.

Система должна позволять авторизованному модератору проверить рецензию, показав ему всю информацию из неё. Результатом проверки модератор изменяет статус рецензии с ожидает проверки на отклонена или опубликована. В случае положительного результата модератор должен опубликовать рецензию, если рецензия отклонена, то модератор должен указать почему.

Для отслеживания процесса чтения и сохранения важных для него книг необходимо предоставить авторизованному пользователю возможность формировать читательский дневник. Система должна показывать пользователю в личном кабинете читательский дневник. Он состоит из списков книг, которые пользователь может добавлять, удалять или редактировать. По умолчанию 3 списка являются постоянными и всегда будут в читательском дневнике пользователя: «Читаю», «Хочу читать», «Прочитал». Список «Читаю» подразумевает хранение книг в нем, которые пользователь читает в данный момент. Список «Хочу читать» подразумевает хранение книг в нем, которые пользователь хотел бы прочитать в будущем. Список «Прочитал» подразумевает хранение книг в нем, которые пользователь уже прочитал. Система должна позволять авторизованному пользователю добавить книгу в один из списков читательского дневника. Эта функция может быть выполнена при просмотре книги. Система должна позволять авторизованному пользователю перемещать книгу из одного списка в другой. Например, книга была в списке «Хочу читать», и затем пользователь начал читать эту книгу, и он может переместить книгу в список «Читаю». Система должна позволять авторизованному пользователю удалить книгу из выбранного списка. Это можно сделать при просмотре списка в личном кабинете. Система должна позволять авторизованному пользователю смотреть выбранный список книг из читательского дневника. Каждый из списков в читательском дневнике при просмотре можно сортировать.

Система должна позволять авторизованному пользователю добавлять новый список в читательский дневник. Для этого нужно лишь указать его название. Система должна позволять авторизованному пользователю удалить или редактировать список из читательского дневника при просмотре читательского дневника.

# 2 Анализ

В связи с большим масштабом системы появилась необходимость выбрать набор функций, которые будут разработаны в первую итерацию. Данное уточнение на этапе анализа будет полезно для более углубленного анализа выбранного раздела системы. Так как в требованиях к системе присутствует авторизация и связанные с ней функции, а авторизация это необходимый процесс для дальнейшей работы в системе, то было принято решение реализовать, в первую очередь, модуль авторизации, включающий в себя: регистрацию, вход в личный кабинет, выход из личного кабинета, просмотр административной панели, просмотр личного кабинета, ознакомление с документацией пользователя.

В данном разделе анализа будут представлены различные модели, где будет сделан акцент на выбранный для разработки в первой итерации функционал.

## 2.1 Модель предметной области (IDEF0)

С помощью наглядного графического языка IDEF0 система для любителей литературы представляется в виде набора взаимосвязанных функциональных блоков.

На рисунке 1 представлена модель «Система для любителей литературы» уровня А0, где на вход поступает потребность пользователя в работе с книгой и данные пользователя. В процессе работы системы с помощью пользователя, базы данных и за счет управления должностными инструкциями, правилами сообщества, пользовательским соглашением, получается результат: незарегистрированный пользователь, мнение о книге, читательский дневник и сформированные книги.

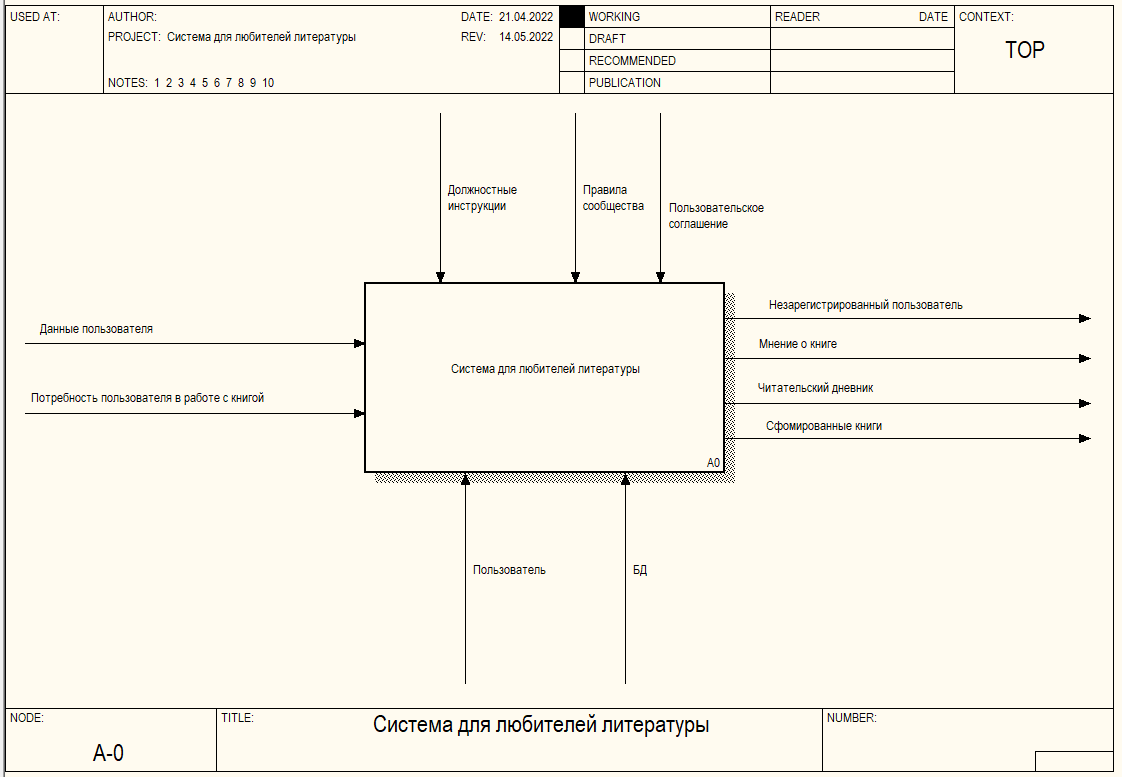


Рисунок 1 – Концептуальный уровень модели «Система любителей литературы»

На рисунке 2 представлена детализация модели «Система для любителей литературы». Детализация содержит 5 обособленных модулей: авторизация, работа с книгой, работа с заявкой, формирование читательского дневника, формирование мнения о книге.

В процессе «Авторизация» пользователь вводит данные пользователя с управлением пользовательского соглашения, в базу данных заносится информация и в результате получается авторизованный пользователь, авторизованный контент-менеджер, авторизованный модератор и неавторизованный пользователь.

Модуль «Работа с книгой» может быть инициализирован неавторизованным пользователем или авторизованным пользователем, с помощью потребности пользователя в работе с книгой. Так же в работе этого модуля необходим авторизованный контент-менеджер. Вся работа управляется должностными инструкциями и выполняется пользователем с помощью БД и в результате получаются сформированные книги, выбранная/найденная книга или не найденная книга.

На вход в блок «Работа с заявкой» поступает не найденная книга, авторизованный пользователь и авторизованный контент-менеджер. Блок управляется должностными инструкциями и правилами сообщества. В результате работы получается результат проверки заявки, например: заявка на добавление книги удовлетворена и нужно добавить книгу. Поэтому результат этого модуля входит в модуль «Работа с книгой», где происходит сам процесс добавления книги.

В процессе «Формирование читательского дневника» участвует авторизованный пользователь и для процесса необходима выбранная/найденная книга. В результате процесса получается читательский дневник.

На вход в модуль «Формирование мнения о книге» поступает авторизованный модератор, авторизованный пользователь и выбранная/найденная книга. Данная модель управляется должностными инструкциями и правилами сообщества. Результатом является мнение о книге.

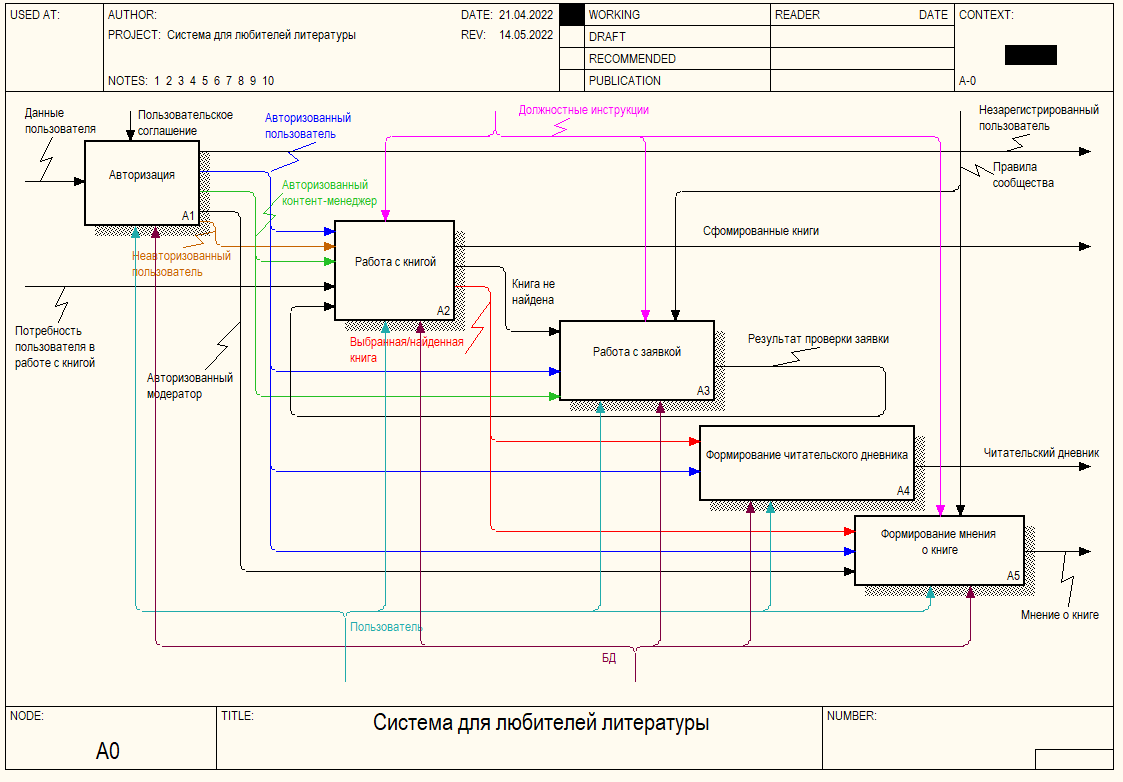


Рисунок 2 – Детализация концептуальной модели

На рисунке 3 представлена детализация модуля «Авторизация». Детализация содержит 3 процесса: регистрация, войти в личный кабинет, выйти из личного кабинета. Процессы выполняются с помощью пользователя и БД.

Процесс «Регистрация» подразумевает, что пользователь вводит данные пользователя, соглашается с пользовательским соглашением и в результате у пользователя в системе появляется учетная запись, пользователь становится зарегистрированным, но если пользователь не захотел регистрироваться, то результатом будет являться не зарегистрированный пользователь.

Процесс «Войти в личный кабинет» подразумевает, что зарегистрированный пользователь вводит свои данные, при этом процесс управляется учетной записью, и в результате получается 3 вида пользователя: авторизованный пользователь, авторизованный модератор и авторизованный контент-менеджер.

Процесс «Выйти из личного кабинета» подразумевает, что 3 вида пользователя, полученных в результате предыдущего процесса, могут выйти из личного кабинета. При этом процесс управляется учетной записью, и в результате получается неавторизованный пользователь.

Такое разделение модуля «Авторизация» обосновано необходимостью авторизованного пользователя, авторизованного модератора и авторизованного контент-менеджера в системе, для работы авторизованного пользователя в его личном кабинете и для контроля информации в системе авторизованным модератором и авторизованным контент-менеджером.

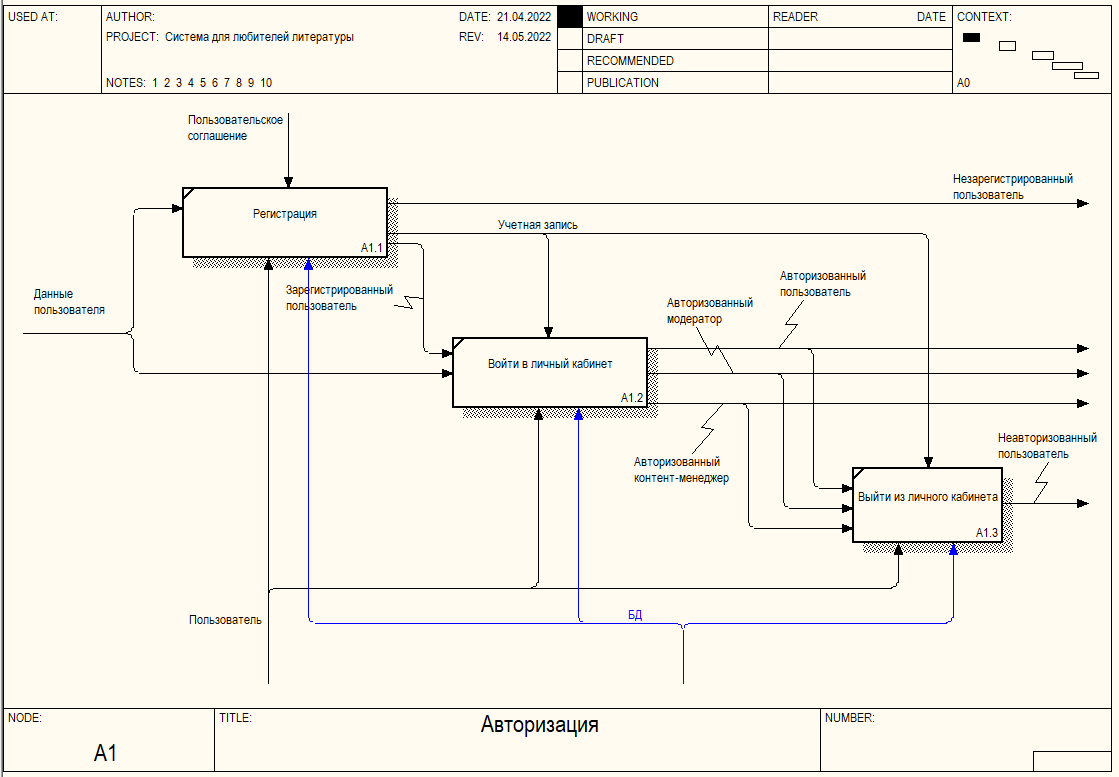


Рисунок 3 – Детализация модуля «Авторизация»

На рисунке 4 представлена детализация модуля «Работа с книгой». Детализация включает 3 процесса: посмотреть, добавить, редактировать и удалить. Процессы выполняются с помощью пользователя и БД.

Процесс «Посмотреть» подразумевает, что у авторизованного или неавторизованного пользователя возникает потребность в работе с книгой. Процесс посмотреть означает ознакомиться с информацией о книгах, жанрах, авторах, подборках книг и включает такие действия пользователя как найти через поисковую строку или через списки книг определенного жанра, автора. В результате книга может быть выбрана/найдена или книга необходимая пользователю не найдена.

Не найденная книга или результат проверки заявки (например: надо добавить книгу) запускает процесс добавления книги/автора авторизованным контент-менеджером и регулируется должностными инструкциями.

Процесс «Редактировать и Удалить» подразумевает, что авторизованный контент-менеджер может редактировать и удалить существующую или добавленную книгу, при этом процесс управляется должностными инструкциями, и в результате получается отредактированная книга.

Результатом модуля «Работа с книгой» является выбранная/найденная книга, не найденная книга и сформированные книги. Выбранную книгу пользователь может использовать в других процессах. Сформированные книги — это источник информации книгах, авторах, жанрах.

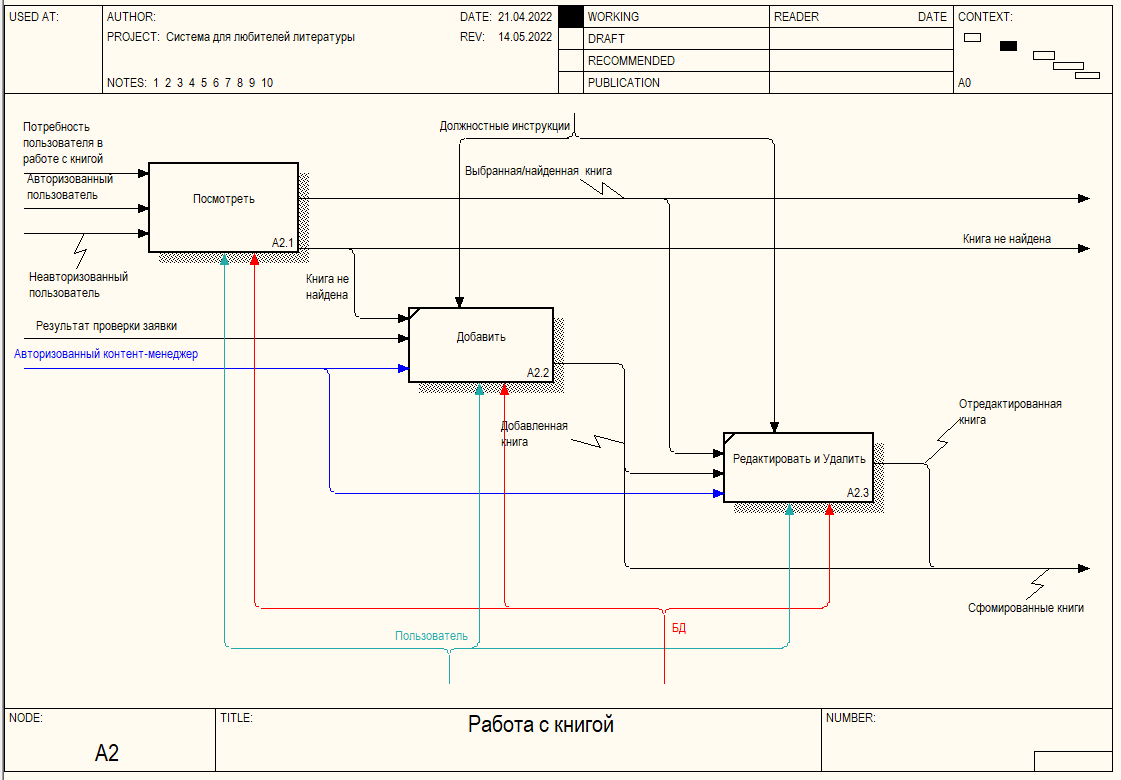


Рисунок 4 – Детализация модуля «Работа с книгой»

На рисунке 5 представлена детализация модуля «Работа с заявкой». Детализация включает 4 процесса: отправить заявку, посмотреть отправленную заявку, редактировать/удалить отправленную заявку, проверить отправленную заявку. Все действия авторизованного пользователя управляются правилами сообщества, в то время как проверка отправленной заявки авторизованным контент-менеджером управляется должностными инструкциями. Процессы выполняются с помощью пользователя и БД.

Авторизованный пользователь может отправить заявку с информацией о книге, которую он не смог найти. После отправки и до момента проверки пользователь может посмотреть отправленную заявку или редактировать/удалить ее. В результате работы модуля получается результат проверки заявки, например: необходимо добавить книгу, которую указал в заявке пользователь.

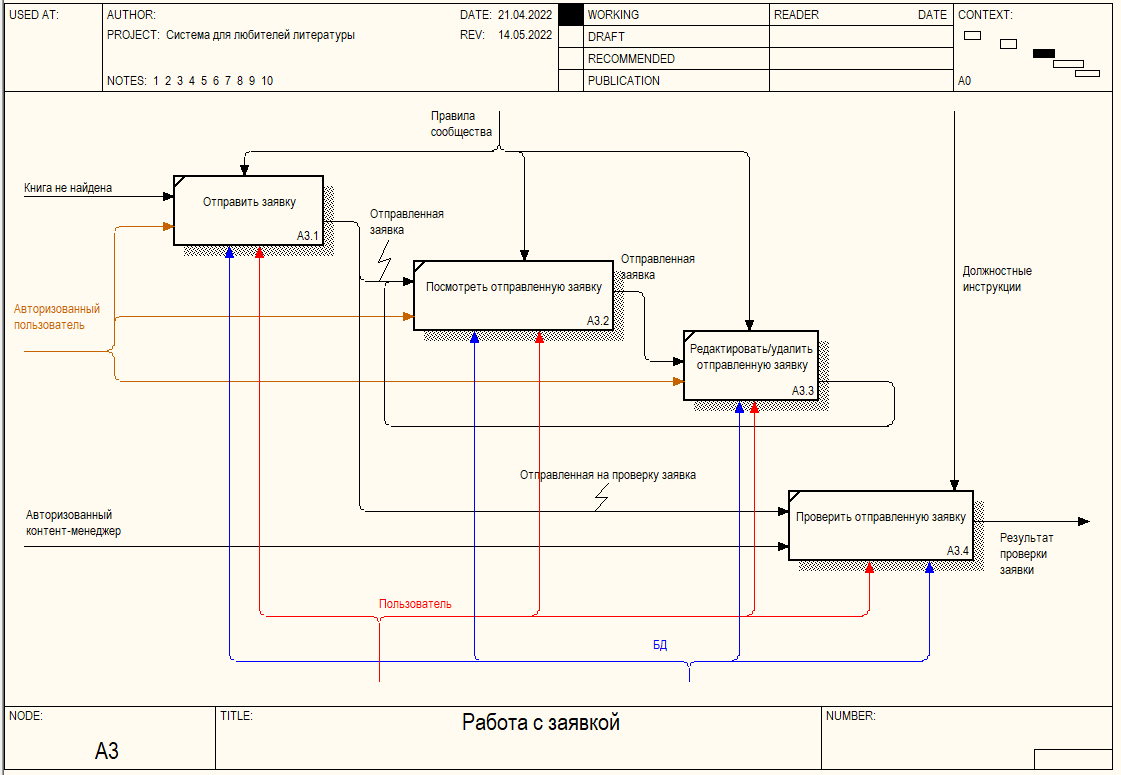


Рисунок 5 – Детализация модуля «Работа с заявкой»

На рисунке 6 представлена детализация модуля «Формирование читательского дневника». Детализация включает 2 процесса: работа со списками читательского дневника и работа с книгами из списков читательского дневника. Оба процесса инициализирует авторизованный пользователь. Процессы выполняются с помощью пользователя и БД.

В первом процессе он может посмотреть все списки, добавить список и редактировать/удалить список. Детализация первого процесса «Работа со списками читательского дневника» представлена на рисунке 7. Результатом работы являются списки читательского дневника.

Указанная на рисунке 8, детализация процесса «Работа с книгами из списков читательского дневника» содержит 4 подпроцесса: посмотреть список читательского дневника, добавить книгу в список читательского дневника, переместить книгу из одного списка в другой, удалить книгу из списка. Для работы этого процесса нужная выбранная/найденная книга, которую пользователь будет добавлять в список дневника. Результатом работы являются сформированный читательский дневник. Процессы выполняются с помощью пользователя и БД.

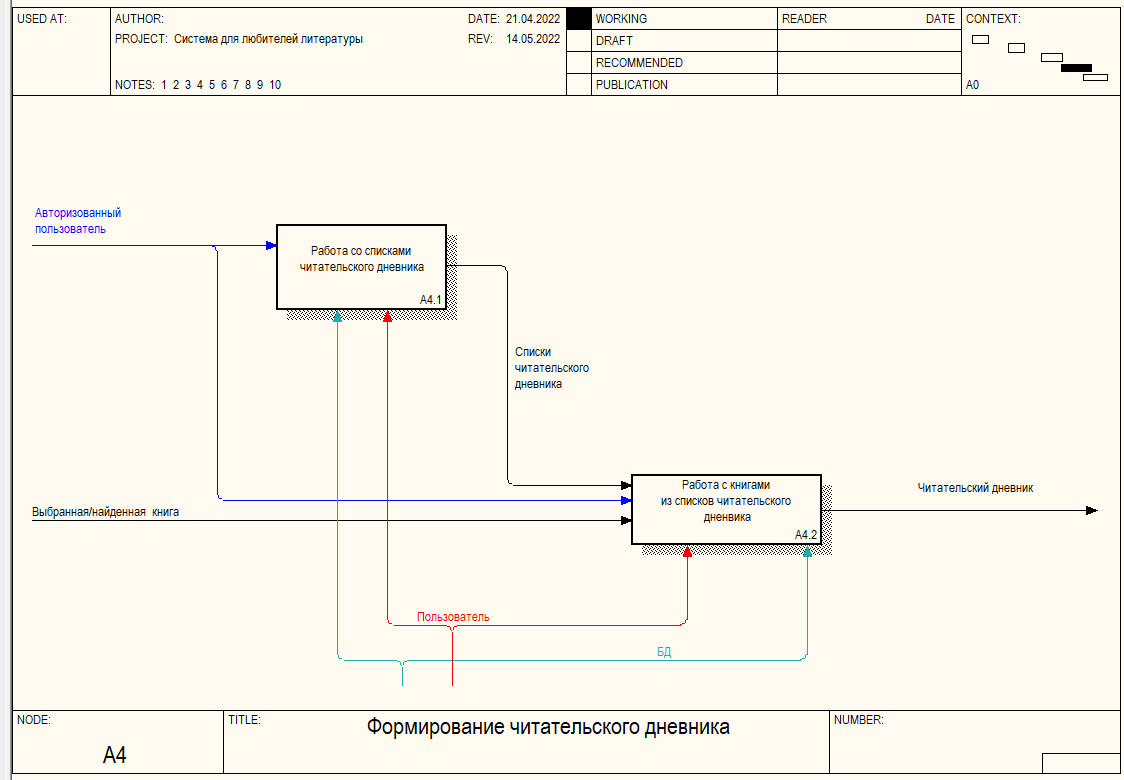


Рисунок 6 – Детализация модуля «Формирование читательского дневника»

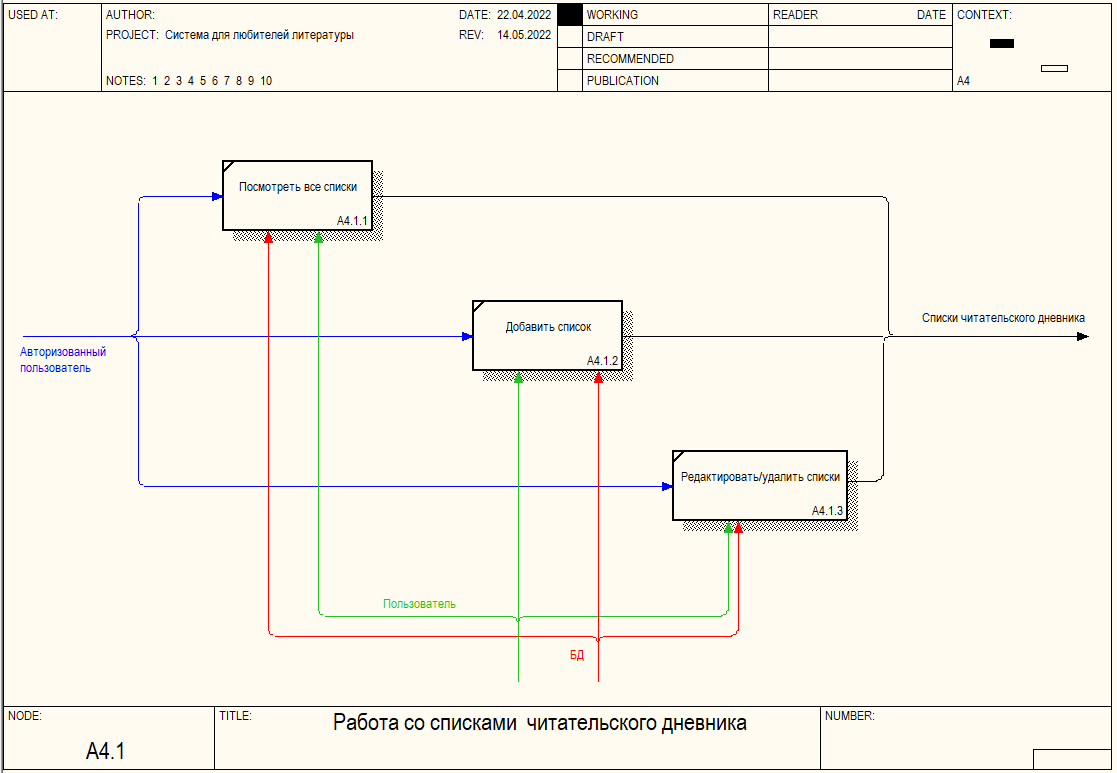


Рисунок 7 – Детализация процесса «Работа со списками читательского дневника»

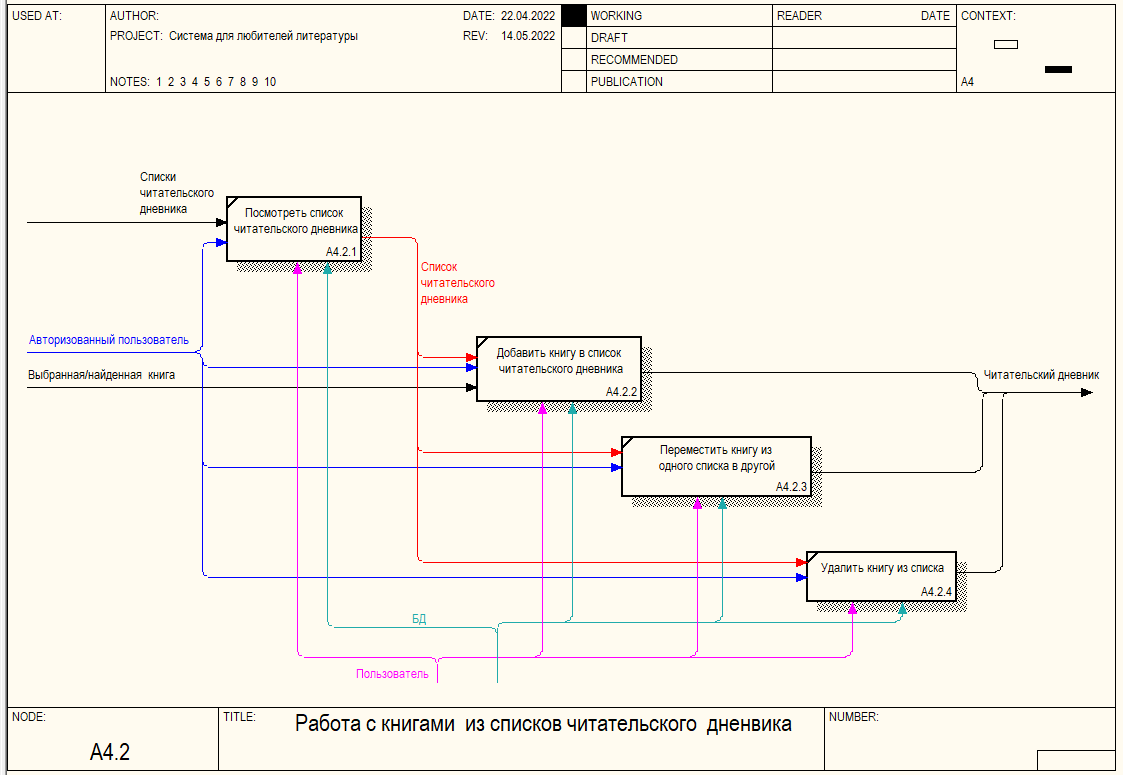


Рисунок 8 – Детализация процесса «Работа с книгами из списков читательского дневника»

На рисунке 9 представлена детализация модуля «Формирование мнения о книге». Детализация включает 3 процесса: выставление оценки книге, формирование рецензии, формирование подборки. Для всех процессов необходима выбранная/найденная книга и авторизованный пользователь. Однако для процессов «Формирование рецензии» и «Формирование подборки» нужен еще и авторизованный модератор, который будет проверять рецензии и подборки. В результате работы каждого из процессов получается рейтинг книги, рецензия и подборка книг, что можно обобщить до понятия «мнение о книге». Управляются процессы правилами сообщества и должностными инструкциями. Процессы выполняются с помощью пользователя и БД.

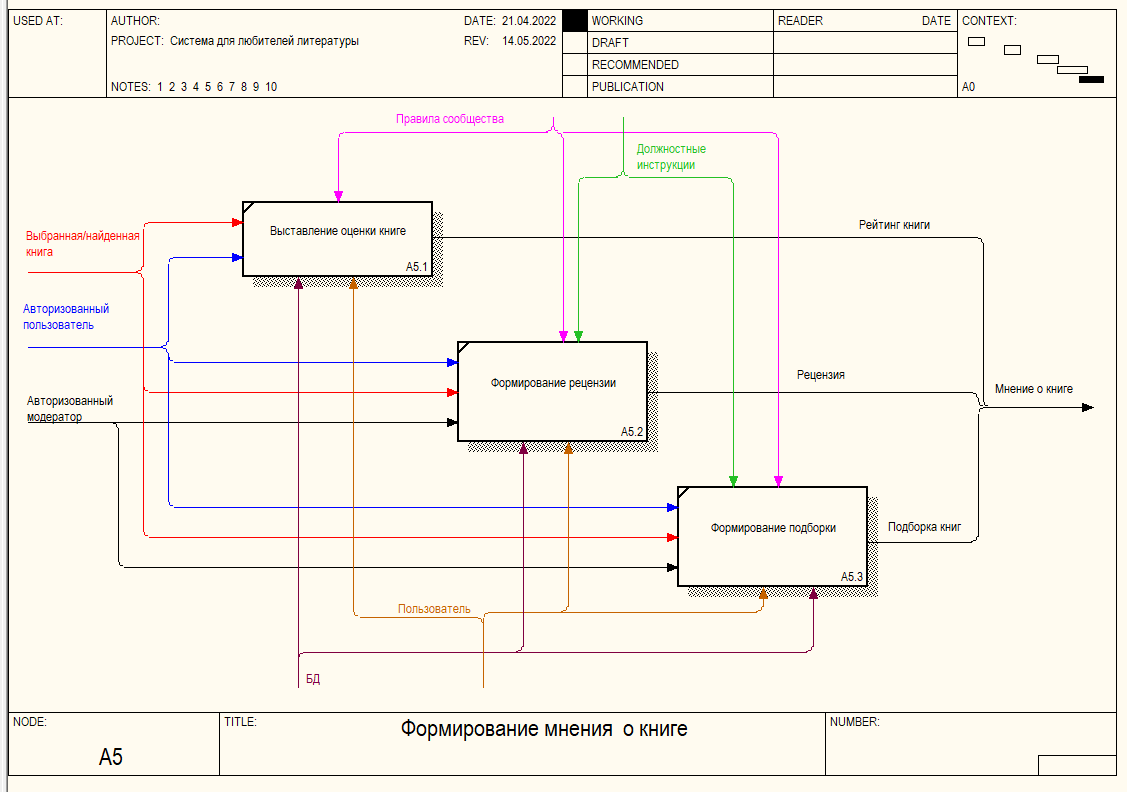


Рисунок 9 – Детализация модуля «Формирование мнения о книге»

## 2.2 Диаграмма вариантов использования UML Use Case

На рисунке 10 изображена диаграмма вариантов использования, которая описывает какой функционал разрабатываемой системы доступен каждой группе пользователей.

Каждая из групп пользователей может пользоваться системой по-своему.

Неавторизованный пользователь может:

* Работать с книгами
* Авторизоваться,что является обобщением действий: регистрации, входа в личный кабинет, выхода из личного кабинета.

Авторизованный пользователь может:

* Работать с книгами
* Авторизоваться,что является обобщением действий: регистрации, входа в личный кабинет, выхода из личного кабинета.
* Работать с заявками
* Обмениваться мнением о книгах
* Формировать читательский дневник

Модератор может:

* Работать с книгами
* Работать с заявками
* Авторизоваться,что является обобщением действий: регистрации, входа в личный кабинет, выхода из личного кабинета.

Контент-менеджер может:

* Работать с книгами
* Работать с заявками
* Авторизоваться,что является обобщением действий: регистрации, входа в личный кабинет, выхода из личного кабинета.

Почти все из представленных вариантов использования являются обобщением большого количества частных случаев действий. Такой уровень детализации обусловлен выбором для разработки в первой итерации функций регистрации, входа в личный кабинет, выхода из личного кабинета, просмотра административной панели. Не смотря на то, что функция просмотра административной панели не отображена на диаграмме, она является частным случаем вариантов использования работа с заявками и работа с книгами.

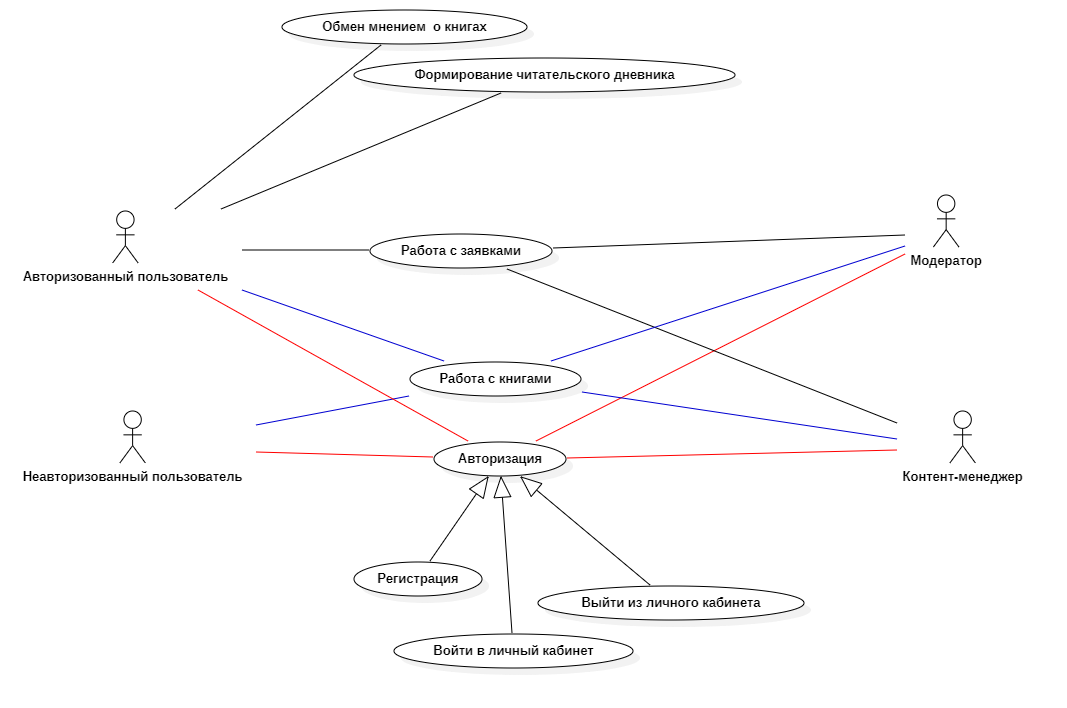


Рисунок 10 – Диаграмма вариантов использования системы любителей литературы

## 2.3 Концептуальная модель хранилища данных

На рисунке 11 представлены главные сущности системы: пользователь, роль, оценка, список читательского дневника, рецензия, заявка, книга, автор, жанр. У каждой сущности есть свои поля.

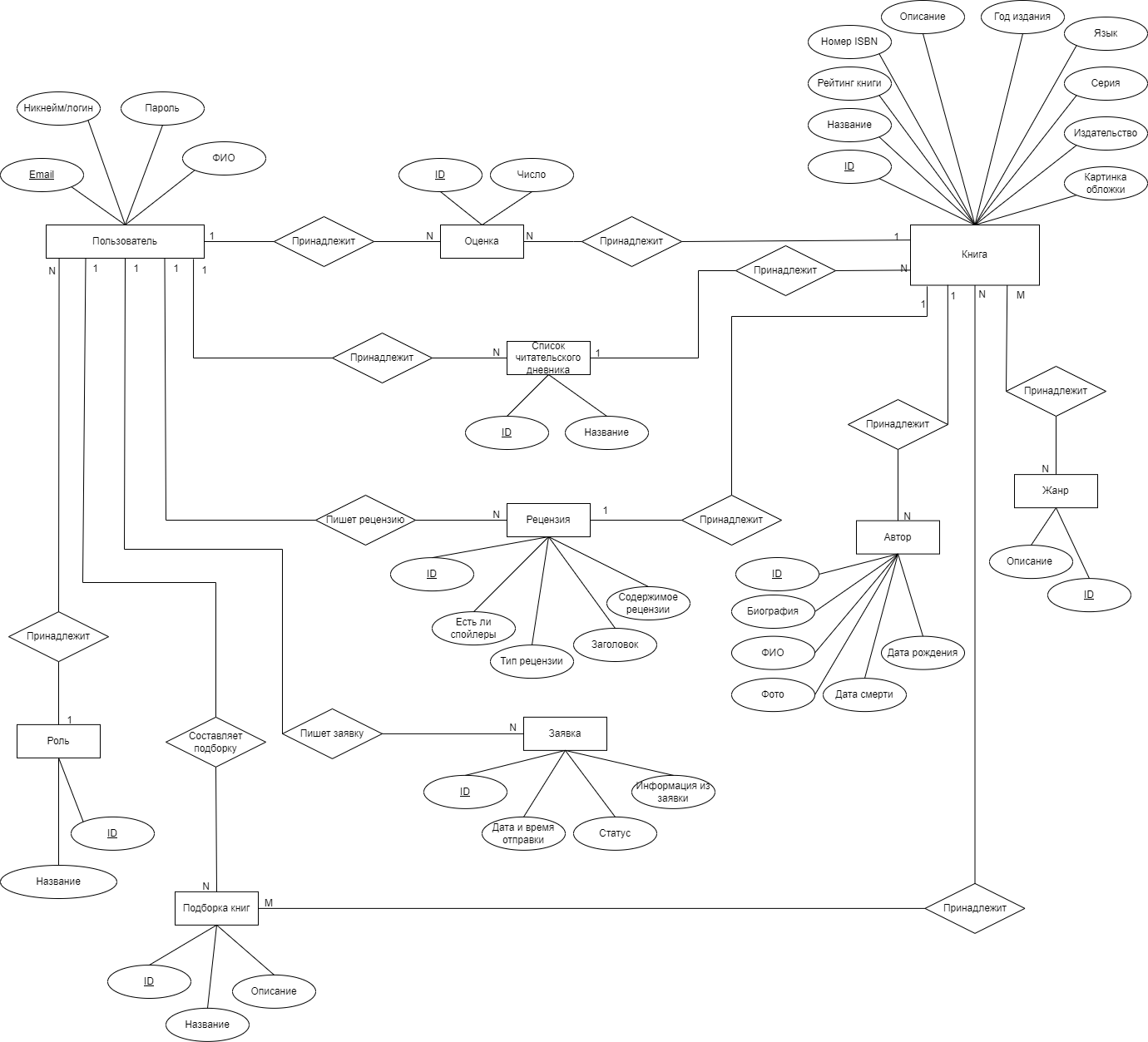


Рисунок 11 – Концептуальная модель хранилища данных системы любителей литературы.

В связи с тем, что для реализации была выбрана авторизация пользователя, то подробное описание необходимо лишь для сущностей пользователь и роль в то время, как другие сущности и связи не требуют подробного описания на данной итерации.

Сущность пользователь имеет следующие поля: email (является уникальным идентификатором), логин, пароль, ФИО.

Сущность роль имеет следующие поля: уникальный идентификатор, название. Эта сущность появилась для разделения прав доступа в системе.

Связь между сущностью пользователь и сущностью роль определяется отношением N:1, так как пользователю может принадлежать только одна роль, в то время как одной роли может принадлежать несколько пользователей.

**2.4 DFD - диаграмма потоков данных**

Диаграмма потоков данных (data flow diagram, DFD) - методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ. Диаграмма потоков данных является одним из основных инструментов структурного анализа и проектирования информационных систем.

В диаграмме потоков данных была выделена внешняя сущность–пользователь.

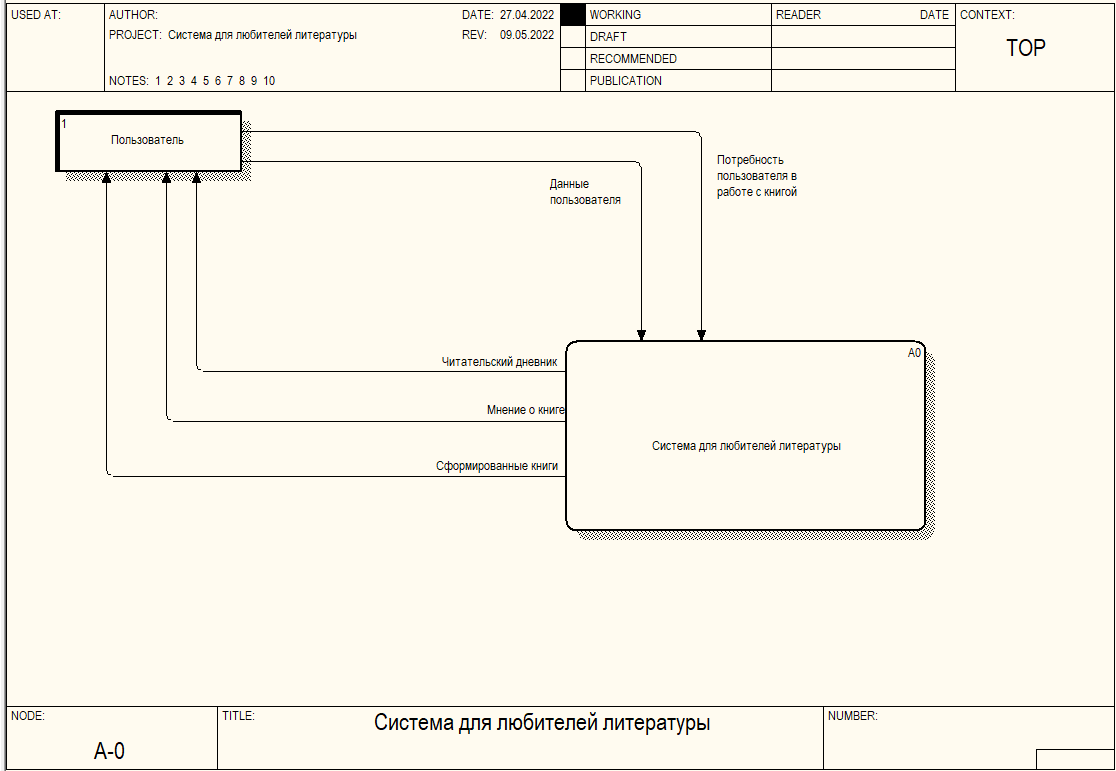


Рисунок 12 – Диаграмма потоков данных системы любителей литературы.

На рис 13 изображена детализация уровня А0, где были выявлены 5 основных модулей, каждый из которых использует информацию из хранилища данных:

1. Модуль «Авторизация»:
   * Использует данные учетной записи из сущности хранилища данных «Учетные записи»
   * Добавляет данные пользователя в сущность хранилища данных «Учетные записи»
   * Добавляет авторизованного пользователя, модератора и контент-менеджера в модуль «Работа с книгой», «Работа с заявкой», «Формирование читательского дневника», «Формирование мнения о книге».
   * Добавляет неавторизованного пользователя в модуль «Работа с книгой».
2. Модуль «Работа с книгой» создает результат – сформированные книги, с помощью следующих потоков данных:
   * Использует авторизованного пользователя, модератора и контент-менеджера из модуля «Авторизация»
   * Использует неавторизованного пользователя из модуля «Авторизация»
   * Использует информацию о книгах, жанрах и авторах из сущностей хранилища данных «Книги, жанры, авторы», «Рецензии», «Оценки книги» и «Подборки».
   * Использует результат проверки заявки из модуля «Работа с заявкой»
   * Добавляет информацию о книгах, жанрах и авторах в сущность хранилища данных «Книги, жанры, авторы»
   * Добавляет выбранную/найденную книгу в модуль «Работа с заявкой» и «Формирование читательского дневника»
3. Модуль «Работа с заявкой»
   * Использует заявки из сущности хранилища данных «Заявки»
   * Использует авторизованного пользователя, модератора и контент-менеджера из модуля «Авторизация»
   * Добавляет результат проверки заявки в модуль «Работа с книгой»
4. Модуль «Формирование читательского дневника» создает результат – читательский дневник, с помощью следующих потоков данных:
   * Использует авторизованного пользователя из модуля «Авторизация»
   * Использует выбранную/найденную книгу из модуля «Работа с книгой»
   * Добавляет списки читательского дневника с книгами в сущность хранилища данных «Списки читательского дневника»
   * Использует списки читательского дневника с книгами из сущности хранилища данных «Списки читательского дневника»
5. Модуль «Формирование мнения о книге» создает результат – мнение о книге, с помощью следующих потоков данных:
   * Использует авторизованного пользователя из модуля «Авторизация»
   * Использует выбранную/найденную книгу из модуля «Работа с книгой»
   * Добавляет рецензии в сущность хранилища данных «Рецензии»
   * Добавляет оценки книги в сущность хранилища данных «Оценки книги»
   * Добавляет подборки в сущность хранилища данных «Подборки»

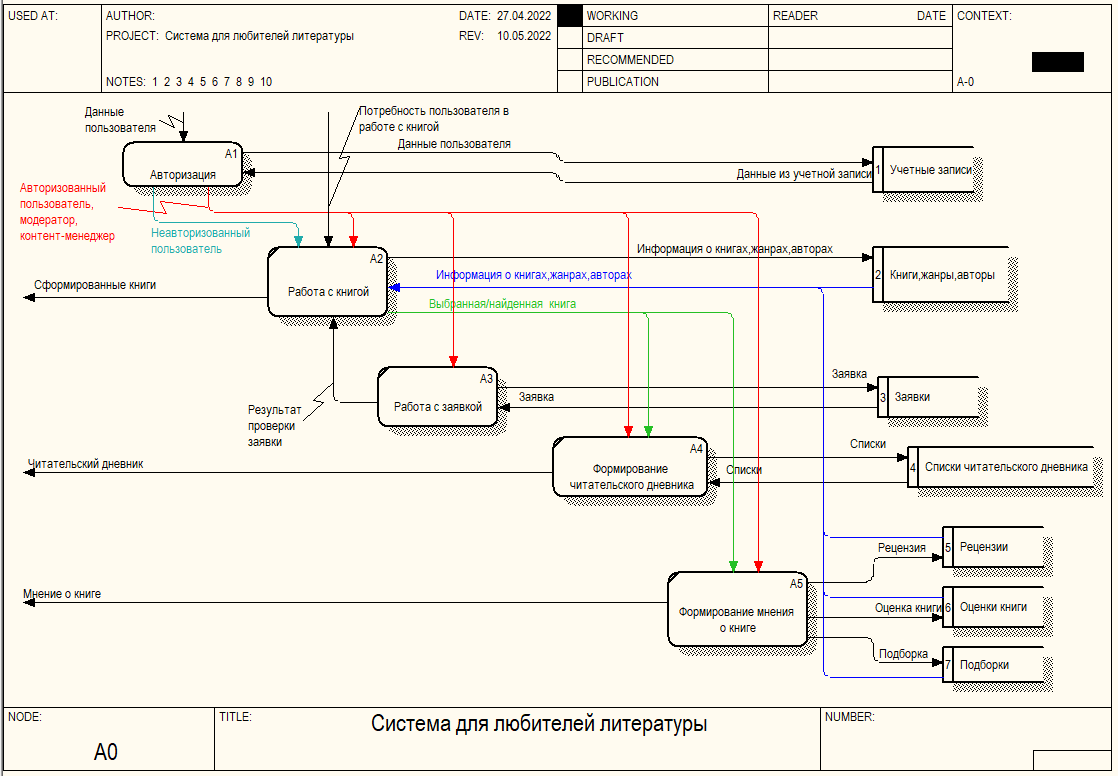


Рисунок 13 – Детализация диаграммы потоков данных системы любителей литературы

Детализация всех модулей не представлена, потому что не имеет значения на первой итерации разработки, и в последующем модель будет расширяться.

На рис 14 изображена детализация модуля «Авторизация», где были выявлены 3 основных процесса:

1. Процесс «Регистрация»
   * Добавляет данные пользователя в сущность хранилища данных «Учетные записи»
2. Процесс «Войти в личный кабинет»
   * Использует данные учетной записи из сущности хранилища данных «Учетные записи».
   * Добавляет авторизованного пользователя, модератора и контент-менеджера в модуль «Работа с книгой», «Работа с заявкой», «Формирование читательского дневника», «Формирование мнения о книге».
3. Процесс «Выйти из личного кабинета»
   * Использует данные учетной записи из сущности хранилища данных «Учетные записи»
   * Добавляет неавторизованного пользователя в модуль «Работа с книгой».

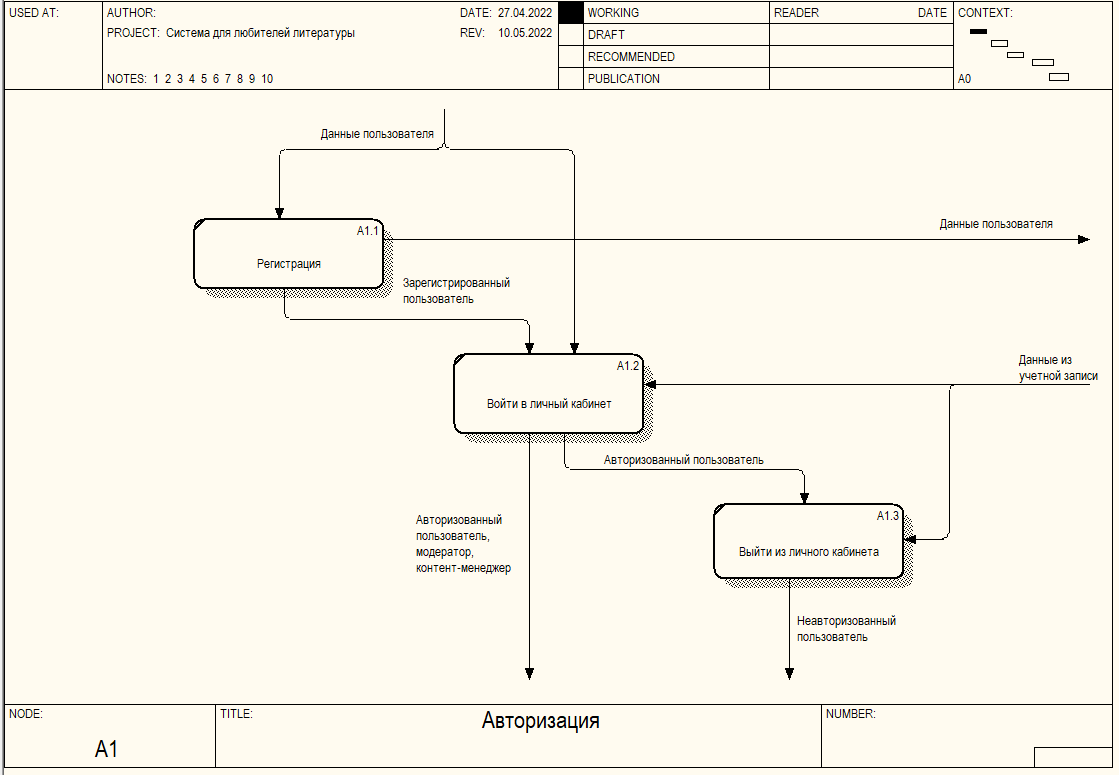


Рисунок 14 – Детализация потоков данных процесса «Авторизация»

## 2.5 Описание вариантов использования

Для описания вариантов использования были выбраны 3 варианта использования: регистрация, войти в личный кабинет, выйти из личного кабинета. Такой выбор обоснован тем, что для реализации в первой итерации был выбран модуль авторизации.

### **2.5.1** **Спецификация варианта использования «Регистрация».**

**Цель:** Зарегистрироваться.

**Активные субъекты:** незарегистрированный пользователь.

**Краткое описание:** Активный субъект может зарегистрироваться в системе.

**Предусловия:** нет.

**Постусловия:** должен быть выполнен вариант использования *«Войти в личный кабинет»*.

**Основной поток событий:**

1. Система публикует форму регистрации, состоящую из: поле ввода ФИО, поле ввода логин, поле ввода email, поле ввода пароль, кнопка "Регистрация".
2. Активный субъект заполняет поля формы.
3. Активный субъект нажимает кнопку для регистрации.
4. Система проверяет есть ли пользователь с таким email.
5. Если пользователя на шаге 4 не обнаружилось, то система добавляет пользователя в БД.
6. Система авторизует активного субъекта.
7. Система демонстрирует главную страницу с добавленными кнопками для личного кабинета и для выхода из личного кабинета.
8. Система завершает работу варианта использования.

**Альтернативные потоки событий:** Нет.

**Сценарий обработки ошибок:**

Предусловие: в основном потоке событий на шаге 4 система находит пользователя, который зарегистрирован с таким же email.

1. Система демонстрирует пользователю сообщение об ошибке «Пользователь уже существует с этим идентификатором электронной почты».

**Специальные требования:** нет.

**Дополнительные замечания:** нет.

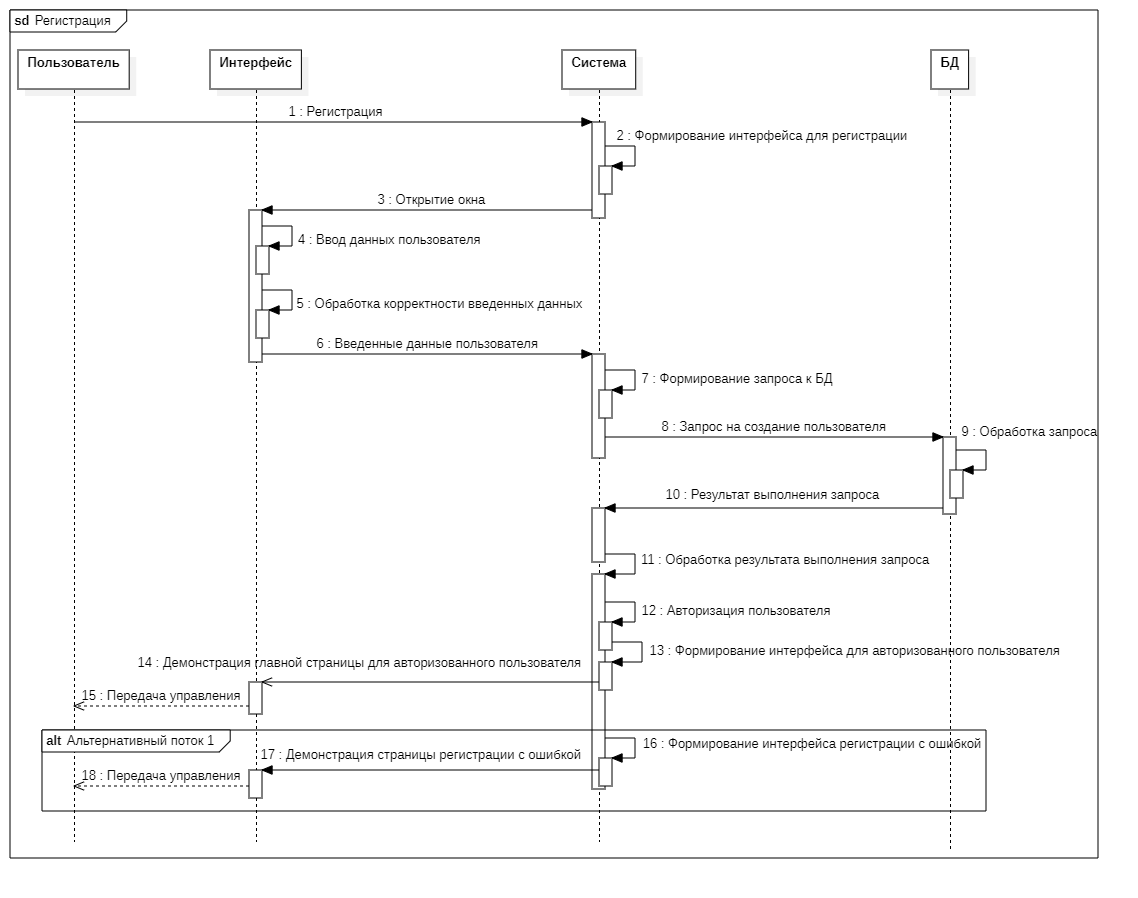


Рисунок 15 – Диаграмма последовательностей варианта использований *«Регистрация»*

### **2.5.2 Спецификация варианта использования «Войти в личный кабинет».**

**Цель:** войти в личный кабинет.

**Активные субъекты:** Неавторизованный пользователь.

**Краткое описание:** Активный субъект может войти в личный кабинет.

**Предусловия:** должен быть выполнен вариант использования *«Регистрация»*.

**Постусловия:** может быть выполнен вариант использования *«Выйти из личного кабинета» или «Работа с заявками»*.

**Основной поток событий:**

1. Система публикует форму входа в личный кабинет, состоящую из: поле ввода email, поле ввода пароль, кнопка "Войти".
2. Активный субъект заполняет поля формы.
3. Активный субъект нажимает кнопку для входа в личный кабинет.
4. Система проверяет есть ли пользователь с таким email.
5. Если пользователь на шаге 4 был найден, то система проверяет на правильность введенный пароль.
6. Если пароль на шаге 5 правильный, то система предоставляет пользователю права доступа.
7. Система демонстрирует пользователю интерфейс для авторизованного пользователя.
8. Система завершает работу варианта использования.

**Альтернативные потоки событий:** нет.

**Сценарий обработки ошибок:**

Предусловие: в основном потоке событий на шаге 4 система не находит пользователя, который зарегистрирован введенным email.

1. Система демонстрирует пользователю сообщение об ошибке «Нет пользователя с этим идентификатором электронной почты»

Предусловие: в основном потоке событий на шаге 5 система сравнила введенный пароль и пароль, которому принадлежит email, и они не совпали.

1. Система демонстрирует пользователю сообщение об ошибке «Неверный пароль или идентификатор электронной почты»

**Специальные требования:** нет.

**Дополнительные замечания:** нет.

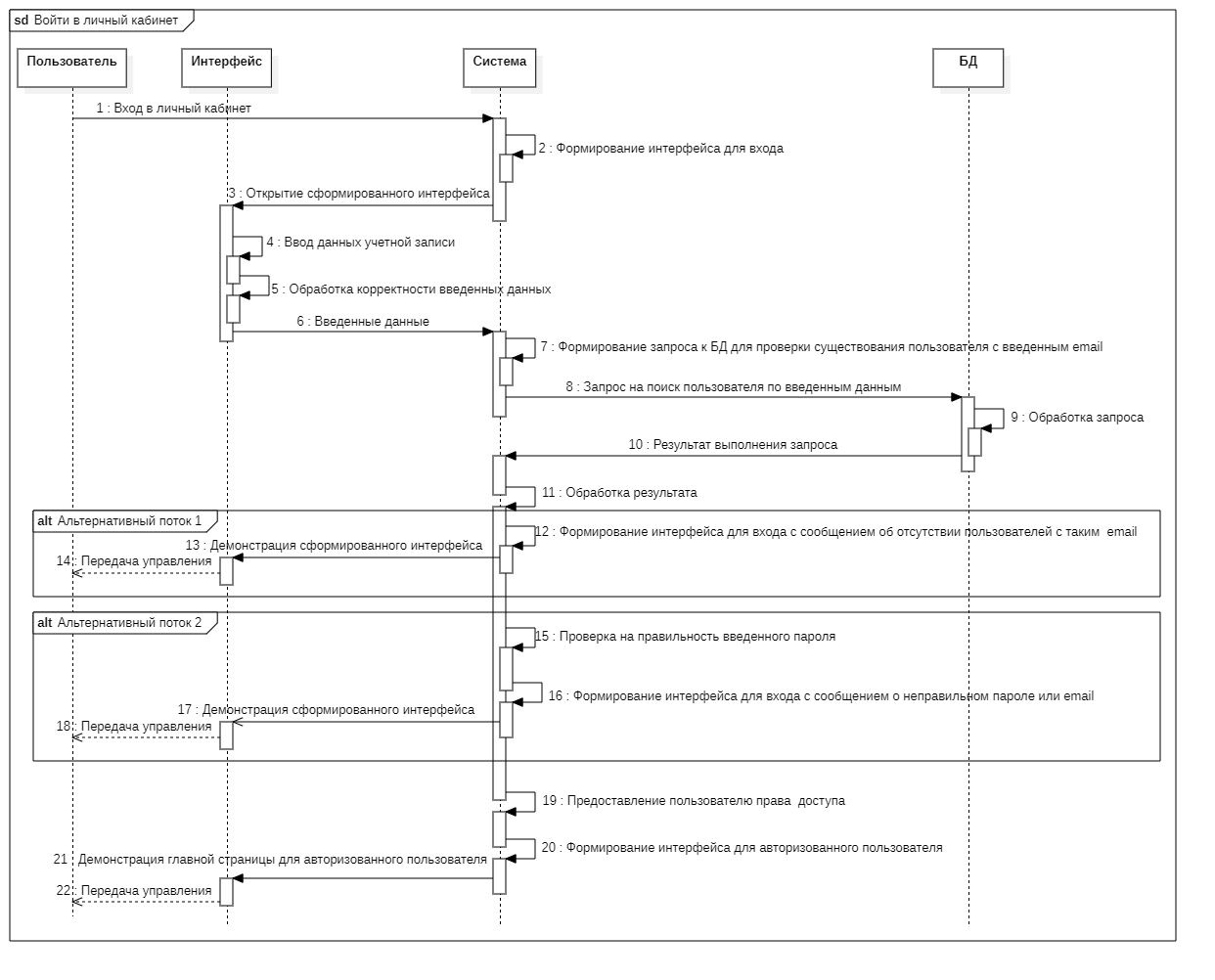


Рисунок 16 – Диаграмма последовательностей варианта использований *«Войти в личный кабинет».*

### **2.5.3 Спецификация варианта использования «Выйти из личного кабинета».**

**Цель:** найти товары, по артикулу, по модели, по названию, по производителю.

**Активные субъекты:** Пользователь.

**Краткое описание:** Активный субъект может найти товары по артикулу, модели, названию и производителю.

**Предусловия:** нет.

**Постусловия:** должен быть выполнен вариант использования *«Посмотреть список товаров»*.

**Основной поток событий:**

1. Система публикует страницу, содержащую кнопку "Выйти".
2. Активный субъект нажимает кнопку выхода.
3. Система проверяет удаляет у пользователя права доступа.
4. Система демонстрирует пользователю интерфейс для неавторизованного пользователя.
5. Система завершает работу варианта использования.

**Альтернативные потоки событий:**

1. Система демонстрирует активному субъекту сообщение: «Товары по заданным критериям не найдены».

**Сценарий обработки ошибок:** нет.

**Специальные требования:** нет.

**Дополнительные замечания:** нет.

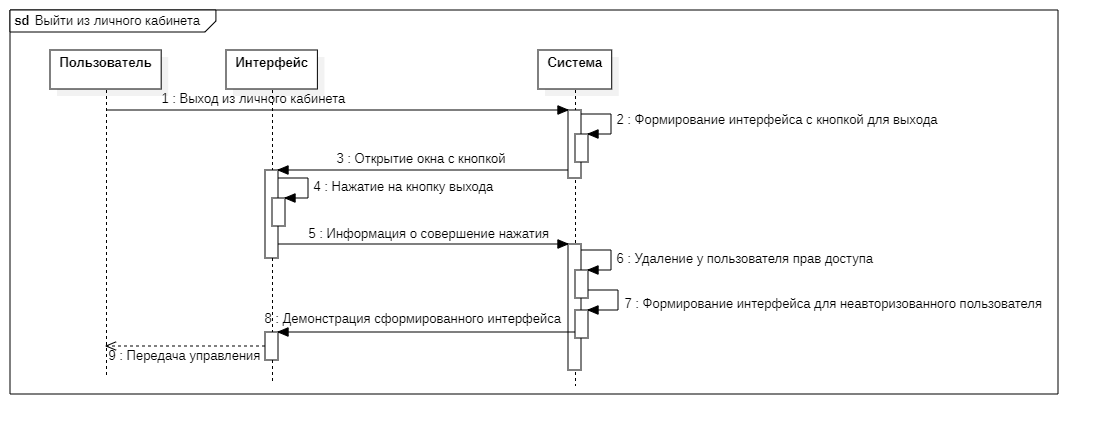


Рисунок 17 – Диаграмма последовательностей варианта использований *«Выйти из личного кабинета»*.

## 2.6 Описание интерфейса

Элементы интерфейса были сформированы на основе диаграммы вариантов использования с учетом выполнения всех разработанных функциональных требований и выбранного набора функций для реализации.

На рисунке 16 изображена диаграмма интерфейсных классов, где отображены главные составляющие интерфейса и функциональная связь между ними. Было выделено 2 набора элементов интерфейса: шапка сайта и меню. Выделение в отдельные наборы и их использование уменьшает дублирование кода. Так же было выделено 6 страниц, необходимых для реализации выбранных функций.

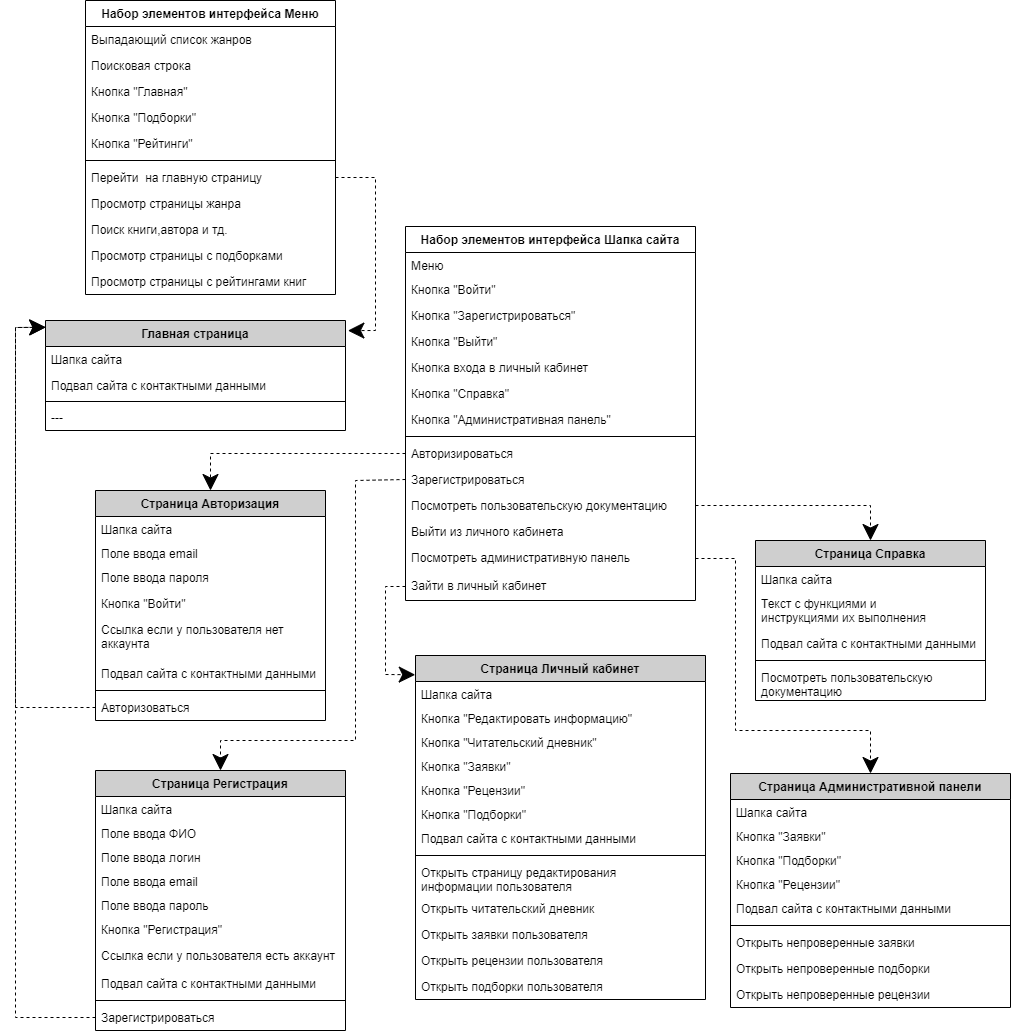


Рисунок 18 – Диаграмма интерфейсных классов

Так же написана таблица с описанием элементов интерфейсов для более лучшего понимания диаграммы интерфейсных классов.

Таблица 4 – Описание элементов интерфейсов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Роль | Страница | Функция | Элемент интерфейса | Ссылка на другую страницу |
| Все виды пользователей | Главная |  | Шапка сайта |  |
|  | Подвал сайта с контактными данными |  |
| Все виды пользователей | Шапка сайта |  | Меню |  |

Продолжение таблицы 4 – Описание элементов интерфейсов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Все виды пользователей | Шапка сайта | Открыть пользовательскую документацию | Кнопка "Справка" | Справка |
| Неавторизованный пользователь | Авторизоваться | Кнопка "Войти"прод | Авторизация |
| Неавторизованный пользователь | Зарегистрироваться | Кнопка "Зарегистрироваться" | Регистрация |
| Авторизованный пользователь,авторизованный контент-менеджер,авторизованный модератор | Выйти из личного кабинета | Кнопка "Выйти" | Главная |
| Зайти в личный кабинет | Кнопка входа в личный кабинет | Личный кабинет |
| Авторизованный контент-менеджер,авторизованный модератор | Посмотреть административную панель | Кнопка "Административная панель" | Административная панель |
| Все виды пользователей | Меню | Просмотр страницы жанра | Выпадающий список жанров |  |
| Поиск книги,автора и тд. | Поисковая строка |  |
| Перейти на главную страницу | Кнопка "Главная" |  |
| Просмотр страницы с подборками | Кнопка "Подборки" |  |
| Просмотр страницы с рейтингами книг | Кнопка "Рейтинги" |  |

Продолжение таблицы 4 – Описание элементов интерфейсов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Неавторизованный пользователь | Авторизация |  | Шапка сайта |  |
|  | Поле ввода email |  |
|  | Поле ввода пароля |  |
| Авторизоваться | Кнопка "Войти" | Главная |
|  | Ссылка если у пользователя нет аккаунта | Регистрация |
|  | Подвал сайта с контактными данными |  |
| Неавторизованный пользователь | Регистрация |  | Шапка сайта |  |
|  | Ссылка если у пользователя есть аккаунт | Авторизация |
|  | Поле ввода ФИО |  |
|  | Поле ввода логин |  |
|  | Поле ввода email |  |
|  | Поле ввода пароль |  |
| Зарегистрироваться | Кнопка "Регистрация" | Главная |
|  | Подвал сайта с контактными данными |  |
| Авторизованный пользователь,авторизованный контент-менеджер,авторизованный модератор | Личный кабинет |  | Шапка сайта |  |
| Открыть страницу редактирования  информации пользователя | Кнопка "Редактировать информацию" |  |
| Открыть читательский дневник | Кнопка "Читательский дневник" |  |
| Открыть заявки пользователя | Кнопка "Заявки" |  |

Продолжение таблицы 4 – Описание элементов интерфейсов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Авторизованный пользователь,авторизованный контент-менеджер,авторизованный модератор | Личный кабинет | Открыть рецензии пользователя | Кнопка "Рецензии" |  |
| Открыть подборки пользователя | Кнопка "Подборки" |  |
|  | Подвал сайта с контактными данными |  |
| Авторизованный контент-менеджер и авторизованный модератор | Административная панель |  | Шапка сайта |  |
| Открыть непроверенные заявки | Кнопка "Заявки" |  |
| Открыть непроверенные подборки | Кнопка "Рецензии" |  |
| Открыть непроверенные рецензии | Кнопка "Подборки" |  |
|  | Подвал сайта с контактными данными |  |
| Все виды пользователей | Справка |  | Шапка сайта |  |
|  | Посмотреть пользовательскую документацию | Текст с функциями и инструкциями их выполнения |  |
|  |  | Подвал сайта с контактными данными |  |

# 3 Проектирование программного продукта

В данном разделе будет произведено проектирование программного продукта для выбранного на этапе анализа модуля «Авторизация», в который входят функции: регистрация, вход в личный кабинет, выход из личного кабинета, просмотр административной панели, просмотр личного кабинета, ознакомление с документацией пользователя. Для выбора инструментов проектирования и дальнейших действий необходимо выбрать тип архитектуры программного обеспечения. Архитектура программного обеспечения – совокупность важнейших решений об организации программной системы.

Для реализации поставленной задачи необходимо создавать именно web-приложение. Благодаря этому, возможно обеспечить доступ из различных устройств с любой точки мира. Исходя из выбранного вида программного обеспечения, для реализации системы была выбрана архитектура «Клиент-Сервер». Это такая архитектура, которая состоит из двух физических машин в качестве сервера и клиента. Клиент содержит программу-клиент, которая предоставляет для пользователя интерфейс, посылает запросы на сервер и получает ответы на запросы. В то же время, сервер хранит данные и содержит программу, которая отвечает за логику обработки данных из поступивших запросов. Преимущества данной архитектуры: обеспечивает изоляцию операций управления данными, обработки данных и операций представления; легко модифицировать приложения; хорошая производительность и масштабируемость.

## 3.1 Выбор и обоснование программных инструментов

В данном разделе приводится описание и аргументация выбора программных инструментов, используемых при проектировании и реализации системы.

Для построения логической модели хранилища данных был выбран Draw.io. Это бесплатное онлайн-приложение для создания диаграмм рабочих процессов, BPM, организационных, сетевых. Обладает современным и понятным интерфейсом, а также позволяет сохранить проект в форматах JPG, PNG и другие. Именно за эти особенности и был выбран данный инструмент.

Для построения физической модели хранилища данных, была выбрана визуальная система для проектирования различных типов баз данных – DBDesigner. Эта система предлагает множество удобных функций: создание баз данных, обратный инжиниринг для MySQL, Oracle, MSSQL и любого другого типа ODBC, а также синхронизация между моделью и самой базой данных и несколько вариантов экспорта проекта. Выбор именно этой системы проектирования физической модели хранилища данных, обоснован: возможностью бесплатного использования, функцией экспорта схемы созданной базы данных и легкостью использования пользовательского интерфейса.

На этапе проектирования так же необходимо построить диаграмму развертывания, диаграмму пакетов, диаграммы классов, диаграмму последовательности. И все эти диаграммы должны быть написаны на языке графического описания UML (Unified Modeling Language). Поэтому появляется необходимость в StarUML – это программный инструмент визуального моделирования, который поддерживает стандартизованный язык графического описания UML. Наличие опыта использования и возможность создания 11 видов диаграмм стало решающим при выборе.

Для разработки макетов пользовательского интерфейса выбран онлайн-сервис Figma, который предназначен для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени. Данный онлайн-сервис был выбран, исходя из следующих преимуществ: наличие опыта использования, бесплатный тарифный план для одного пользователя, кроссплатформенность, сохранение истории изменений макетов.

Для разработки самой системы были выбраны такие языки программирования как JavaScript и TypeScript.

JavaScript – это мультипарадигменный язык программирования, который поддерживает объектно-ориентированный, императивный и функциональный стили. Является одним из самых популярных, поэтому и был выбран для разработки со стороны сервера. Однако, обрабатывать все запросы к серверу с помощью JavaScript возможно только благодаря среде выполнения Node.js.

Так же необходимо использовать со стороны сервера Express js. Это фреймворк web-приложений, который является стандартным каркасом для Node.js. Он спроектирован для создания веб-приложений и API. Предоставляет ряд готовых абстракций, которые упрощают создание сервера и серверной логики, в частности, обработка отправленных форм, работа с куками, CORS и т.д.

Для разработки на JavaScript потребуется текстовый редактор для написания кода. В качестве текстового редактора будет использована такая программа как Visual Studio Code. Она бесплатна, имеет множество возможностей и может быть установлена как на Windows, так и на Linux и MacOS.

Программа Postman будет использована для тестирования работы API, а также для отправки запросов POST и GET. Она имеет графический интерфейс, поэтому легко осваивается даже новичками.

Angular представляет фреймворк от компании Google для создания клиентских приложений. Именно этот фреймворк будет использован со стороны клиентской части. Angular позволяет создавать большие и сложные по части бизнес-логики приложения. Данный фреймворк предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация и так далее. Angular использует в качестве языка программирования TypeScript.

Так же в клиентской части системы будет использован Bootstrap. Это свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

Для работы с базой данных необходимо выбрать систему управления базами данных. СУБД – это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных. Среди множества СУБД была выбрана объектно-реляционная система управления базами данных PostgreSQL. Использование PostgreSQL бесплатно, она является одной из наиболее популярных. По этим причинам и был сделан выбор.

## 3.2 Проектирование хранилища данных

Для хранилища данных была выбрана реляционная модель данных из-за наглядности табличного представления данных и удобства работы с ними.

При преобразовании концептуальной модели на основе выбранной модели данных в логическую была построена модель, показанная на рисунке 18. Модель не зависит от особенностей выбранной СУБД для физической реализации базы данных. Информация представляется в виде данных и логических связей между данными вне зависимости от того, что представляют собой данные, и какие технические средства будут использованы для хранения данных.

Для сущности «Пользователь» из концептуальной модели хранилища данных была создана таблица Пользователь, которая содержит атрибуты: электронная почта, никнейм, пароль, ФИО. Атрибут электронная почта является первичным ключом данной сущности.

Для сущности «Роль» из концептуальной модели хранилища данных была создана таблица Роль, которая содержит атрибуты: код роли и название роли. Атрибут код роли является первичным ключом данной сущности.

Так же была установлена связь между таблицами посредством использования первичного ключа код роли из таблицы «Роль» в таблице «Пользователь» в качестве внешнего ключа. Тем самым становится возможным утверждение, что одному пользователю может принадлежать одна роль, в то время как одной роли может принадлежать несколько пользователей.

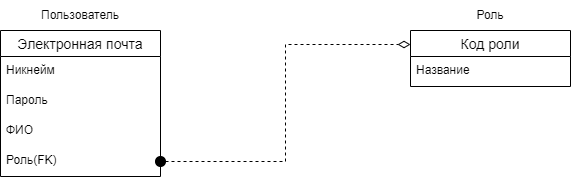


Рисунок 19 – Логическая модель хранилища данных

На этапе создания физической модели хранилища данных было произведено описание конкретной реализации базы данных, размещаемой во внешней памяти компьютера. На этапе выбора и обоснования программных инструментов была выбрана и СУБД – PostgreSQL. В онлайн сервисе DBDesigner была спроектирована структура хранения данных на физическом уровне (рис. 19). Названия таблиц были изменены на английские, так же, как и названия атрибутов. У всех атрибутов появился тип данных и ограничения по необходимости.

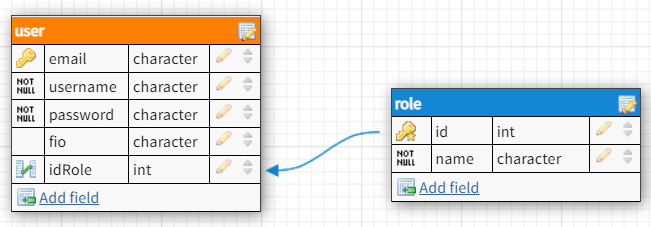


Рисунок 20 – Физическая модель хранилища данных

## 3.3 Описание процесса авторизации

Поскольку известно, что почти весь интернет так или иначе построен на протоколе HTTP/HTTPS и что он не отслеживает состояние, то есть при каждом запросе HTTP ничего не знает, что происходило до этого, он лишь передает запросы, то возникает следующая проблема: так как аутентификация пользователя происходит с помощью email и пароля, то при любом следующем запросе приложение не будет знать все тот же ли этот человек, и поэтому придётся каждый раз заново вводить email и пароль. Решением данной проблемы является токен, а конкретно его самая популярная реализация - JSON Web Tokens (JWT).

JSON Web Token (JWT) — это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON. Фактически это просто строка символов (закодированная и подписанная определенными алгоритмами) с некоторой структурой. И эта строчка передается клиентом приложению при каждом запросе, когда есть необходимость идентифицировать и понять кто прислал этот запрос.

Процесс авторизации с помощью JWT следующий: первым делом пользователь проходит аутентификацию, конечно же, если не делал этого ранее и в этом есть необходимость, а именно, вводит свой email и пароль. Далее приложение выдаст ему токен. Пользователь сохраняет его себе в локальное хранилище. Затем, когда пользователь с клиентской части делает запрос к серверной части, то добавляется полученный ранее токен. И, наконец, север, получив данный запрос с токеном, проверяет, что данный токен действительный, считывает полезные данные, которые помогут идентифицировать пользователя и проверить, что он имеет право на запрашиваемые ресурсы. Таким образом происходит основная логика работы с JSON Web Tokens.

## 3.4 Проектирование клиентской части разрабатываемой системы

В процессе проектирования клиентской части разрабатываемой системы необходимо учитывать особенности выбранного фреймворка Angular. Описание проектирования клиентской части можно разделить на 2 части: классы бизнес-логики и классы представлений.

Классы бизнес-логики необходимо разместить в папке core клиентской части. Выделение этих классов от классов представлений позволит достичь сосредоточенности кода компонентов непосредственно на работе с представлением. Кроме того, это решит проблему повторения кода, если нам потребуется выполнить одну и ту же задачу в разных компонентах и классах.

Для удобства работы с данными между страницами сайта необходимо реализовать интерфейсы User, Profile, Errors в папке core\models. Для процесса авторизации, необходимо реализовать класс HttpInterceptor, с помощью которого будет происходить добавление токена в Header запроса перед отправкой каждого запроса. Реализованный от HttpInterceptor класс необходимо назвать HttpTokenInterceptor и расположить в папке core\interceptors.

Для классов, которые будут инкапсулировать бизнес-логику необходимо использовать сервисы. Класс ApiService отвечает за отправку запросов и принятие ответов с сервера. Класс UserService и ProfilesService содержат поля и методы, необходимые для формирования запросов к серверу, а также для передачи информации о пользователе между представлениями. Класс JwtService необходим для работы с токеном: метод для сохранения, удаления и get метод. AuthGuard отвечает за защиту доступа. Все эти классы необходимо расположить в папке core\services.

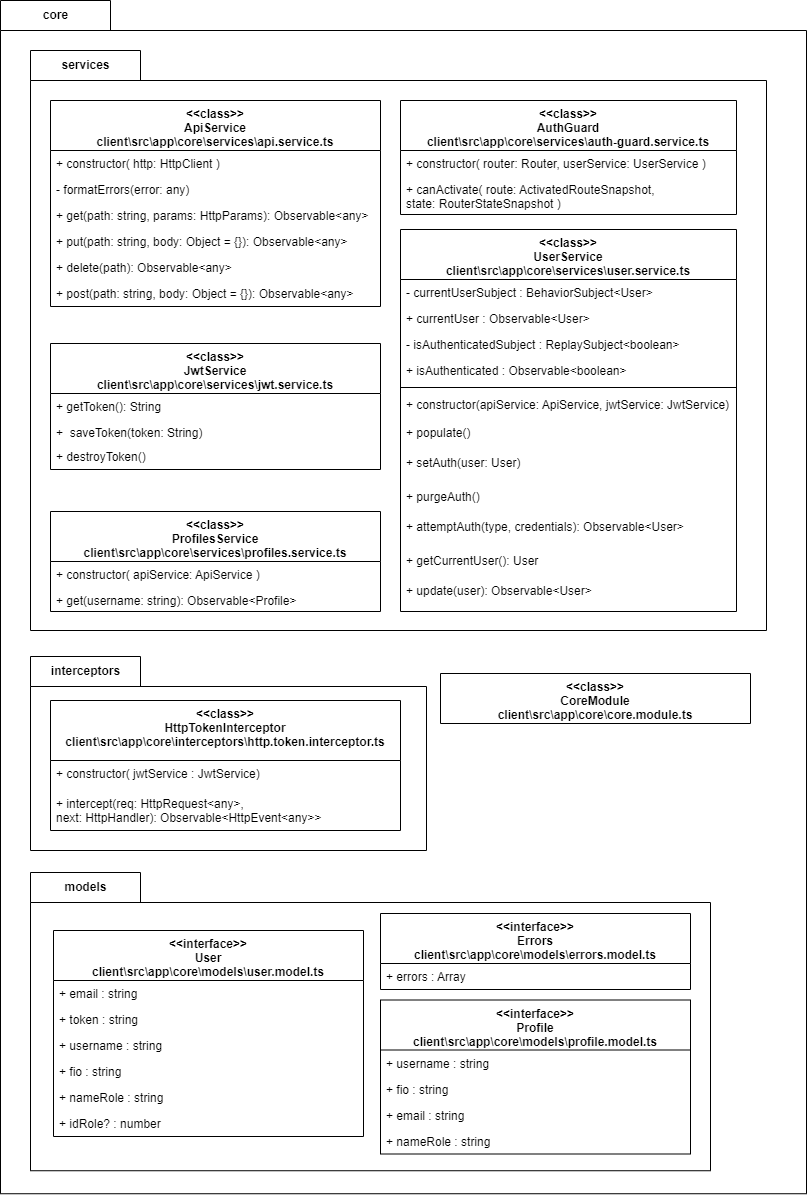


Рисунок 21 – Диаграмма классов бизнес - логики

При запуске приложения первым выполняется код, который определен в файле main.ts. Именно этот файл является точкой входа в программу.

Клиентское приложение с использованием фреймворка Angular должно состоять из отдельных модулей. Приложение должно иметь один корневой модуль, который, согласно условностям, необходимо назвать AppModule. Для работы данного модуля необходимо создать классы AppRoutingModule(должен отвечать за маршрутизацию), AppComponent(должен подгружать app.component.html).

Так же неотъемлемой частью приложения являются компоненты - элементы представления, которые управляют отображением представления на экране.

Для каждого представления необходимо выделить папку, внутри которой будет лежать класс модуля, компонента, маршрутизации и файл HTML. Так же необходимо не забывать об особенностях файлов HTML: есть файл index.html, внутрь которого встраиваются header.component.html, footer.component.html а также файлы модулей приложения. Представление для входа в личный кабинет и регистрации должно быть создано в одном и том же модуле и компоненте. В папке shared должны содержатся классы и файлы необходимые для шапки и подвала сайта. Сама же папка shared должна находиться в папке src. Более подробное описание необходимой реализации представлено на рисунке 22 и 23.



Рисунок 22 – Диаграмма классов представлений

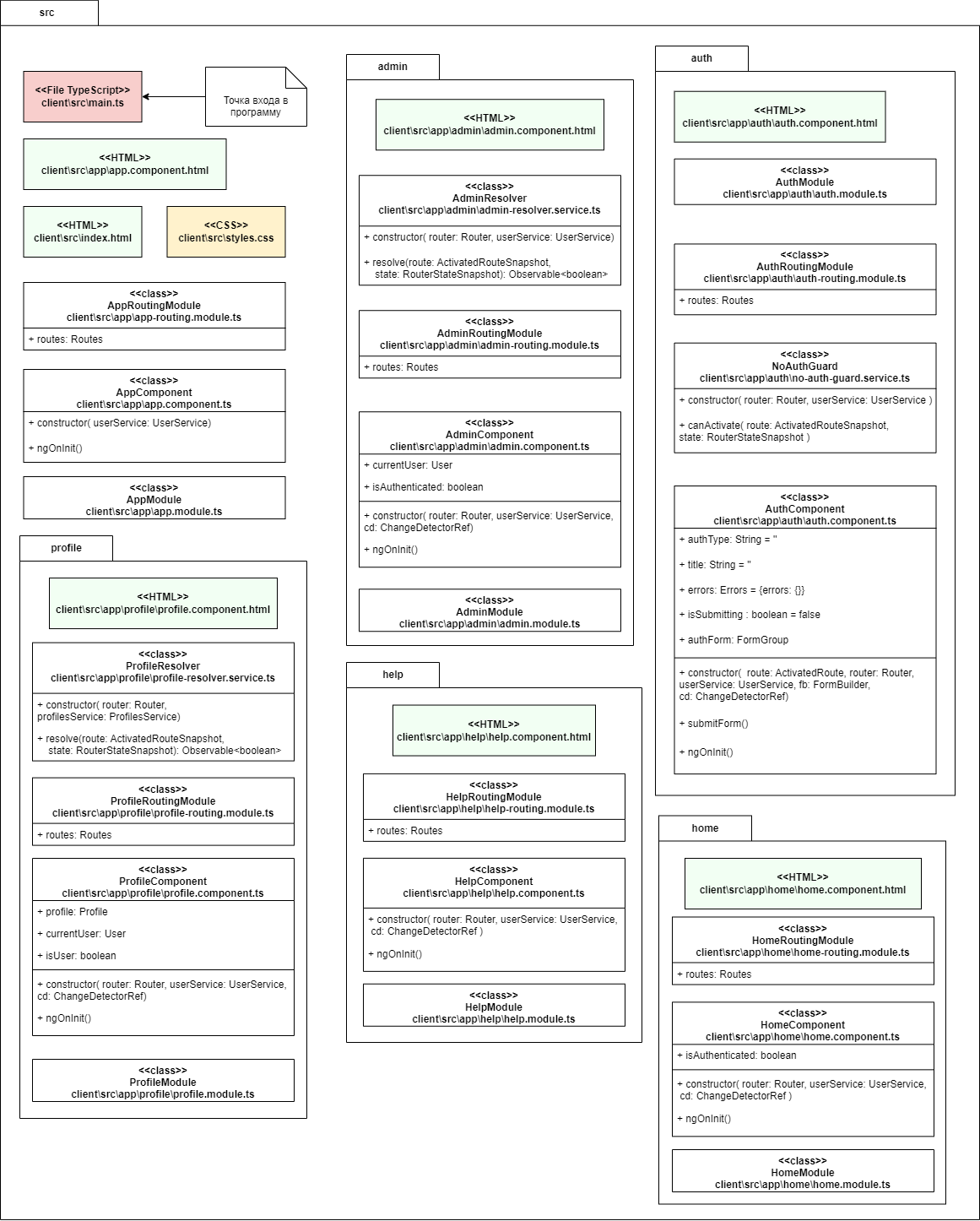


Рисунок 23 – Диаграмма классов представлений

## 3.5 Проектирование серверной части разрабатываемой системы

В связи с особенностью выбранного языка программирования JavaScript для серверной части, было принято использовать функциональный стиль программирования, в связи с уже имеющимся опытом. При проектировании серверной части, необходимо сказать о 4 составляющих серверного приложения: точка входа в программу, маршруты, методы, модели и подключение к БД.

Точкой входа в программу будет являться файл index.js, который должен располагаться в корневой папке server серверной части. Этот файл должен отвечать за подключение фреймворка Express, создание объекта приложения, определение дочерних подмаршрутов со своими обработчиками и начало прослушивания подключения на выделенном порту.

За подключение к БД будет отвечать файл dbConnection.js, который должен располагаться в корневой папке server серверной части. Этот файл должен содержать переменную sequelize, которая содержит информацию о подключении.

Необходимо выделить подмаршруты, в связи с тем, что набор функций, которые будет выполнять сервер, можно сгруппировать в 2 группы. Объект Router позволяет связать подобный функционал в одно целое и упростить управление им.

Таблица 5 – Список объектов Router

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Маршрут | Назначение | Файл |
| userRoute | /api | Работа с информацией пользователя | users.js |
| profileRoute | /api/profiles | Предоставление информации о пользователе для личного кабинета | profile.js |

Для реализации необходимо определить не только главные маршруты, но и дочерние подмаршруты, а также спроектировать методы, которые будут выполнять всю работу. Файлы, отвечающие за маршруты, необходимо разместить в server\routes\.

Таблица 6 – Дочерние подмаршруты userRoute

|  |  |  |  |
| --- | --- | --- | --- |
| Назначение | Метод | Полный маршрут | Метод |
| Регистрация пользователя | POST | http://localhost:8081/api/users | createUser |
| Идентификация и аутентификация пользователя | POST | http://localhost:8081/api/users/login | loginUser |
| Предоставление информации о пользователе по email, где email указан в строке параметров | GET | http://localhost:8081/api/user/:email | authByToken, getUserByEmail |
| Предоставление информации о пользователе по email, где email указан в строке запроса | GET | http://localhost:8081/api/user | authByToken, getUserByEmail |

Таблица 7 – Дочерние подмаршруты profileRoute

|  |  |  |  |
| --- | --- | --- | --- |
| Назначение | Метод | Полный маршрут | Метод |
| Предоставление информации о пользователе для личного кабинета | GET | http://localhost:8081/:username | authByToken, getProfile |

Так же, серверу необходимо работать с базой данных, но что бы делать это не напрямую SQL запросами, предлагается использовать модели библиотеки ORM Sequelize. Модели должны описывать структуру хранящейся в базе данных информации. Файлы с моделями необходимо разместить в папке server\models\.

Таблица 8 – Модели

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Назначение | Поля | Файл |
| Role | Описывает структуру сущности «Роль» в БД | id | Role.js |
| name |
| User | Описывает структуру сущности «Пользователь» в БД | email | User.js |
| username |
| fio |
| password |

Таблица 9 – Методы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Функция | Описание работы системы | Каталог | Файл |
| getProfile | Просмотр личного кабинета | Отправляет запрос в БД c использованием модели User и Role, затем возвращает информацию о пользователе для показа в личном кабинете | server\controllers\ | profile.js |
| createUser | Регистрация | Проверяет корректность введенных данных;  Проверяет на наличие пользователя с введенным email;  Хеширует пароль, который ввел пользователь с помощью метода hashPassword;  Создает пользователя по введенным данным с помощью модели User;  Создает и присваивает пользователю JWT токен с помощью метода sign; | server\controllers\ | users.js |

Продолжение таблицы 9 – Методы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| loginUser | Вход в личный кабинет | Проверяет корректность введенных данных;  Проверяет на наличие пользователя с введенным email;  Проверяет на правильность пароль, который ввел пользователь, с помощью метода matchPassword;  Авторизует пользователя с помощью метода sign; | server\controllers\ | users.js |
| getUserByEmail | Правильный показ кнопки административная панель | Отправляет запрос в БД c использованием модели User, затем возвращает информацию о пользователе | server\controllers\ |
| sign |  | Создает JWT токен | server\utils\ | jwt.js |
| decode |  | Декодирует JWT токен |
| hashPassword |  | Хеширует пароль | password.js |
| matchPassword |  | Сравнивает хешированный пароль из БД и пароль, который ввел пользователь |
| authByToken |  | Проверяет JWT токен для предоставления прав доступа пользователю и ищет пользователя по токену | server\middleware\ | auth.js |
| notFound |  | Отвечает на запрос по маршруту которого не существует | errorHandler.js |

## 3.6 Реализация ВИ с учетом спроектированных классов данных и интерфейсных классов

После проектирования, необходимо убедиться в том, что используя спроектированные методы и классы удастся реализовать запланированные функции. На рисунке 24 представлена диаграмма последовательностей варианта использования «Регистрация», где используются спроектированные методы и классы клиентской и серверной части приложения. Так же в диаграмме учтен альтернативный поток, который выполняется в случае, если пользователь с введенными данными уже существует.

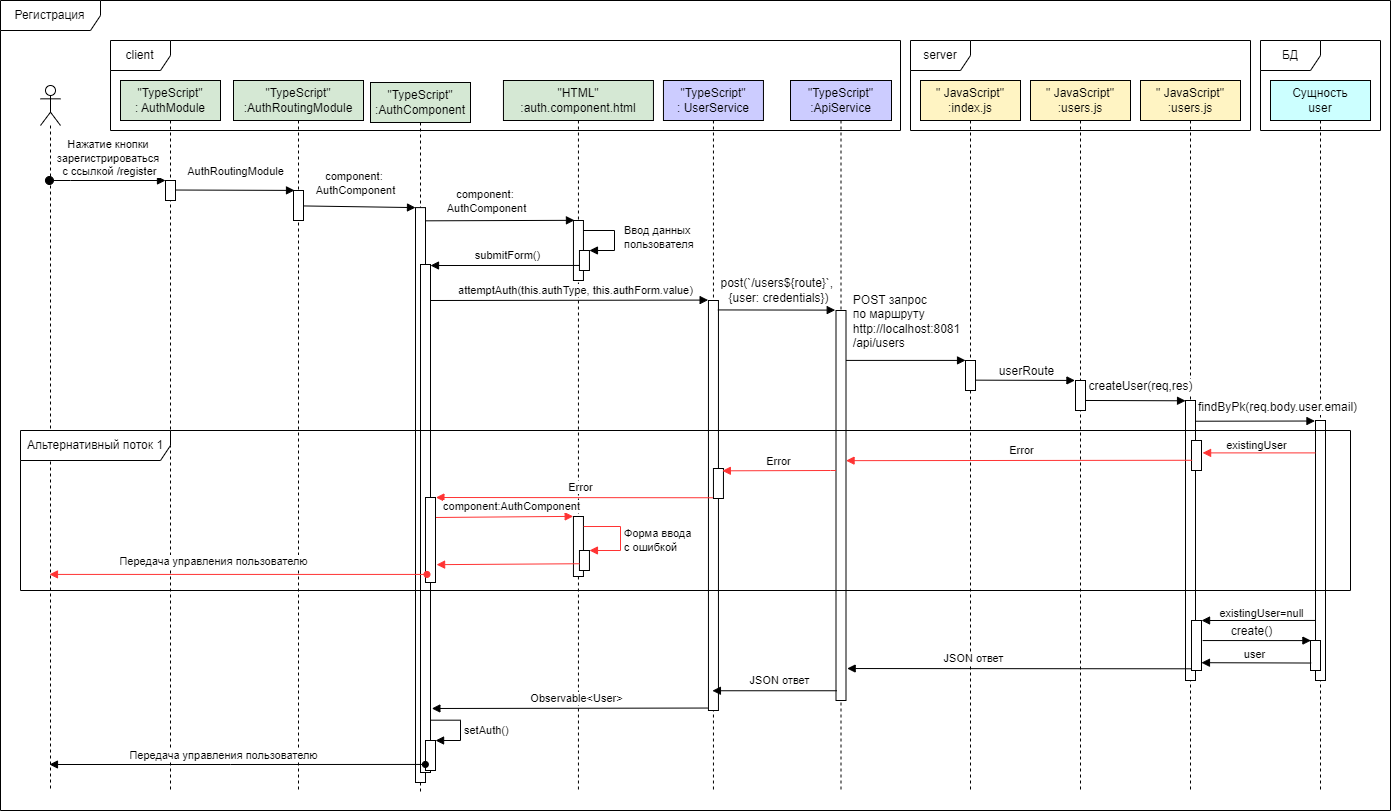


Рисунок 24 – Диаграмма последовательностей варианта использования «Регистрация»

## 3.7 Фактическая организация модулей системы

На рисунке 25 отображено разделение севрерной части программного продукта на пакеты и показано взаимодействие пакетов между собой. Пакет node\_modules является библиотечным и не изменяется, однако его необходимо установить для работы среды выполнения кода на JavaScript.

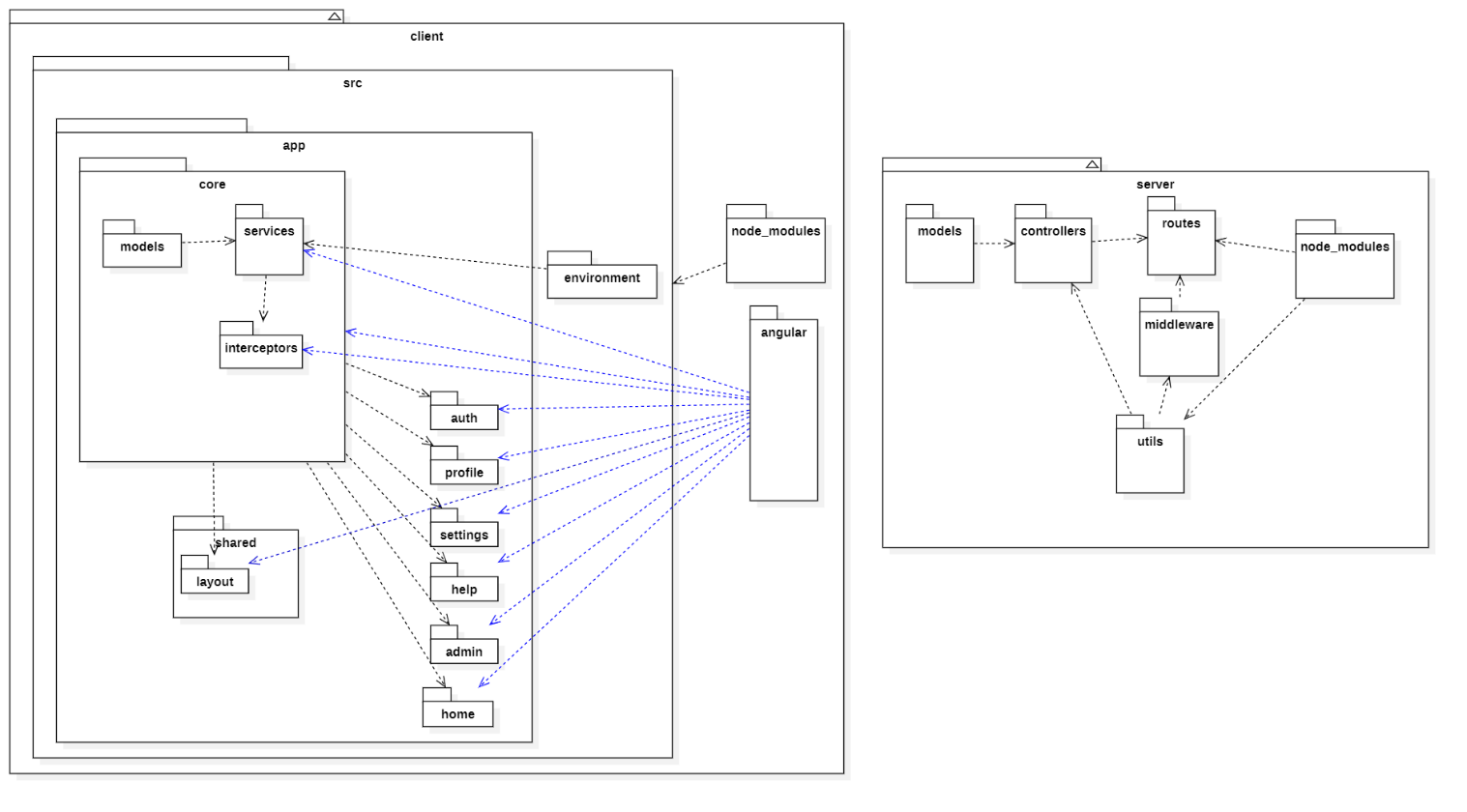


Рисунок 25 – Диаграмма пакетов серверной части

На рисунке 26 отображено разделение клиентской части программного продукта на пакеты и показано взаимодействие пакетов между собой. Пакеты angular и node\_modules являются библиотечными и не изменяются, однако их необходимо установить для работы фреймворка и среды выполнения кода на JavaScript.

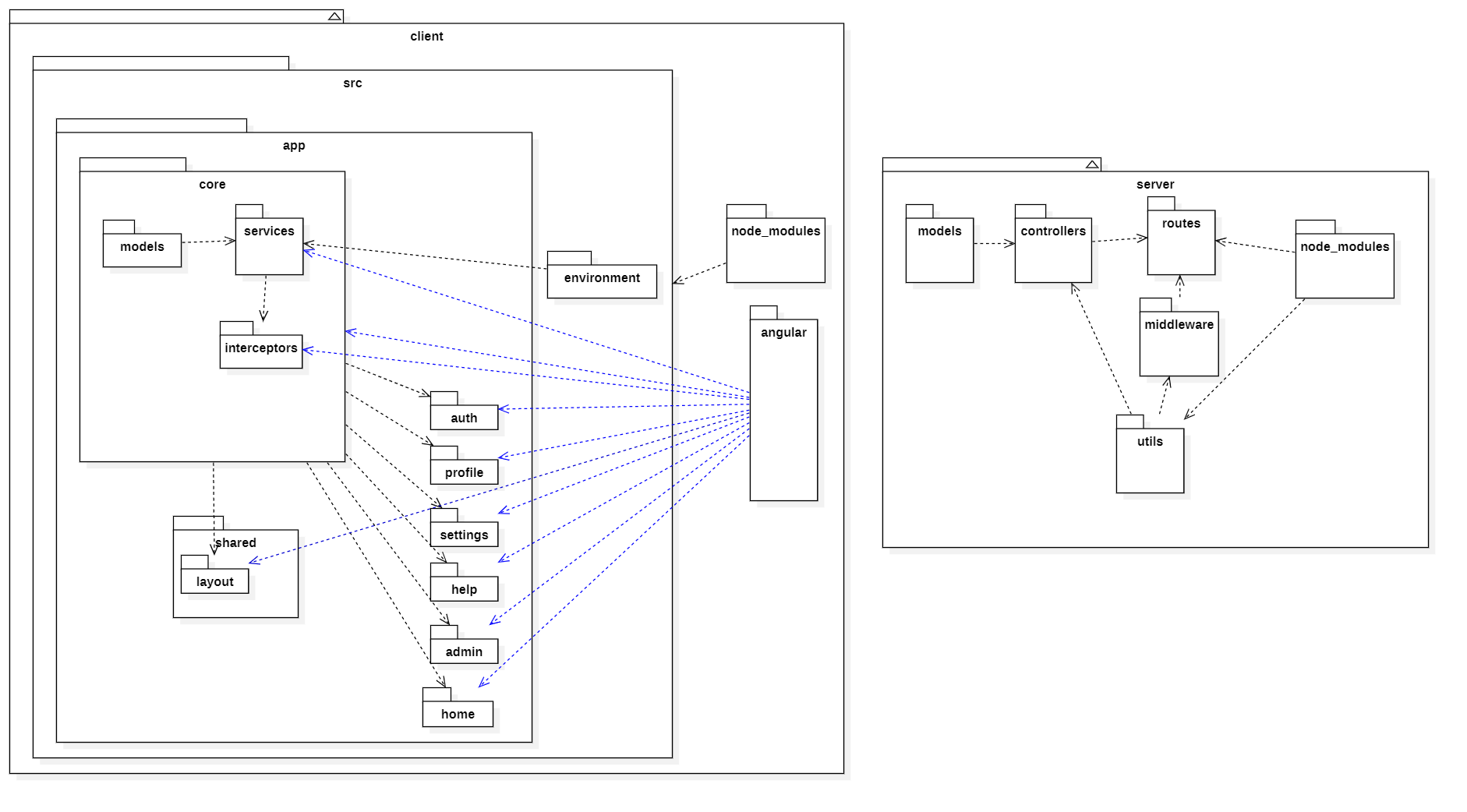


Рисунок 26 – Диаграмма пакетов клиентской части

## 3.8 Модель развертывания продукта

Для развертывания программного продукта необходимо иметь сервер, на котором необходимо развернуть 2 приложения и базу данных с СУБД. Первое приложение — это построенное скомпилированное клиентское приложение, которое уже готово к развертыванию и полноценному использованию. Второе приложение отвечает за серверную часть приложения, которое необходимо запускать.

Оба приложения могут работать только при наличии среды выполнения Node.js, однако у каждого приложения свои зависимости и библиотеки.

На стороне клиента необходим только браузер с выходом в интернет.

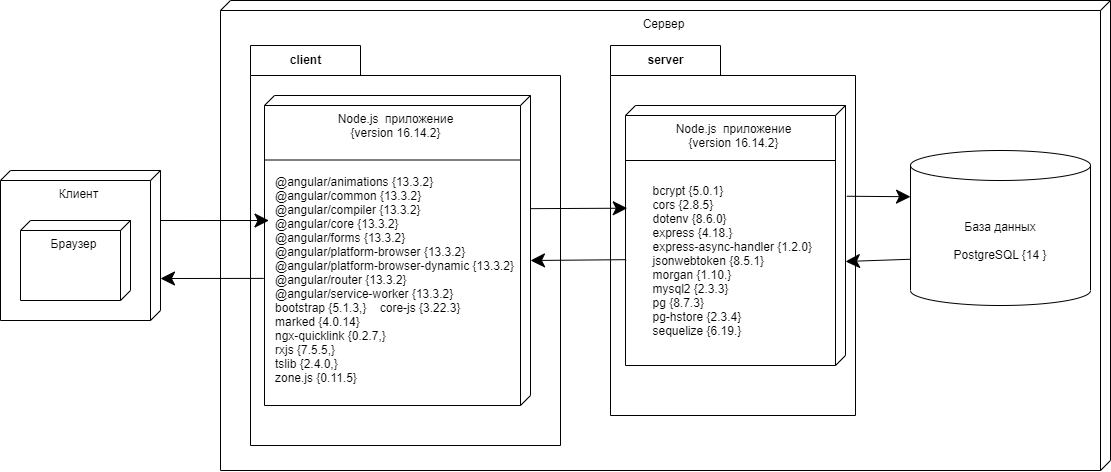


Рисунок 27 – Диаграмма развёртывания

## 3.9 Макеты пользовательских интерфейсов

Были разработаны макеты интерфейсных классов, в соответствии с диаграммой интерфейсных классов, моделями реализации ВИ и процессов.

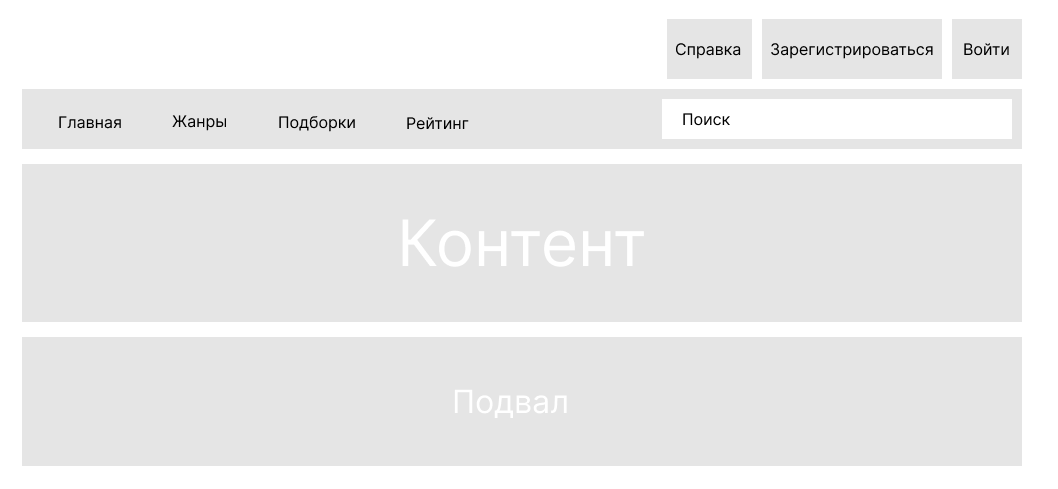


Рисунок 28 – Макет главной страницы без авторизации

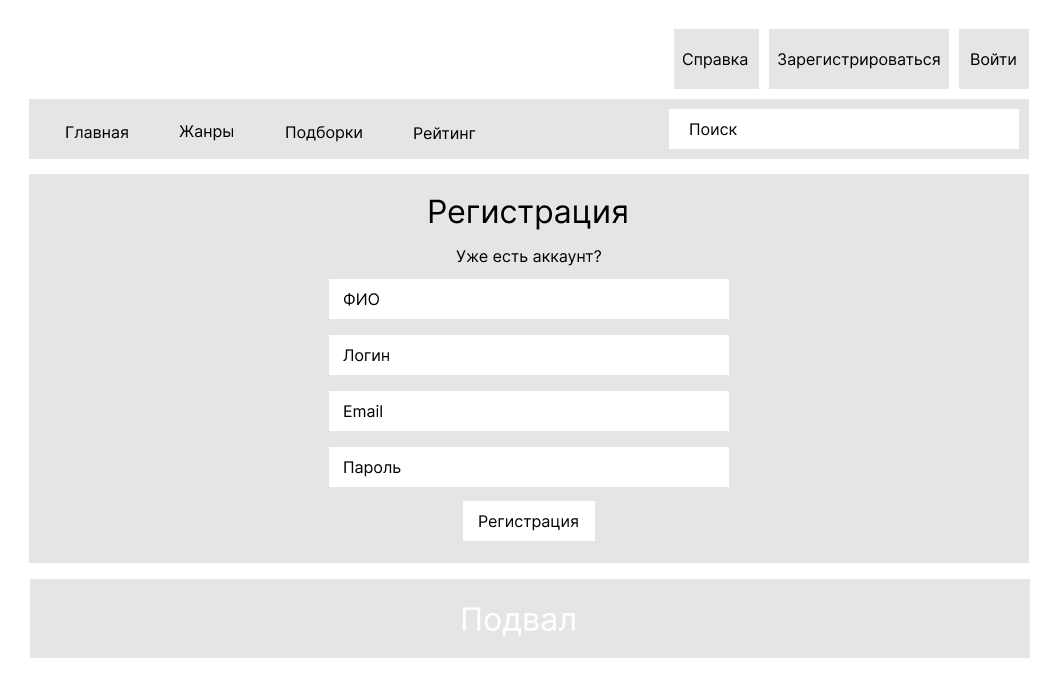


Рисунок 29 – Макет страницы «Регистрация»

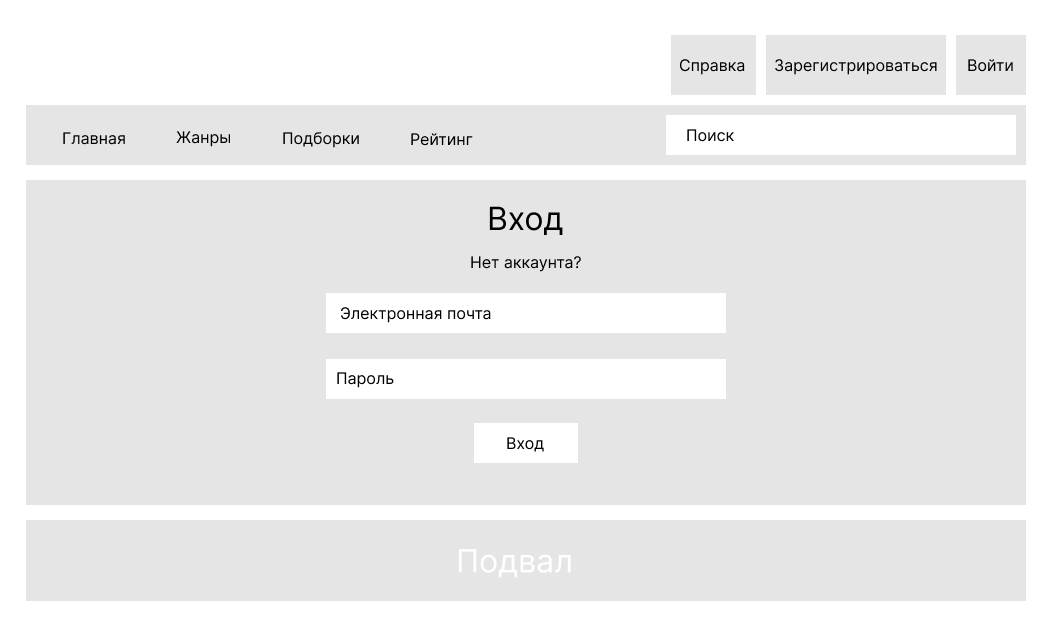


Рисунок 30 – Макет страницы «Войти в личный кабинет»

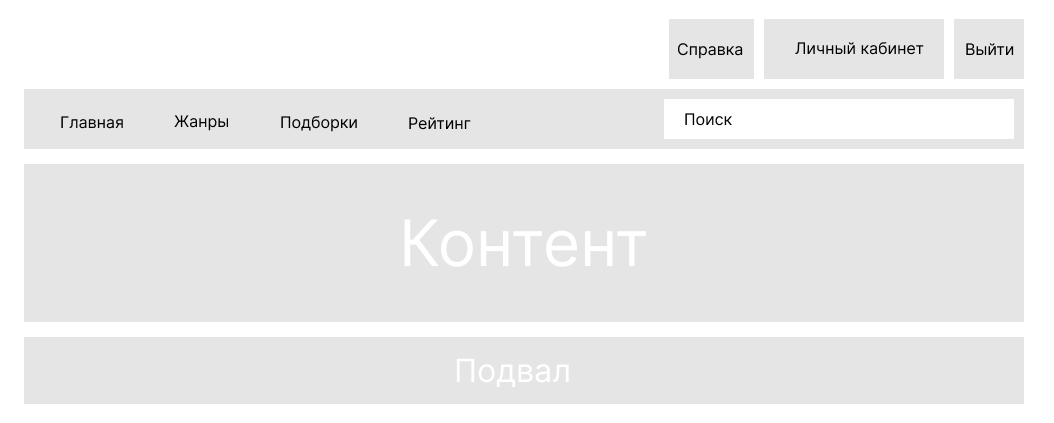


Рисунок 31 – Макет главной страницы после авторизации обычным пользователем

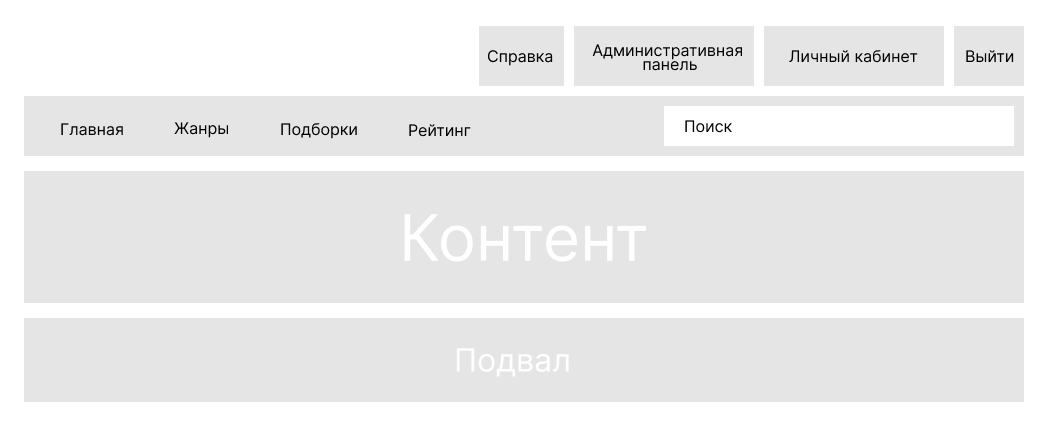


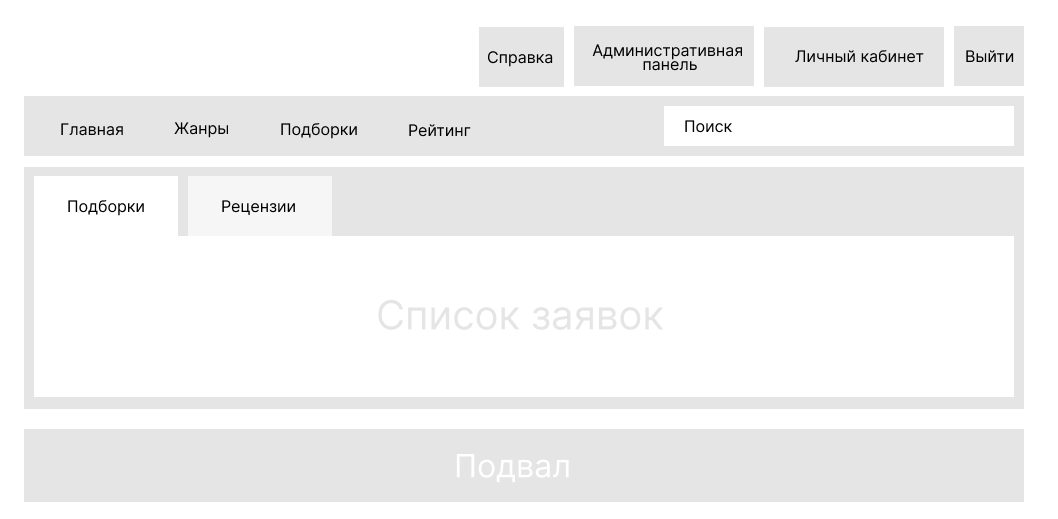
Рисунок 32 – Макет главной страницы после авторизации контент-менеджером или модератором

Рисунок 33 – Макет страницы административной панели модератора

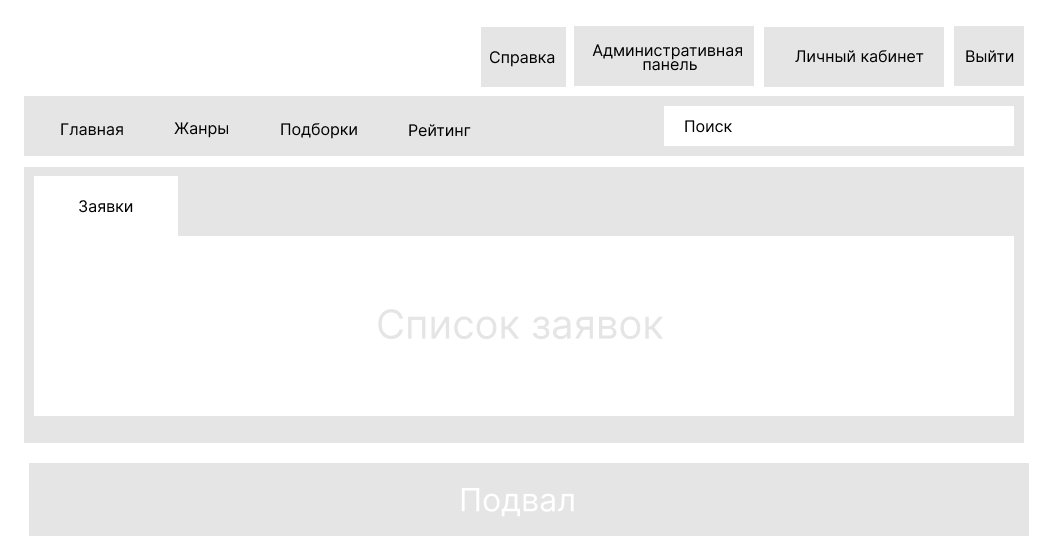


Рисунок 34 – Макет страницы административной панели контент-менеджера

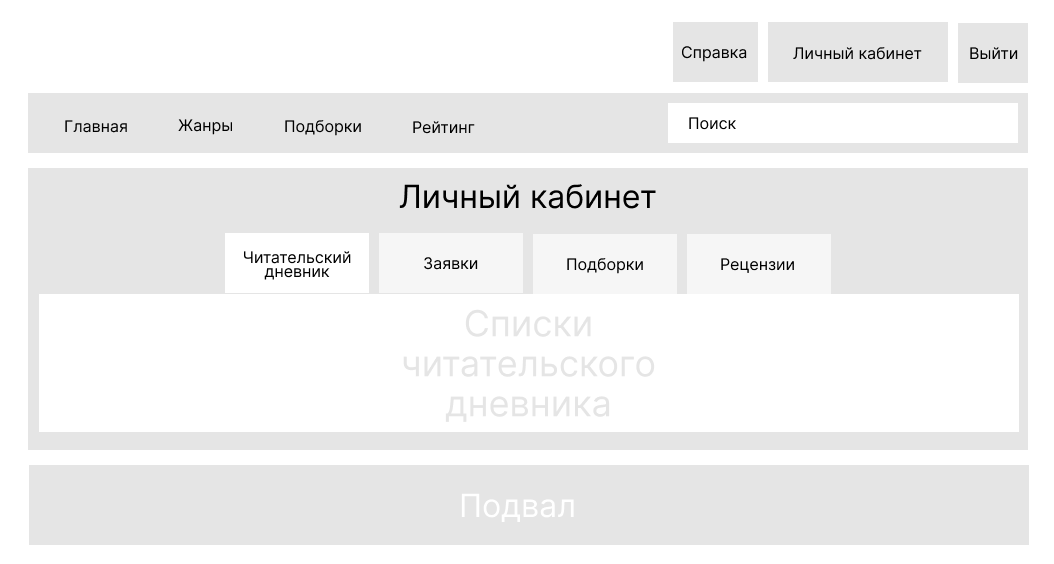


Рисунок 35 – Макет страницы личного кабинета

# 4 Программная реализация

В ходе разработки были реализованы все запланированные на этапе анализа функции: регистрация, вход в личный кабинет, выход из личного кабинета, просмотр административной панели, просмотр личного кабинета, ознакомление с документацией пользователя.

Для реализации функции регистрации в клиентской части были созданы следующие классы: AuthRoutingModule, AuthComponent, AuthModule, NoAuthGuard. Ниже представлен код файла auth.component.ts, который содержит класс AuthComponent.

import { Component, OnInit, ChangeDetectionStrategy, ChangeDetectorRef } from '@angular/core';

import { FormBuilder, FormGroup, FormControl, Validators } from '@angular/forms';

import { ActivatedRoute, Router } from '@angular/router';

import { Errors, UserService } from '../core';

@Component({

selector: 'app-auth-page',

templateUrl: './auth.component.html',

changeDetection: ChangeDetectionStrategy.OnPush

})

export class AuthComponent implements OnInit {

authType: String = '';

title: String = '';

errors: Errors = {errors: {}};

isSubmitting = false;

authForm: FormGroup;

constructor(

private route: ActivatedRoute,

private router: Router,

private userService: UserService,

private fb: FormBuilder,

private cd: ChangeDetectorRef

) {

// use FormBuilder to create a form group

this.authForm = this.fb.group({

'email': ['', Validators.required],

'password': ['', Validators.required]

});

}

ngOnInit() {

this.route.url.subscribe(data => {

// Get the last piece of the URL (it's either 'login' or 'register')

this.authType = data[data.length - 1].path;

// Set a title for the page accordingly

// add form control for username if this is the register page

if (this.authType === 'register') {

this.authForm.addControl('username', new FormControl());

this.authForm.addControl('fio', new FormControl());

}

this.cd.markForCheck();

});

}

submitForm() {

this.isSubmitting = true;

this.errors = {errors: {}};

const credentials = this.authForm.value;

this.userService

.attemptAuth(this.authType, credentials)

.subscribe(

data => this.router.navigateByUrl('/'),

err => {

this.errors = err;

this.isSubmitting = false;

this.cd.markForCheck();

}

);

}

}

Помимо классов представлений, были разработаны классы бизнес-логики: ApiService, AuthGuard, JwtService, и UserService, код которого представлен ниже.

import { Injectable } from '@angular/core';

import { Observable , BehaviorSubject , ReplaySubject } from 'rxjs';

import { ApiService } from './api.service';

import { JwtService } from './jwt.service';

import { User } from '../models';

import { map , distinctUntilChanged } from 'rxjs/operators';

@Injectable({

providedIn: 'root'

})

export class UserService {

private currentUserSubject = new BehaviorSubject<User>({} as User);

public currentUser = this.currentUserSubject.asObservable().pipe(distinctUntilChanged());

private isAuthenticatedSubject = new ReplaySubject<boolean>(1);

public isAuthenticated = this.isAuthenticatedSubject.asObservable();

constructor (

private apiService: ApiService,

private jwtService: JwtService

) {}

// Verify JWT in localstorage with server & load user's info.

// This runs once on application startup.

populate() {

// If JWT detected, attempt to get & store user's info

if (this.jwtService.getToken()) {

this.apiService.get('/user')

.subscribe(

data => this.setAuth(data.user),

err => this.purgeAuth()

);

} else {

// Remove any potential remnants of previous auth states

this.purgeAuth();

}

}

setAuth(user: User) {

// Save JWT sent from server in localstorage

this.jwtService.saveToken(user.token);

// Set current user data into observable

this.currentUserSubject.next(user);

// Set isAuthenticated to true

this.isAuthenticatedSubject.next(true);

}

purgeAuth() {

// Remove JWT from localstorage

this.jwtService.destroyToken();

// Set current user to an empty object

this.currentUserSubject.next({} as User);

// Set auth status to false

this.isAuthenticatedSubject.next(false);

}

attemptAuth(type, credentials): Observable<User> {

const route = (type === 'login') ? '/login' : '';

return this.apiService.post(`/users${route}`, {user: credentials})

.pipe(map(

data => {

this.setAuth(data.user);

return data;

}

));

}

getCurrentUser(): User {

return this.currentUserSubject.value;

}

}

А также был создан файл auth.component.html, который содержит следующий код:

<div class="auth-page">

<div class="container page">

<div class="row ">

<div class="col-md-6 offset-md-3 col-xs-12 ">

<h1 class="text-xs-center" \*ngIf="authType === 'register'">Регистрация</h1>

<h1 class="text-xs-center" \*ngIf="authType === 'login'">Вход</h1>

<p class="text-xs-center">

<a [routerLink]="['/login']" \*ngIf="authType === 'register'">Уже есть аккаунт?</a>

<a [routerLink]="['/register']" \*ngIf="authType === 'login'">Нет аккаунта?</a>

</p>

<app-list-errors [errors]="errors"></app-list-errors>

<form [formGroup]="authForm" (ngSubmit)="submitForm()">

<fieldset [disabled]="isSubmitting">

<fieldset class="form-group">

<input

formControlName="username"

placeholder="Логин"

class="form-control form-control-lg"

type="text"

\*ngIf="authType === 'register'" />

</fieldset>

<fieldset class="form-group">

<input

formControlName="fio"

placeholder="ФИО"

class="form-control form-control-lg"

type="text"

\*ngIf="authType === 'register'" />

</fieldset>

<fieldset class="form-group">

<input

formControlName="email"

placeholder="Email"

class="form-control form-control-lg"

type="text" />

</fieldset>

<fieldset class="form-group">

<input

formControlName="password"

placeholder="Пароль"

class="form-control form-control-lg"

type="password" />

</fieldset>

<button class="btn main " [disabled]="!authForm.valid" type="submit" \*ngIf="authType === 'register'">

Регистрация

</button>

<button class="btn main " [disabled]="!authForm.valid" type="submit" \*ngIf="authType === 'login'">

Вход

</button>

</fieldset>

</form>

</div>

</div>

</div>

</div>

Так же были реализованы функции в серверной части, одна из них – createUser, код которой представлен ниже:

const { databaseVersion } = require('../dbConnection');

const User = require('../models/User');

const {hashPassword,matchPassword} = require('../utils/password')

const {sign,decode} = require('../utils/jwt')

module.exports.createUser = async (req,res) => {

try{

if(!req.body.user.username) throw new Error("Требуется логин")

if(!req.body.user.fio) throw new Error("Требуется ФИО ")

if(!req.body.user.email) throw new Error("Требуется электронная почта")

if(!req.body.user.password) throw new Error("Требуется ввести пароль")

const existingUser = await User.findByPk(req.body.user.email)

if(existingUser)

throw new Error('Пользователь уже существует с этим идентификатором электронной почты')

const password = await hashPassword(req.body.user.password);

const user = await User.create({

username: req.body.user.username,

fio: req.body.user.fio,

password: password,

email: req.body.user.email,

idRole: 1,

})

if(user){

if(user.dataValues.password)

delete user.dataValues.password

user.dataValues.token = await sign(user)

res.status(201).json({user})

}

}catch (e){

res.status(422).json({errors: { body: [ 'Could not create user ', e.message ] }})

}

}

Весь исходный код выложен в репозитории онлайн-сервиса Github[2].

На рисунке 36 представлена реализованная страница регистрации.

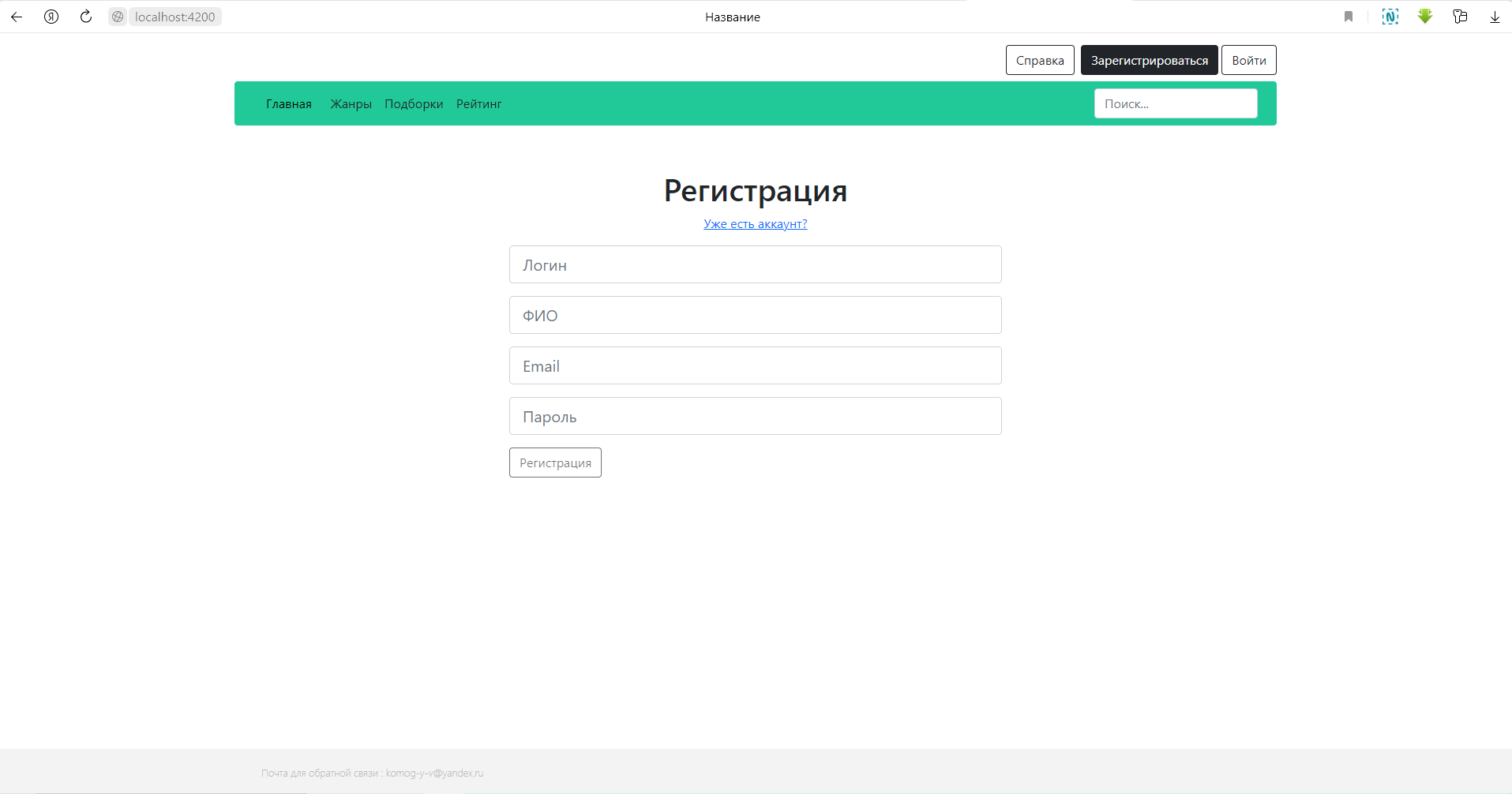


Рисунок 36 – Страница регистрации

На рисунке 37 представлена реализованная страница входа в личный кабинет

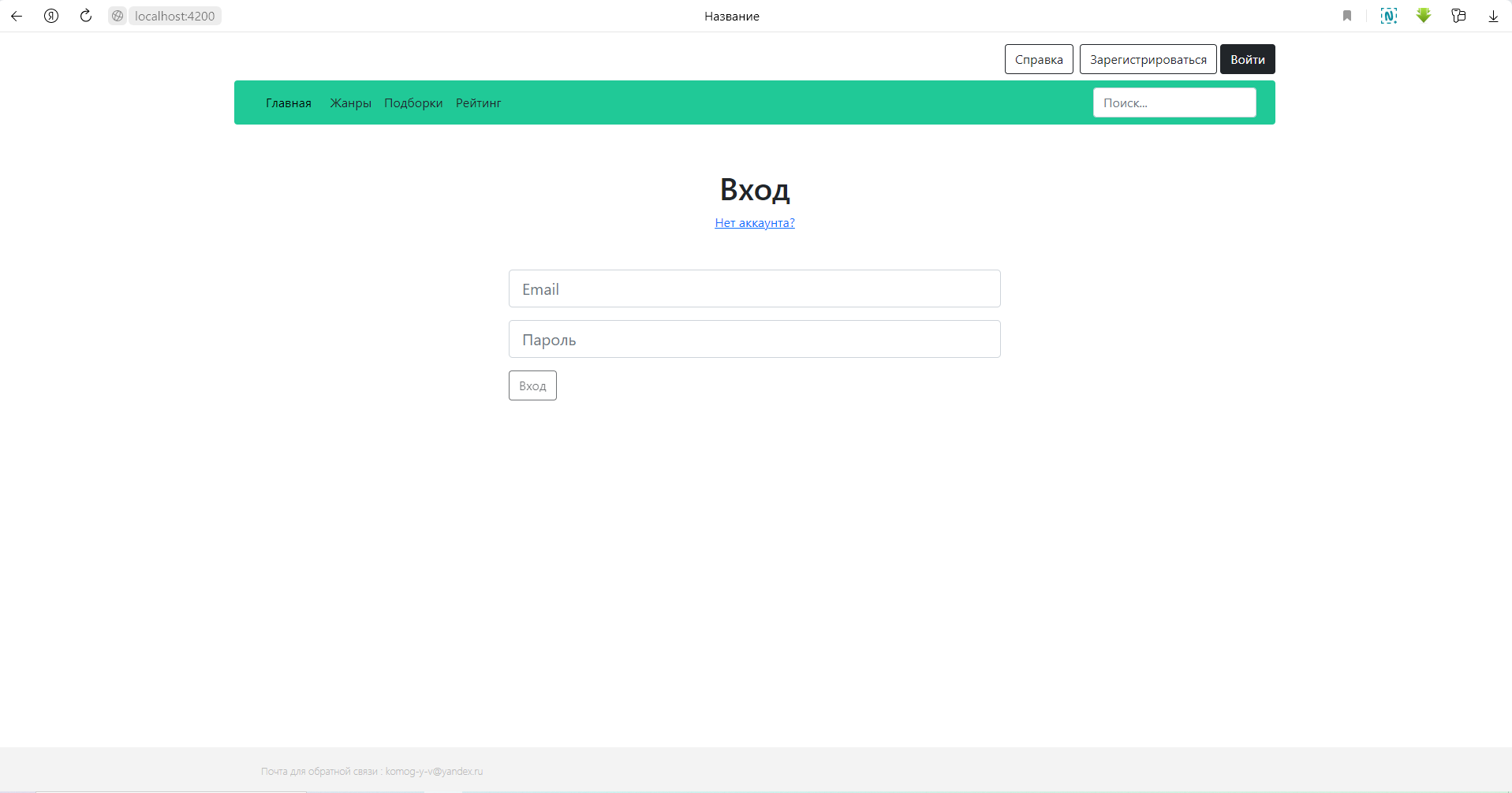


Рисунок 37 –Страница входа в личный кабинет

На рисунке 38 представлена реализованная кнопка для выхода из личного кабинета

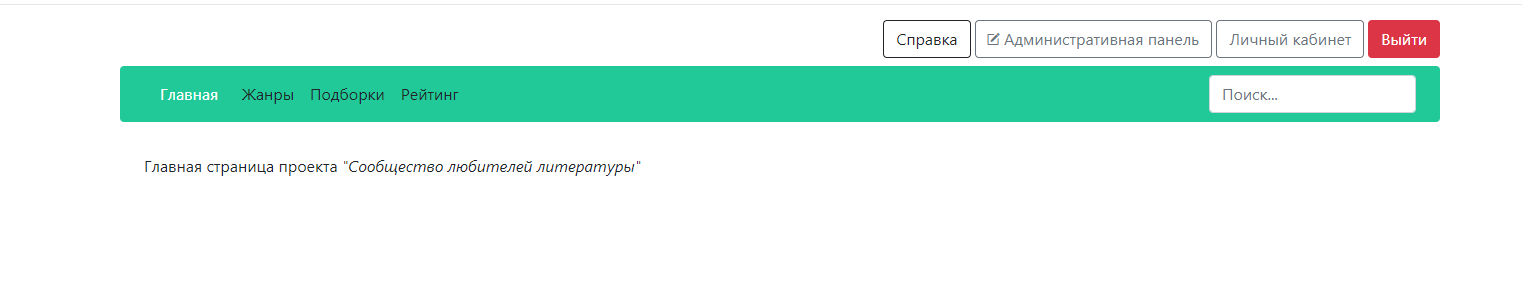


Рисунок 38 – Реализованная кнопка для выхода из личного кабинета

На рисунке 39 представлена реализованная страница просмотра административной панели.

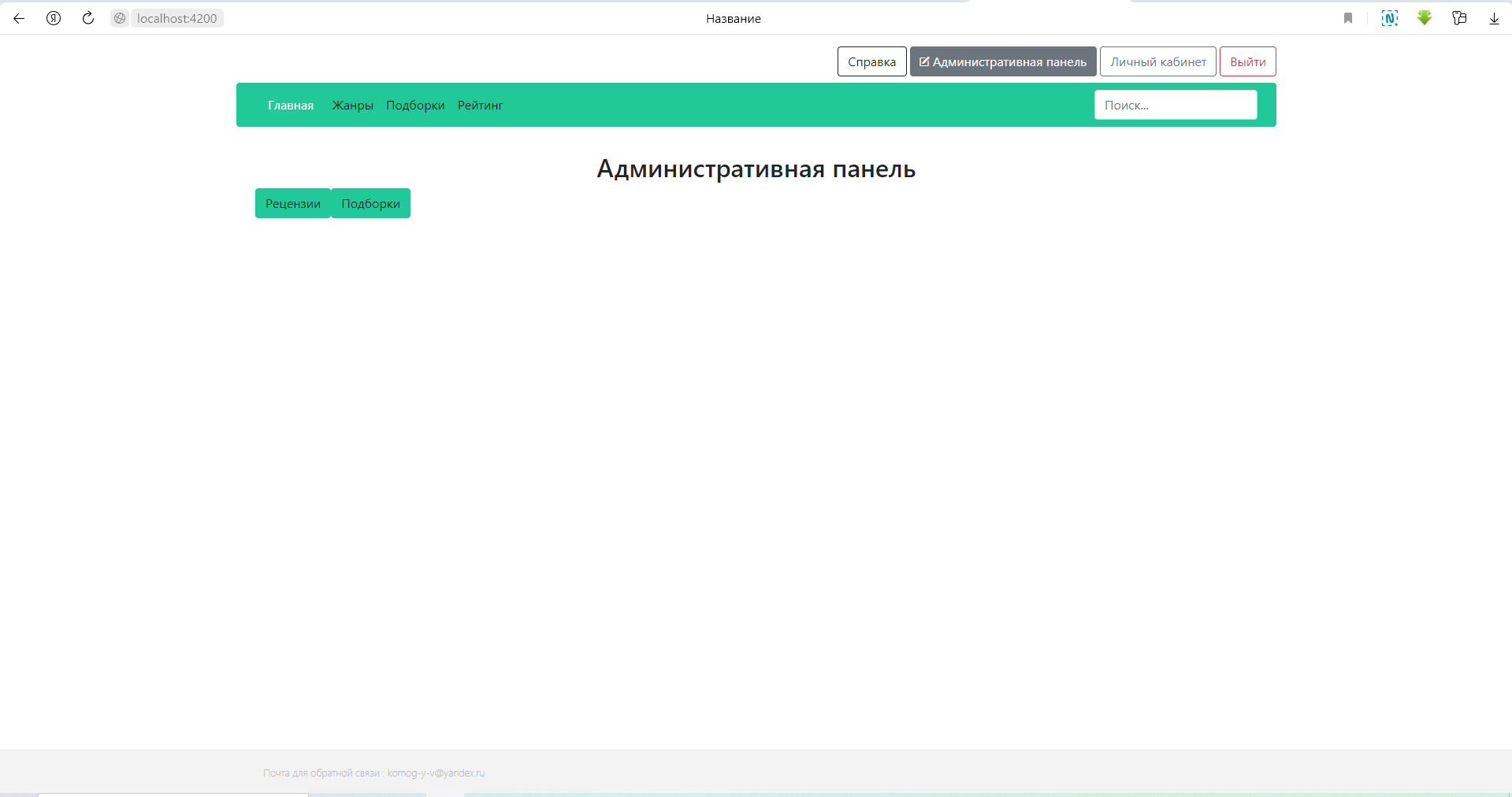


Рисунок 39 – Страница административной панели

На рисунке 40 представлена реализованная страница просмотра личного кабинета.

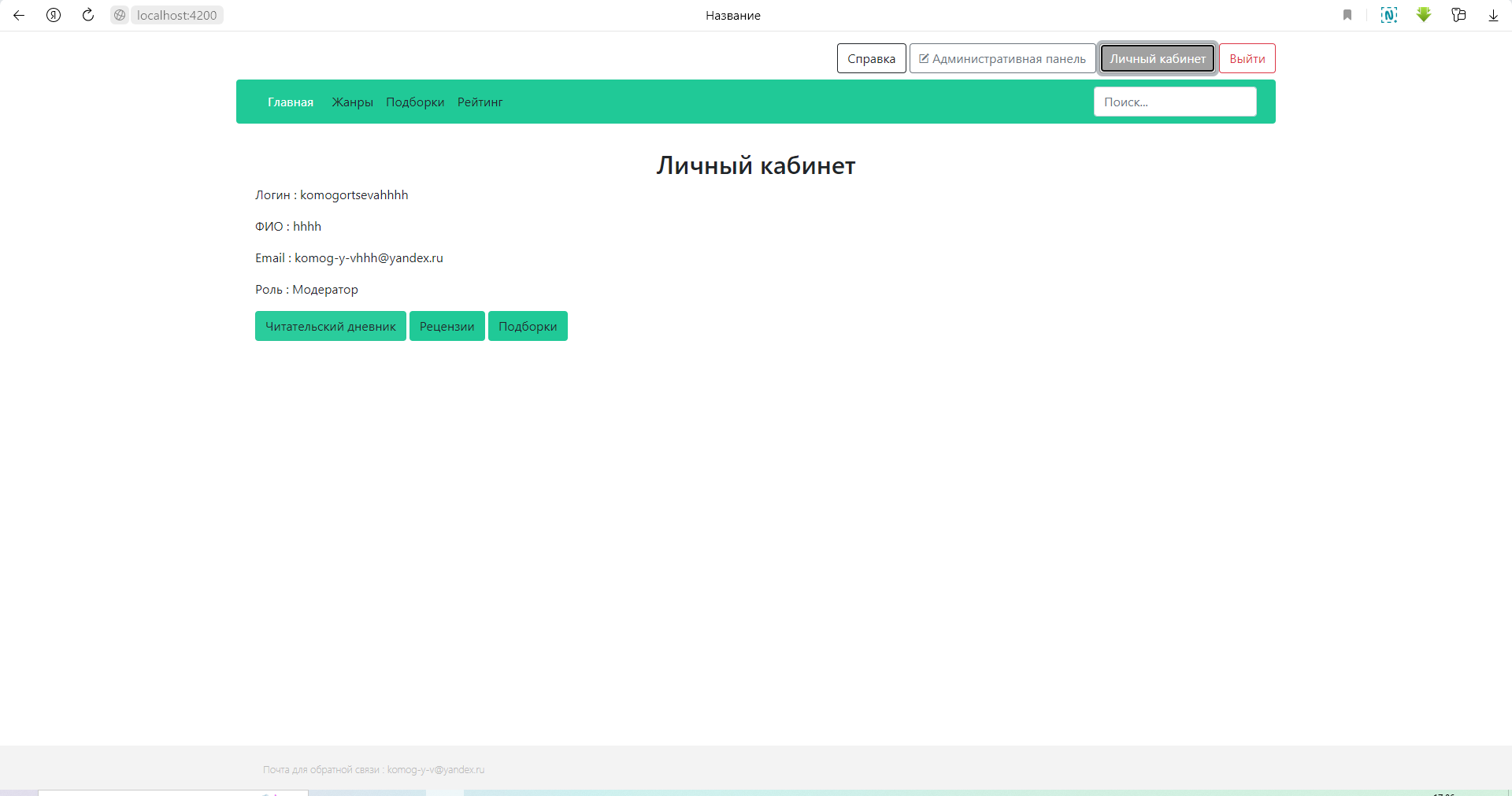


Рисунок 40 – Страница личный кабинет

На рисунке 41 представлена реализованная страница с пользовательской документацией.

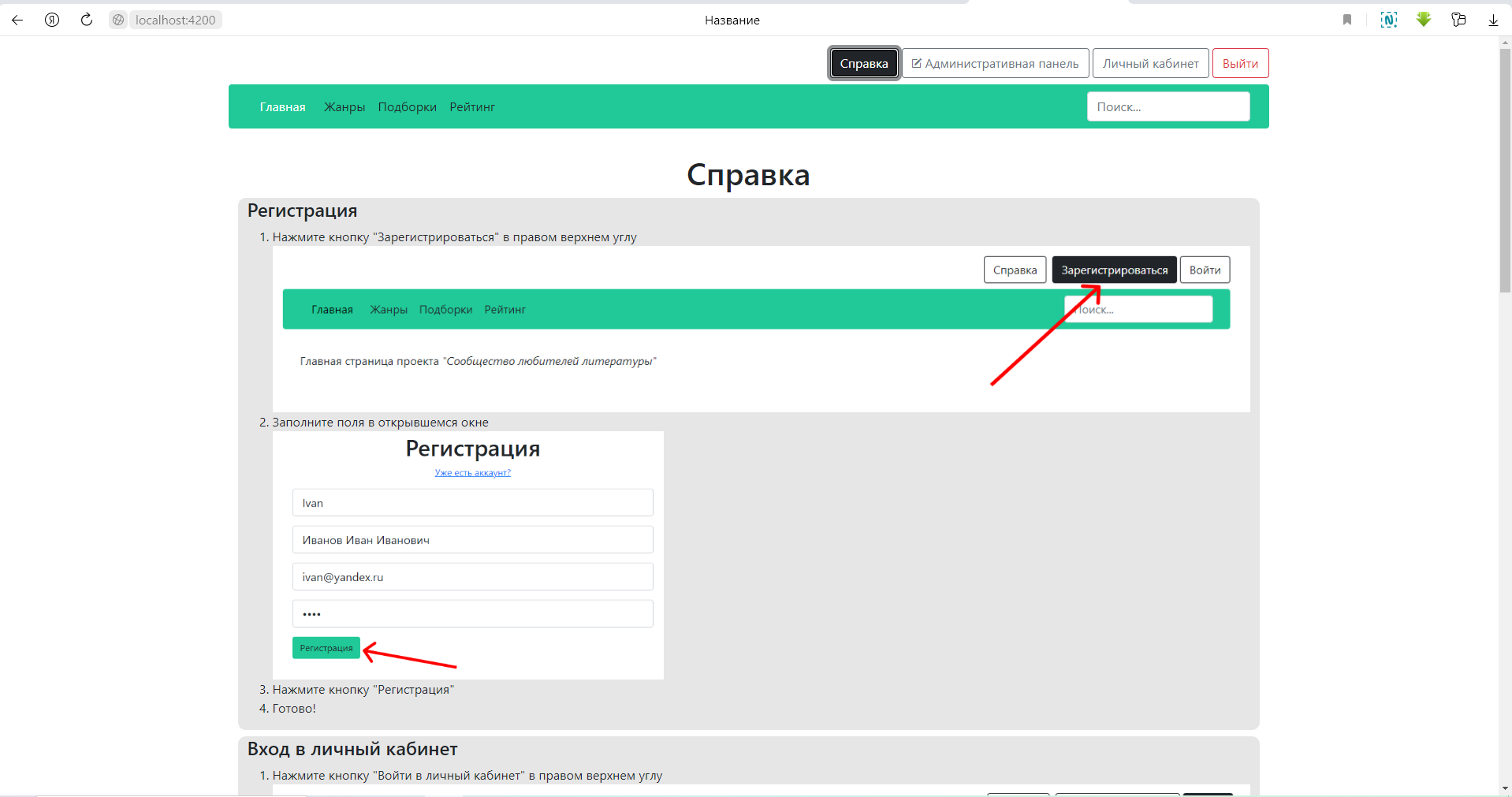


Рисунок 40 – Страница справки

# 5 Тестирование

## 5.1 Модульное тестирование

В качестве модульного тестирования была проведена проверка класса бизнес-логики ProfilesService, в том числе и метода get, который возвращает информацию для показе в личном кабинете.

import { TestBed } from "@angular/core/testing";

import { ProfilesService } from "./profiles.service";

import { HttpClientTestingModule } from "@angular/common/http/testing";

import { Profile } from "../models";

describe("ProfilesService", () => {

let service: ProfilesService;

//переменная с которой будем сравнивать

const profiles: Profile = {

username: "komogortseva",

fio: "Комогорцева Юлия Владимировна ",

email: "komog-y-v@yandex.ru",

nameRole: "Пользователь",

};

//код который надо выполнить перед выполнением теста

beforeEach(() => {

TestBed.configureTestingModule({

imports: [HttpClientTestingModule],

providers: [ProfilesService],

});

//создание обьекта ProfilesService

service = TestBed.inject(ProfilesService);

});

//тест для проверки создался ли обьект

it('Создание обьекта класса ProfilesService', () => {

expect(service).toBeTruthy();

});

//тест для проверки правильно ли работает метод обьекта,возвращает ли данные и правильные ли они

it("Получение информации для личного кабинета методом get класса ProfilesService", () => {

service.get("komogortseva").subscribe(data => {

expect(data).toEqual(profiles);

});

}

);

});

Тест успешно прошел проверку.

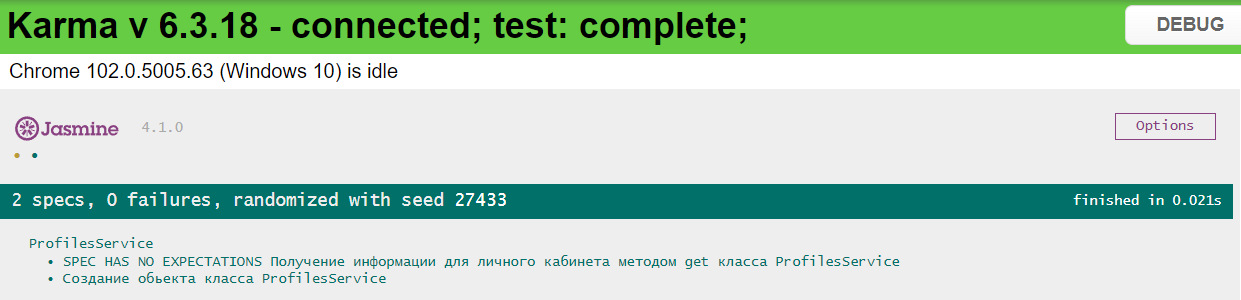


Рисунок 41 – Результаты модульного тестирования

## 5.2 Интерфейсное тестирование

В качестве интерфейсного тестирования была проведена проверка страницы с документацией пользователя.

import { ComponentFixture, TestBed } from "@angular/core/testing"

import { RouterTestingModule } from "@angular/router/testing";

import { SharedModule } from '../shared';

import { HelpComponent } from "./help.component";

describe('help component', () => {

let fixture: ComponentFixture<HelpComponent>

beforeEach(async() => {

await TestBed.configureTestingModule({

declarations: [HelpComponent],

imports: [

RouterTestingModule,

SharedModule

],

}).compileComponents();

fixture = TestBed.createComponent(HelpComponent)

fixture.detectChanges();

})

it('Создание компонента HelpComponent', () => {

expect(fixture).toBeTruthy();

})

it('Проверка отображения правильного шаблона HTML', () => {

expect(fixture.nativeElement.querySelector('h1').textContent).toContain("Справка");

})

})

Тест успешно прошел проверку.

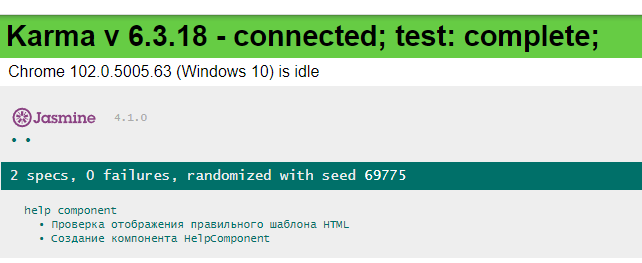


Рисунок 42 – Результаты интерфейсного тестирования

## 5.3 Функциональное тестирование

Для функционального тестирования выбрана функция входа в личный кабинет.

import { Ng2RealApp } from './app.po';

import { browser, element, by } from 'protractor';

describe('ng-demo App', () => {

let page: Ng2RealApp;

beforeEach(() => {

page = new Ng2RealApp();

});

it('Работа навигации для главной страницы', () => {

page.navigateToHome();

expect(page.getText()).toEqual('Главная страница проекта "Сообщество любителей литературы"');

});

it('Работа навигации для страницы входа', () => {

page.navigateToLogin();

expect(page.getHeadingText()).toEqual('Вход');

});

it('Работа навигации для страницы регистрации', () => {

page.navigateToRegister();

expect(page.getHeadingText()).toEqual('Регистрация');

});

it('Невозможность отправить форму без заполненных полей', () => {

page.navigateToLogin();

expect(element(by.buttonText('Вход')).isEnabled()).toEqual(false);

});

it('Функция входа в личный кабинет', () => {

page.navigateToLogin();

element(by.name('email')).sendKeys('komog-y-v@yandex.ru');

element(by.name('password')).sendKeys('1234');

element(by.buttonText('Вход')).click();

browser.sleep(5000);

browser.ignoreSynchronization = true;

expect(page.getText()).toEqual('Главная страница проекта "Сообщество любителей литературы"')

element(by.buttonText('Личный кабинет')).click();

browser.sleep(5000);

browser.ignoreSynchronization = true;

expect( element(by.css('h2')).getText()).toEqual('Личный кабинет');

});

});

Тест успешно прошел проверку.

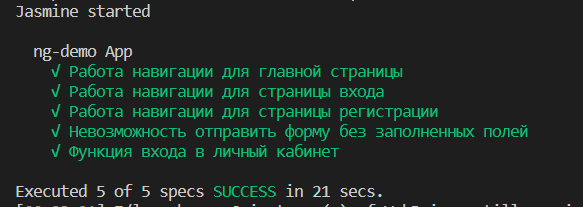


Рисунок 43 – Результаты функционального тестирования

# Заключение

В результате курсовой работы было разработано и частично реализовано веб-приложение любителей литературы. В процессе разработки были решены следующие задачи:

* + - 1. Описана предметная область
      2. Составлены требования к функциональности ПО
      3. Выполнен анализ предметной области
      4. Составлены модели системы с использованием различных нотаций
      5. Выбрана и спроектирована архитектура приложения
      6. Реализованы функции веб-приложения, запланированные на первой итерации разработки, согласно требованиям и проектированию
      7. Выполнено тестирование
      8. Составлен отчёт по всей проделанной работе

Так же, были получены и закреплены следующие навыки:

1. Формирования требований
2. Построения моделей системы в различных программных инструментах
3. Реализации клиентской части с использованием фреймворка Angular
4. Реализации серверной части с использованием фреймворка Express и ORM Sequalize
5. Работы с технической литературой

Исходный код приложения опубликован в публичном репозитории на сервисе GitHub [2].

Так же, анализируя проделанную работу, необходимо упомянуть выявленную ошибку: для удобной работы в модуле авторизация не хватает роли пользователя – администратора, которой бы назначал роли контент-менеджеру и модератору. Однако в следующей итерации разработки данную ошибку планируется исправить.

Цель курсовой работы достигнута.

# Список литературы

* 1. Курсовая работа «Разработка прикладного программного обес печения» Руководство и методические указания для студентов направления: 09.03.02 «Информационные системы и технологии». Составитель Бахвалова З.А. – Иркутск, Изд-во ИРНИТУ, 2018, 61 с.
  2. Репозиторий с исходным кодом проекта // GitHub URL: https://github.com/KomogortsevaYulia/CourseWork\_CommunityLiteratureLovers\_TRPK
  3. Эрик Фримен, Элизабет Робсон Head First Паттерны проектирования обновленное юбилейное издание, 2020. 656 с. ISBN: 978-5-4461-1034-6 Серия: Head First O’Reilly
  4. Руководство по Node.js. [Электронный ресурс] URL: https://metanit.com/web/nodejs/(дата обращения)
  5. Руководство по Angular. [Электронный ресурс] URL: https://metanit.com/web/angular2/
  6. StarUML documentation. [Электронный ресурс] URL: https://docs.staruml.io/
  7. Node.js v18.2.0 documentation. [Электронный ресурс] URL: https://nodejs.org/api/console.html
  8. Introduction to the Angular Docs. [Электронный ресурс] URL: https://angular.io/docs#introduction-to-the-angular-docs
  9. Manual | Sequalize. [Электронный ресурс] URL: https://doc.esdoc.org/github.com/sequelize/sequelize/
  10. Introduction Bootstrap. [Электронный ресурс] URL: https://getbootstrap.com/docs/5.1/getting-started/introduction/
  11. CodebaseShow – RealWorld Example Apps. [Электронный ресурс] URL: https://codebase.show/projects/realworld?category=frontend