

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление 09.03.01 «Информатика и вычислительная техника»
подготовки:

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

СЕРВИС ЭЛЕКТРОННОГО
БРОНИРОВАНИЯ КНИЖНЫХ ИЗДАНИЙ

Студент

А. Китаев 13.06.24 Китаев Андрей Андреевич
(подпись, дата)

Руководитель

А. Токарев 13.06.24 Токарев Андрей Николаевич
(подпись, дата)

Нормоконтролёр

А. Бычков 13.06.24 Бычков Андрей Станиславович
(подпись, дата)

Работа допущена к защите (протокол заседания. кафедры от 27.05.2024г. № 11)

Заведующий кафедрой

М.А. Митрохин
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

17.06.2024 № 28)

Секретарь ГЭК

Е. Гурин
(подпись)

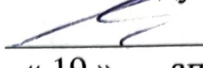
Гурин Е.И.
(фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
« 19 » апреля 2024 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Китаев Андрей Андреевич гр.20ВВП2 факультета вычислительной техники

2. Тема работы: Сервис электронного бронирования книжных изданий

Тема утверждена приказом ПГУ № 405/о от 19 апреля 2024 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Токарев А.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать сервис электронного бронирования книжных изданий. Приложение должно обладать следующими функциями:

позволять библиотекам добавлять, редактировать, удалять информацию о печатных изданиях,

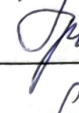
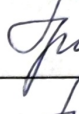
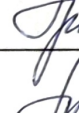
позволять пользователям искать книжные издания, получать информацию о доступности печатных изданий в библиотеках, бронировать книги, получать номер бронирования (заказа),

предоставлять сотрудникам библиотек возможность обработки заказа бронирования книг.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги сервиса, изучить способы проектирования и разработки приложений и виды архитектурных паттернов. Также необходимо изучить основные языки программирования для разработки приложений.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области методов обнаружения объектов на изображении	32	19.04.2024- 22.04.2024	
2	Анализ предметной области методов измерения расстояния	32	23.04.2024- 27.04.2024	
3	Анализ инструментов разработки	65	28.04.2024- 02.05.2024	
4	Разработка программного обеспечения на основе проведенного анализа	94	03.05.2024- 23.05.2024	
5	Отладка и тестирование	49	24.05.2024- 27.05.2024	
6	Оформление выпускной работы	32	28.05.2024- 30.05.2024	

Дата выдачи задания " 19 " апреля 2024 г.

Руководитель бакалаврской работы  А.Н. Токарев
(подпись, дата)

Задание к исполнению принял студент  А.А. Китаев
(подпись, дата)

Работу к защите допустить
Декан факультета  Л.Р. Фионова
17.06.2024 (подпись, дата)

Реферат

Работа содержит 125 л., 36 рисунков, 9 таблиц, 18 источников, 4 приложения

JAVASCRIPT, VUE, NUXT, POSTGRESQL, WEB-ПРИЛОЖЕНИЕ, ORM PRISMA

Цель работы – разработать сервис электронного бронирования книжных изданий.

Сервис должен позволять библиотекам добавлять имеющиеся у них книги в базу данных, позволять пользователям искать книжные издания, видеть информацию о доступности книжных изданий в библиотеках, бронировать книжные издания, получать номер бронирования (заказа), а также должно предоставлять сотрудникам библиотек возможности для обработки забронированных книг.

Результатом проделанной работы является проект, удовлетворяющий поставленной задаче, а именно, спроектировано, разработано и протестировано приложения для бронирования книжных изданий в библиотеках. При разработке было использовано множество технологий, в числе которых язык программирования JavaScript, фреймворк Vue 3, Nuxt 3, CSS-фреймворк Tailwind, библиотека для управления состоянием Pinia, СУБД PostgreSQL и ORM Prisma, а также многочисленные библиотеки для Vue.js

ПГУ 1.090301.29.001 ПЗ

					ПГУ 1.090301.29.001 ПЗ						
Изм	Лист	№ докум.	Подпись	Дата	Сервис электронного бронирования книжных изданий Пояснительная записка			Литера	Лист	Листов	
Разраб.	Китаев А.А.	А. Китаев	13.06.24							2	125
Пров.	Токарев А.Н.	А. Токарев	13.06.24								
Н. Контр.	Бычков А.С.	А. Бычков	13.06.24								
Утв.	Митрохин М.А.	М. Митрохин	13.06.24					ФВТ, каф. ВТ гр. 20ВВП2			

Перечень сокращений и обозначений

БД – база данных

ВЦИОМ – всероссийский центр изучения общественного мнения

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

СУБД – система управления базой данных

API (Application programming interface) – программный интерфейс приложения

HTTP (Hypertext Transfer Protocol) – протокол передачи гипертекста

JSON (Java Script Object Notation) – текстовый формат для обмена структурированными данными и для хранения их.

MVVM (англ. Model-View-ViewModel) — паттерн проектирования архитектуры

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»

SQL (Structured Query Language) - язык структурированных запросов

SFC (Single File Component) – В контексте Vue однофайловый компонент, инкапсулирующий содержимое и логику в каждый компонент

SPA (Single Page Application) – Одностраничное приложение

UML (Unified Modeling Language) – унифицированный язык моделирования

Содержание

1	Анализ предметной области	9
1.1	Анализ существующих аналогов.....	11
1.1.1	Сервис «Библиотеки Москвы»	11
1.1.2	Сервис «Общедоступные библиотеки Санкт-Петербурга».....	15
1.2	Постановка задачи	18
1.3	Функционал приложения	19
2	Выбор технологий и средств разработки приложения	23
2.1	Выбор языка программирования.....	23
2.1.1	Python	23
2.1.2	C#	24
2.1.3	JavaScript	24
2.2	Выбор фреймворка.....	25
2.2.1	React.....	25
2.2.2	Vue.js	26
2.2.3	Nuxt.js	26
2.3	Выбор СУБД.....	27
2.3.1	MySQL.....	27
2.3.2	PostgreSQL	28
2.4	Выбор среды разработки	29
2.4.1	WebStorm	29
2.4.2	IntelliJ IDEA	29
2.4.3	Visual Studio Code	30
2.5	Выбор паттерна проектирования архитектуры.....	31
2.5.1	Архитектурный паттерн MVC	31
2.5.2	Архитектурный паттерн MVP	32
2.5.3	Архитектурный паттерн MVVM	32
3	Реализация приложения	35
3.1	Описание работы приложения.....	35
3.2	Структура базы данных	36

3.3 Описание компонентов.....	42
3.4 Основные алгоритмы взаимодействия клиента и сервера.....	52
4 Тестирование	59
4.1 Регистрация пользователя.....	59
4.2 Авторизация пользователя.....	61
4.3 Поиск книг, просмотр страницы книги, бронирование	62
4.4 Тестирование функционала администратора.....	63
5 Руководство пользователя.....	66
5.1 Результат работы приложения.....	66
6 Безопасность жизнедеятельности.....	73
Заключение	76
Список использованных источников	77
Приложение А	79
Приложение А.1 – Диаграмма последовательностей.....	80
Приложение А.2 – Диаграмма развертывания.....	81
Приложение А.3 – Диаграмма компонентов.....	82
Приложение А.4 – Диаграмма вариантов использования	83
Приложение Б – Листинг программы	84
Приложение В – Презентация	121

Введение

В современном мире, в котором поток информации непрерывно растет, роль традиционных библиотек в обеспечении доступа к знаниям остается огромной, несмотря на появление электронных книг, аудиокниг и других видов цифровых изданий. Библиотеки на протяжении тысяч лет являются центрами распространения знаний и образования, обеспечивая доступ к широкому спектру информационных ресурсов, включая книги, журналы, научные статьи другие материалы.

Однако, с развитием технологий и переходом к цифровой среде, традиционные библиотеки должны адаптироваться к изменяющимся потребностям и предпочтениям пользователей. В этом контексте разработка и внедрение сервиса электронного бронирования книг становится не только логичным шагом, но и необходимостью для современных библиотек.

Сервис электронного бронирования книжных изданий дополняет традиционные функции библиотек, предоставляя пользователям возможность находить и резервировать книги удаленно, через интернет. Это значительно увеличивает удобство пользования библиотечными ресурсами, позволяя легко отслеживать доступность книг в различных библиотеках.

Таким образом, внедрение сервиса электронного бронирования книг в работу библиотек позволяет не только расширить доступ пользователей к книжным ресурсам, но и сделать этот процесс более удобным и эффективным. Адаптация библиотечных сервисов к современным технологическим требованиям способствует укреплению роли библиотек в обществе как центров знаний и культуры, продвигая принципы доступности информации для всех слоев населения.

В связи с вышесказанных, реализация сервиса электронного бронирования книг является актуальной темой для дипломного проекта.

					ПГУ 1.090301.29.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

1 Анализ предметной области

Перед разработкой сервиса электронного бронирования книжных изданий необходимо провести анализ предметной области, чтобы понять основные потребности пользователей и особенности функционирования библиотечной системы.

В первую очередь, следует рассмотреть существующие проблемы и ограничения традиционной системы бронирования книг в библиотеках.

В настоящий момент библиотечное дело в России имеет целый ряд проблем. Эти проблемы определены в Стратегии развития библиотечного дела в РФ, утверждённой распоряжением Правительства РФ 13 марта 2021 года. Согласно этому документу, имеются неудовлетворительная материально-техническая база, недостаточный уровень информатизации библиотек, сокращение количества и времени работы библиотек [1]. При этом, определены направления развития библиотечного дела, одним из таких направлений является «Развитие информационных технологий и цифровая трансформация деятельности библиотек».

Согласно Стратегии развития библиотечного дела, следствием цифровой трансформации библиотек должны стать [2]:

- перевод основных процессов, обеспечивающих жизнедеятельность библиотек, на цифровые технологии;
- формирование цифровой среды библиотеки, ориентированной на потребности разных групп населения;
- выравнивание ситуации по активности и уровню цифровизации библиотек по сравнению с другими учреждениями культуры и науки;
- формирование у IT-сообщества отношения к библиотеке как к равноправному и перспективному партнеру, встраивание библиотек в цифровую среду.
- наряду с традиционными печатными изданиями должно осуществляться пополнение библиотечных фондов электронными документами, российскими и

зарубежными удаленными информационными ресурсами (базами данных, электронными библиотеками и др.).

Согласно данным Минкультуры РФ, очная посещаемость библиотек остаётся стабильной, на уровне 420 миллионов посещений в год, и всё ещё выше, чем посещаемость онлайн-библиотек, услугами которых воспользовались 201 млн раз за 2022 год [3]. При этом, посещаемость онлайн-библиотек неуклонно растёт [4]. Сервис электронного бронирования книжных изданий объединяет в себе удалённое и очное пользование библиотечными ресурсами, значит, вследствие роста популярности удалённых библиотечных услуг, может повысить и очную посещаемость библиотек.

Согласно Стратегии развития библиотечного дела в РФ, в настоящее время наблюдаются существенное отставание библиотек страны в области развития информационно-коммуникативных и внедрения современных цифровых технологий и, как следствие, недостаточно качественный уровень информационно-библиотечного обслуживания пользователей [5].

Недостаточно качественный уровень информационно-библиотечного обслуживания выражается, в том числе, в ограниченной доступности информации о наличии книг в библиотеках. Это может привести к необходимости физически посещать каждую библиотеку для поиска нужного издания, что занимает время и ресурсы пользователя. Вследствие этого, к библиотекам снижается интерес молодого поколения, привыкшего к цифровым сервисам и онлайн взаимодействию.

Актуальность темы данной работы обусловлена, прежде всего, отсутствием аналогичных сервисов во многих городах России, в частности, в городе Пенза. Существуют различные электронные каталоги Пензенских библиотек, сводные каталоги библиотек России, библиографические базы данных, но они представляют мало пользы для простого пользователя, так как не предоставляют удобного пользовательского интерфейса и, обычно, не содержат данных о местонахождении книг и их доступности, а также, не предлагают возможности удалённого бронирования книжных изданий.

					ПГУ 1.090301.29.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

Таким образом, разработка сервиса электронного бронирования книжных изданий актуальна в нашем стремительно развивающемся обществе.

1.1 Анализ существующих аналогов

Проблема малой интегрированности традиционных библиотек в цифровую среду не нова, поэтому существуют сервисы для электронного бронирования книг в библиотеках. Эти сервисы необходимо рассмотреть, чтобы разработать продукт, учитывающий существующие наработки в этой области, сохраняющий достоинства и лишённый недостатков своих аналогов.

Среди аналогов можно выделить:

- «Библиотеки Москвы»
- «Общедоступные библиотеки Санкт-Петербурга»

1.1.1 Сервис «Библиотеки Москвы»

На рисунке 1.1 отображена главная страница сервиса «Библиотеки Москвы»

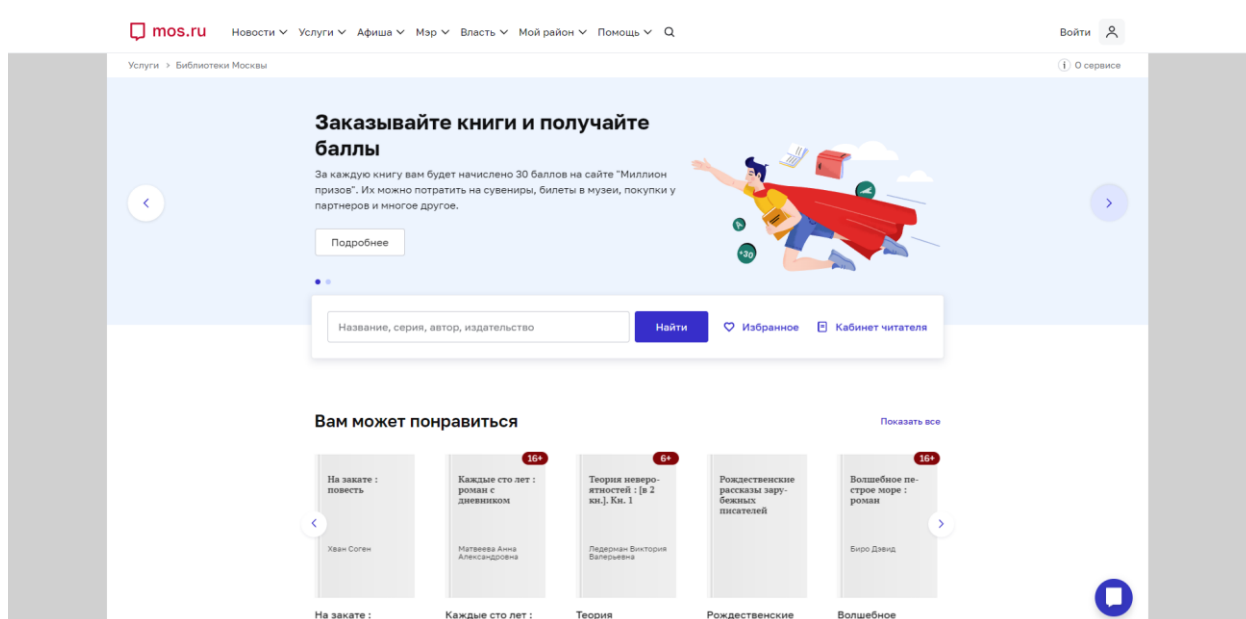


Рисунок 1.1 – Главная страница сервиса «Библиотеки Москвы»

На рисунке 1.2 представлен поиск в сервисе «Библиотеки Москвы»

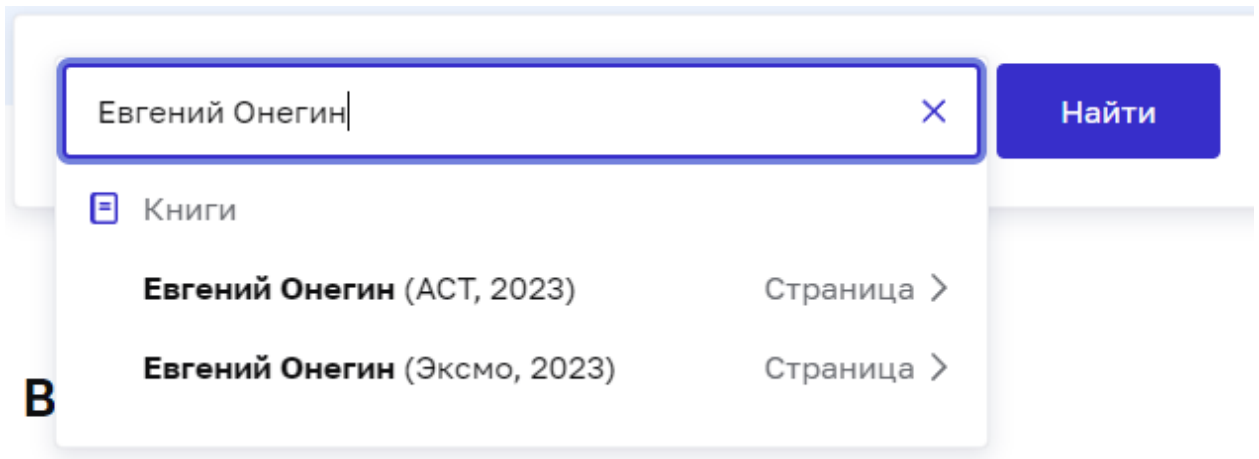


Рисунок 1.2 – Поиск книги

Результат поиска представлен на рисунке 1.3

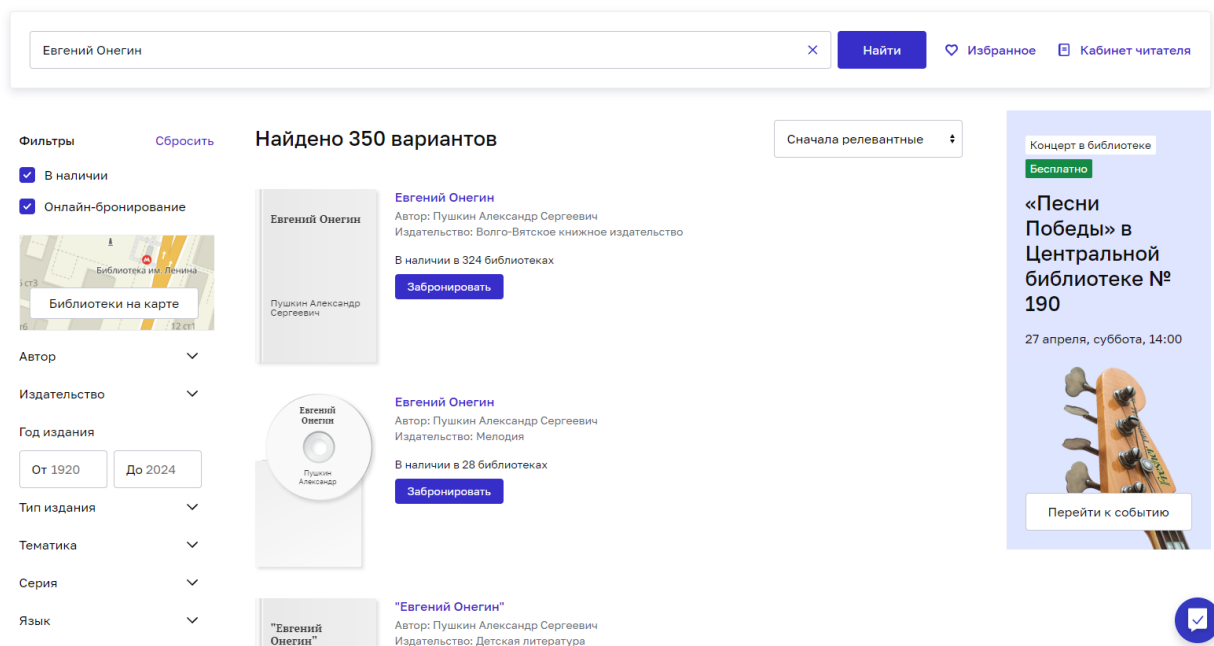


Рисунок 1.3 – Результат поиска в «Библиотеки Москвы»

Страница книги представлена на рисунках 1.4 и 1.5

Оформление заказа

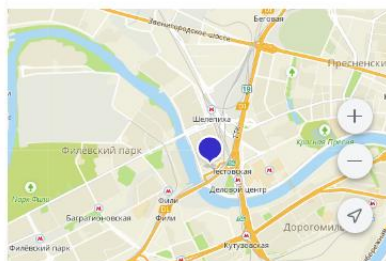
Детали

Евгений Онегин : роман в стихах
 Автор: Пушкин Александр Сергеевич
 Издательство: Детская и юношеская книга
 Возрастное ограничение: 12+

[Раскрыть](#) ▼

Библиотека №11 имени Д.В. Давыдова
 Мукомольный проезд, дом 9, корпус 2

[Раскрыть](#) ▼



Получатель заказа

Введите номер читательского билета
 13 цифр

[Нет читательского билета? Оформить билет онлайн](#)

[Инструкция по оформлению читательского билета](#)

[Забронировать](#)

[Отменить заказ](#)



Рисунок 1.6 – Страница оформления заказа

Достоинства сервиса «Библиотеки Москвы»

- удобный интерфейс, похожий на интерфейс интернет-магазинов, современный дизайн;
- удобный поиск с подсказками, позволяющими осуществлять поиск более точно (выбрать издание, издательство и т.д.);
- наличие фильтров поиска по автору, издательству, типу издания, языку, тематике, серии, возрастным ограничениям;
- интуитивно понятное отображение наличия книги в библиотеках, в виде списка адресов, и в виде точек на карте;
- возможность оформить читательский билет в режиме онлайн

Недостатки сервиса «Библиотеки Москвы»;

- отсутствие графических изображений обложек книг;

– недостаток информации о книгах: отсутствует информация о количестве страниц, номер ISBN, содержание книги у сборников произведений.

К недостаткам также можно отнести географическую ограниченность сервиса городом Москва, но это обусловлено самой идеей «экосистемы» государственных услуг для жителей Москвы: сервис является лишь малой частью предоставляемых сайтом услуг.

Части интерфейса и основные функциональные возможности сервиса «Общедоступные библиотеки Санкт-Петербурга» представлены на рисунках 1.7, 1.8, 1.9, 1.10, 1.11, 1.12.

1.1.2 Сервис «Общедоступные библиотеки Санкт-Петербурга»

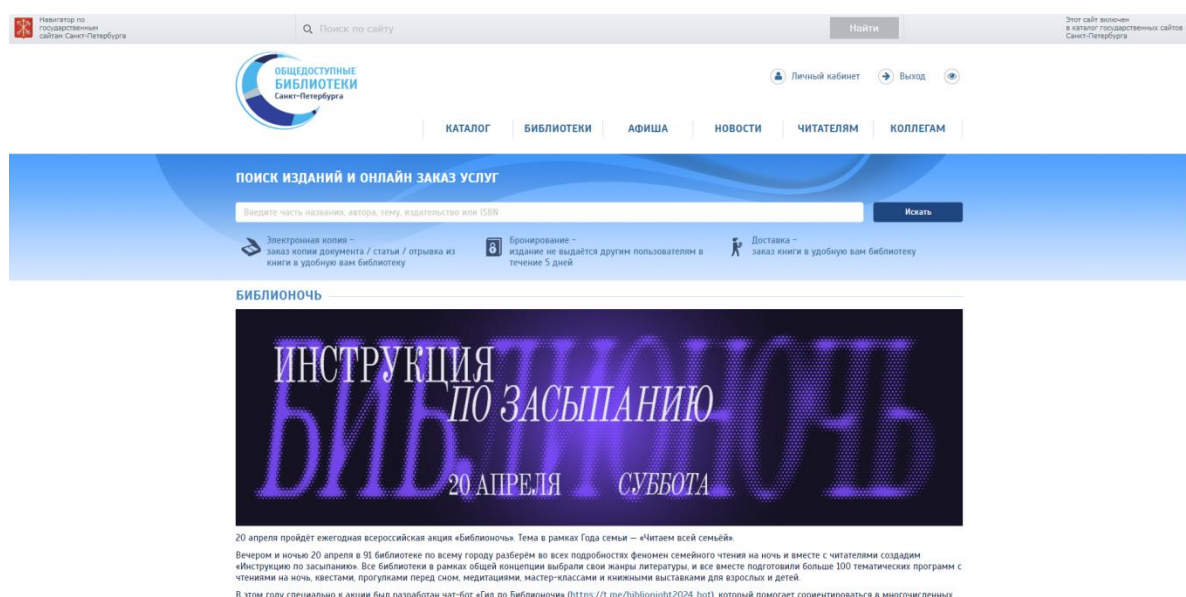


Рисунок 1.7 – Главная страница сервиса «Общедоступные библиотеки Санкт-Петербурга»

КАТАЛОГ

Издания Статьи

Поиск изданий Евгений Онегин ?

☐ В наличии

Расширенный

Искать

Очистить

Результаты поиска. Найдено 954

Сортировка: По релевантности



Евгений Онегин 12+ **В наличии 1 из 4 экз.**
Автор: Пушкин Александр Сергеевич
Тематика: Русская литература — Поэзия — Сборники (19 в.), Русская литература — Драма — Сборники (19 в.)
Издательство: АСТ (Москва)
Серия: Русская классика
Год издания: 2021



Евгений Онегин. Роман в стихах, стихотворения 12+ **В наличии 5 экз.**
Автор: Пушкин Александр Сергеевич
Тематика: Русская литература — Поэзия (19 в.), Любовь — в художественной литературе
Издательство: Азбука (Санкт-Петербург), Азбука-Аттикус (Санкт-Петербург)
Серия: Мировая классика
Год издания: 2020



Евгений Онегин **В наличии 5 из 6 экз.**

Рисунок 1.8 – Результат поиска

КАТАЛОГ

Издания Статьи

Поиск изданий Евгений Онегин ?

☐ В наличии

Расширенный

Библиотека Все библиотеки

Автор

Содержит

Название

Содержит

Тематика

Содержит

Персоналии

Содержит

Серия

Содержит

Издательство

Содержит

Тип издания

Язык

Год издания

Искать

Очистить

Результаты поиска. Найдено 954

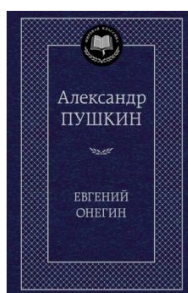
Сортировка: По релевантности

Рисунок 1.9 – Расширенный поиск

КАТАЛОГ

Евгений Онегин. Роман в стихах, стихотворения 12+

Автор: Пушкин Александр Сергеевич



Тематика: Русская литература — Поэзия (19 в.), Любовь — в художественной литературе

Издательство: Азбука (Санкт-Петербург), Азбука-Аттикус (Санкт-Петербург)

Серия: Мировая классика

Год издания: 2020

Объем: 446 с.

Язык: русский

Тип издания: Книги

ISBN: 978-5-389-04903-1



[Полное библиографическое описание](#)

Доставка

Бронирование

Копия

Наличие в библиотеках

Содержание

Адмиралтейский район

- Библиотека "Семеновская" (Московский пр., 50/42)
- Библиотека "Измайловская" (Измайловский пр., 18)

В наличии 2 экз.

В наличии 2 экз.

Центральный район

- Библиотека им. А. И. Герцена (Новгородская ул., 27)

В наличии 1 экз.

Рекомендации

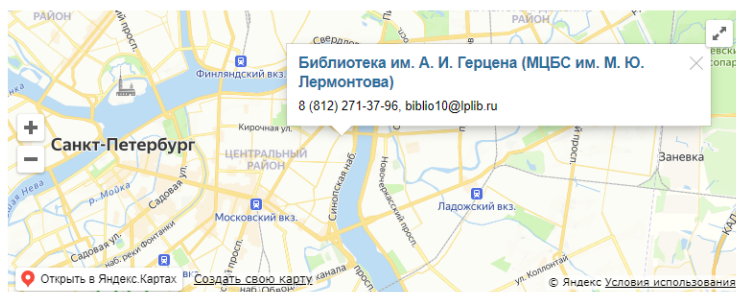
[Другие издания автора](#)

Рисунок 1.10 – Страница книги в сервисе «Общедоступные библиотеки Санкт-Петербурга»

КАТАЛОГ

Заказ бронирования издания

Пушкин, Александр Сергеевич. Евгений Онегин : роман в стихах, стихотворения / Александр Пушкин, 2020. – 446, [1] с. (Введено оглавление)



Информация по услуге:

- Издание не выдается другим пользователям в течение **5 дней**.
- В библиотеке можно оформить выдачу издания на дом.
- При обращении в библиотеку необходимо иметь при себе **паспорт**.
- Срок выполнения заказа — **2 дня**.

Возможен отказ в бронировании, если издание:

- утрачено или выдано
- не выдается на дом
- исчерпан лимит заказов на текущий день

Библиотека * Библиотека им. А. И. Герцена (Новгородская ул., 27)

Комментарий к заказу

Заказать

Рисунок 1.11 – Страница оформления заказа бронирования издания

Номер	Дата	Заглавие	Библиотека	Услуга	Статус
275108	15 апреля 2024 02:20	А. С. Пушкин. Евгений Онегин [Текст] : анализ текста, основное содержание, сочинения / сост. Т. Г. Кучина, 2000. – 158 с.	Библиотека "Парголовская" (Выборгское ш., 369/3, литер А)	Бронирование	Новый

Рисунок 1.12 – Личный кабинет, список заказов

Достоинства сервиса «Общедоступные библиотеки Санкт-Петербурга»

- удобный и функциональный пользовательский интерфейс;
- наличие фильтров поиска по автору, издательству, типу издания, языку, тематике, серии, возрастным ограничениям;
- интуитивно понятное отображение наличия книги в библиотеках, в виде списка адресов, и в виде точек на карте;
- исчерпывающая информация о книгах: имеются изображения обложек книг, номер ISBN, количество страниц. В случае со сборниками произведений приведено содержание книги.

Недостатки сервиса «Общедоступные библиотеки Санкт-Петербурга»

- отсутствие возможности оформить читательский билет в режиме онлайн;
- несовершенная система оформления заказа: пользователь имеет возможность забронировать книгу, не имея читательского билета, не имея привязанного к профилю телефона или паспортных данных.

1.2 Постановка задачи

Необходимо разработать веб приложение для поиска и бронирования книжных изданий в общедоступных библиотеках. Приложение должно обладать следующими возможностями:

Возможность сотрудникам библиотек загружать в базу данных информацию о книгах: автор, заглавие, издательство, серия, тематика, изображение обложки, номер ISBN, количество страниц, содержание книги (оглавление), библиотека и количество экземпляров.

Обработка поисковых запросов пользователей: поиск по любому параметру, фильтры поиска.

Наличие личного кабинета читателя с информацией о забронированных и выданных книгах.

Обработка сервером процесса бронирования книги, внесение соответствующих изменений в БД и отображение клиентом изменившейся информации о наличии и доступности книги.

Для сотрудников библиотек возможность автоматизированной выдачи книг посредством сканирования QR-кода заказа

Минимальные системные требования:

- браузер Google Chrome, Mozilla Firefox, Opera (с поддержкой HTML5, CSS3);
- минимальная скорость интернета: 2-3 Мбит/сек;
- минимальный объем оперативной памяти: 1 ГБ (рекомендуется 2 ГБ и выше).

1.3 Функционал приложения

Функции данного клиент-серверного приложения можно разделить на три группы:

- для авторизованного пользователя;
- для неавторизованного пользователя;
- для сотрудника библиотеки.

Неавторизованный пользователь обладает ограниченным функционалом. Он может просмотреть главную страницу приложения, раздел «Недавние поступления», где отображены книги, недавно поступившие в библиотеки и внесённые в БД, выполнить поиск книги, перейти на страницу книги и тем самым получить информацию о доступности искомого книжного издания в библиотеках.

Диаграмма вариантов использования приложения для неавторизованного пользователя приведена на рисунке 1.13.

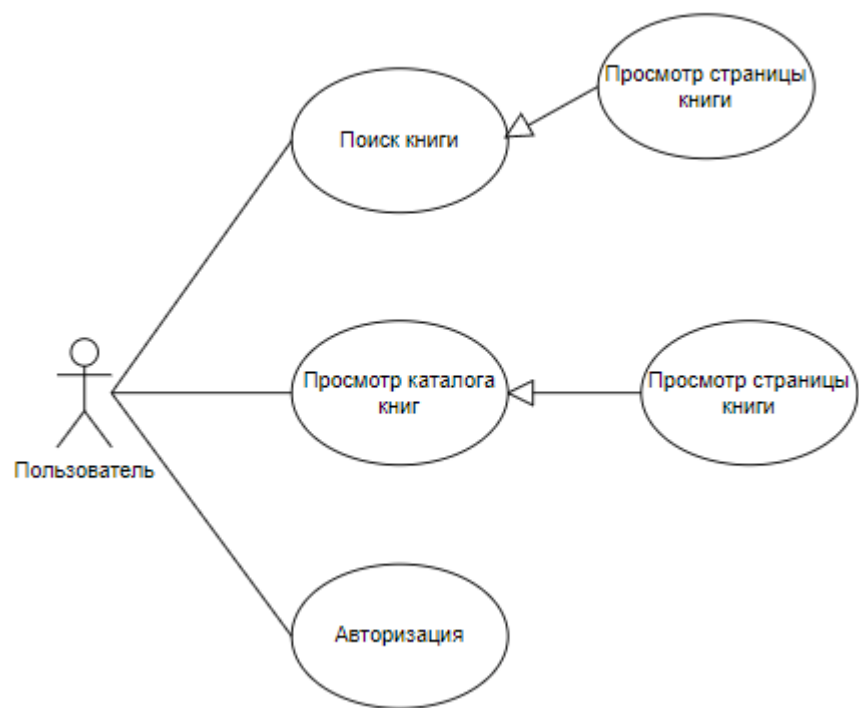


Рисунок 1.13 – Диаграмма прецедентов неавторизованного пользователя

Авторизованный пользователь обладает расширенным функционалом. Помимо функций, доступных неавторизованному пользователю, он имеет доступ к личному кабинету, может забронировать книгу и получить её в библиотеке, предоставив номер заказа. Личный кабинет позволяет увидеть забронированные и полученные на руки книги.

Диаграмма вариантов использования приложения для авторизованного пользователя приведена на рисунке 1.14.



Рисунок 1.14 – Диаграмма прецедентов авторизованного пользователя

Сотрудник библиотеки обладает следующими возможностями. Он может создавать новые записи в базе данных, изменять записи в базе данных, обрабатывать заказы пользователей, выдавая или принимая заказы по номеру заказа бронирования.

Диаграмма вариантов использования приложения для сотрудника библиотеки приведена на рисунке 1.15.

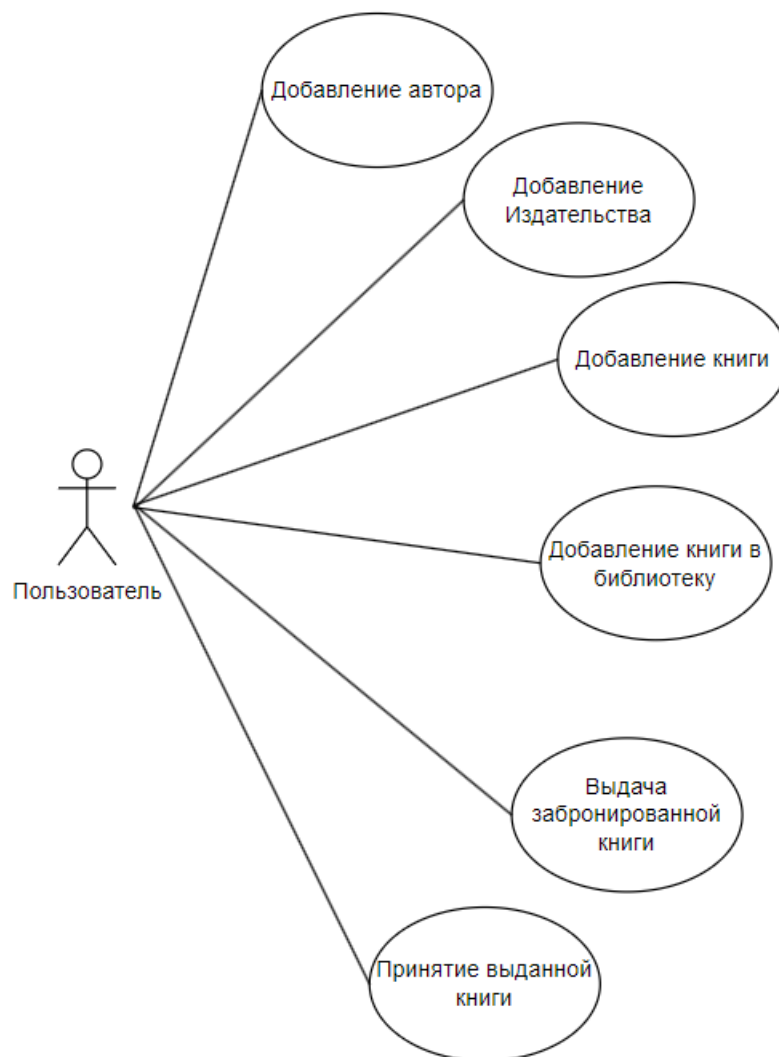


Рисунок 1.15 – Диаграмма прецедентов сотрудника библиотеки

2 Выбор технологий и средств разработки приложения

В этом разделе проведен анализ языков программирования, фреймворков, СУБД, и сред разработки. Исходя из анализа выбраны технологии, которые обеспечивают наибольшее удобство разработки и поддержки приложения

2.1 Выбор платформы исполнения

Выбор платформы, на которой будет работать приложение является важным шагом, так как он определяет то, какие технологии целесообразно применять при разработке. Также, платформа определяет, как пользователи будут взаимодействовать с приложением и какой аудитории приложение будет доступно.

Выбор платформы должен удовлетворять следующим требованиям:

- максимальный охват аудитории;
- скорость и простота разработки и поддержки.

Максимальный охват аудитории позволяет получить наличие сайта или десктопной версии вместе с мобильным приложением. Из этого следует необходимость разработки и поддержки двух, или даже трёх версий приложения, что неизбежно вызовет сложности при разработке одним программистом.

Для удовлетворения вышеупомянутых требований и упрощения разработки, оптимальным выбором является разработка единого веб-приложения с адаптивной вёрсткой, которая позволит использовать приложение как на мобильных устройствах, так и на ПК.

2.2 Выбор языка программирования

2.2.1 Python

Python – это язык программирования общего назначения, который может использоваться, в том числе, для создания серверной логики веб-приложений. Python является одним из самых популярных языков программирования

благодаря своему простому синтаксису, большому количеству инструментов для разработки и библиотек.

Разрабатывать веб-приложения на языке Python позволяют многочисленные веб-фреймворки. Среди наиболее популярных можно выделить Django, Flask и Pyramid. Данные фреймворки предоставляют множество инструментов, таких как маршрутизация, работа с базами данных, поддержка веб-серверов и т.д. Все эти инструменты помогают разработчикам эффективно создавать и поддерживать веб-приложения [6].

2.2.2 C#

C# – объектно-ориентированный язык программирования, разработанный компанией Microsoft. Данный язык программирования является универсальным, и используется для создания десктопных приложений для различных операционных систем, веб-сервисов, нейросетей и мобильных приложений.

Разработка веб-приложений на языке C# обычно подразумевает использование технологий платформы .NET. Платформа .Net является фреймворком и платформой для создания и выполнения приложений на различных устройствах и операционных системах. Этот фреймворк является кроссплатформенным и часто используется для создания крупных веб-порталов и веб-сервисов. Основные особенности данного фреймворка это:

- общезыковая среда исполнения (CLR);
- кроссплатформенность;
- мощная библиотека классов;
- высокая производительность.

2.2.3 JavaScript

JavaScript – мультипарадигменный язык программирования, используемый для разработки веб-приложений. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам [7].

Главным преимуществом языка JavaScript является то, что он поддерживается всеми современными браузерами. Это сделало его неотъемлемой частью фронтенд-разработки. Благодаря таким фреймворкам, как React, Angular и Vue.js, разработка на JavaScript становится еще более эффективной, так как эти фреймворки предоставляют мощные инструменты для разработки веб-приложений. Они позволяют осуществлять управление состоянием приложения, маршрутизацией, а также предоставляют возможности для динамического обновления контента, что позволяет создавать высокопроизводительные приложения с отзывчивым интерфейсом.

JavaScript также может быть использован на серверной стороне благодаря платформе Node.js. Node.js – это программная платформа, которая превращает JavaScript из узкоспециализированного языка в язык общего назначения. Данная платформа делает язык JavaScript универсальным решением для fullstack разработки.

JavaScript был выбран в качестве языка программирования для разработки данного проекта. Этот выбор обусловлен возможностью использования одного и того же языка как на стороне клиента, так и на сервере, что упрощает разработку и поддержку проекта.

2.3 Выбор фреймворка

2.3.1 React

React – это JavaScript-библиотека для создания пользовательских интерфейсов, разработанная Facebook. Цель React — предоставить высокую скорость разработки, простоту написания кода и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, Redux и GraphQL [8][9].

Особенности и возможности React:

Виртуальный DOM. React использует виртуальный DOM, который позволяет вычислять разницу между предыдущим и текущим состояниями

интерфейса и обновлять только измененные его части, улучшая производительность приложения.

Гибкость. React можно использовать вместе с другими библиотеками и фреймворками, такими как Redux, для управления состоянием приложения, что обеспечивает большую гибкость в разработке.

Однонаправленная передача данных. Свойства передаются от родительских компонентов к дочерним. Компоненты получают свойства как множество неизменяемых значений, поэтому компонент не может напрямую изменять свойства, но может вызывать изменения через callback-функции. Такой механизм называют «свойства вниз, события вверх».

2.3.2 Vue.js

Vue.js – это прогрессивный фреймворк для разработки пользовательских интерфейсов и веб-приложений на языке JavaScript. Он решает задачи уровня представления (view) и модели представления (viewModel).

Главным преимуществом Vue.js является его реактивный подход к разработке. Реактивность означает, что представление в модели MVC изменяется по мере изменения модели. В Vue разработчики просто привязывают представление к соответствующей модели, и Vue автоматически наблюдает за изменениями в модели и перерисовывает представление. Эта функция делает управление состоянием Vue довольно простым и интуитивно понятным.

Также Vue используется вместе с другими библиотеками и инструментами, такими как vue-router для маршрутизации, Vuex или Pinia для управления состоянием приложения, что делает его полноценным фреймворком для разработки масштабируемых и производительных одностраничных приложений.

В целом, Vue.js представляет собой эффективный и гибкий инструмент для создания современных веб-приложений с плавным и отзывчивым пользовательским интерфейсом.

2.3.3 Nuxt.js

Nuxt.js – это бесплатная библиотека JavaScript с открытым исходным кодом, основанная на Vue.js, Nitro и Vite. Nuxt вдохновлен Next.js, который представляет собой фреймворк аналогичного назначения, основанный на React.js.

Фреймворк рекламируется разработчиками как «мета-фреймворк для универсальных приложений». Термин «универсальный» используется здесь в том смысле, что цель платформы — дать пользователям возможность создавать веб-представления в JavaScript с использованием системы однофайловых компонентов Vue.js, которая может функционировать как в виде представлений одностраничного приложения (SPA) в браузере, а также веб-представления, которые рендерятся на сервере, а затем (после рендеринга на сервере) «регидратируются» до полной функциональности SPA [10][11].

Nuxt.js предоставляет множество инструментов для разработки приложений. Он включает в себя маршрутизацию, предварительную загрузку данных, генерацию статических файлов и многое другое. Nuxt.js также предоставляет стандартную структуру проекта и механизмы для управления состоянием приложения, что облегчает его разработку и поддержку [12].

Nuxt открывает возможности для написания fullstack-приложений. Он имеет в своём составе серверный движок Nitro, который используется для управления серверной логикой приложения, для написания которой используется HTTP фреймворк h3 [13].

В качестве среды исполнения Nitro использует Node.js. Таким образом, Nuxt предоставляет дополнительные уровни абстракции через Nitro и h3, но базовой средой выполнения остается Node.js, и все преимущества и возможности Node.js остаются доступны.

В качестве фреймворка был выбран Vue в сочетании с Nuxt. Это обеспечивает лучшую производительность по сравнению с React, благодаря встроенной поддержке серверного рендеринга и автоматической маршрутизации.

2.4 Выбор СУБД

2.4.1 MySQL

					ПГУ 1.090301.29.001 ПЗ	Лист
						27
Изм.	Лист	№ докум.	Подпись	Дата		

MySQL – это самая популярная система управления базами данных, которая высоко ценится разработчиками за свою скорость работы и защиту доступа к данным. Защита доступа к данным в ней реализована с помощью функции разграничения доступа. Это одна из самых популярных реляционных баз данных, известная своей надежностью и производительностью.

2.4.2 PostgreSQL

PostgreSQL – это кроссплатформенная объектно-реляционная система управления базами данных, которая предоставляет широкий спектр функций и возможностей для хранения, организации, управления и анализа данных.

Основные преимущества PostgreSQL:

- механизмы транзакций и репликации;
- наследование;
- расширяемость;
- расширяемая система встроенных языков программирования.

PostgreSQL была выбрана в качестве СУБД в данном проекте, так как она имеет широкие возможности, надежность и поддержку современных функций SQL. Она обеспечивает высокую производительность и масштабируемость, что делает её оптимальным выбором

Также, для облегчения разработки в области работы с базой данных была использована технология объектно-реляционного преобразования данных – PrismaORM.

Prisma — это ORM (англ. Object-RelationalMapping, рус. объектно-реляционное отображение, или преобразование) нового поколения с открытым исходным кодом для Node.js и TypeScript. Она состоит из следующих инструментов:

- Prisma Client: Автогенерируемый и типобезопасный клиент базы данных.
- Prisma Migrate: Декларативное моделирование данных и миграции с возможностью пользовательского редактирования.

– Prisma Studio: Современный пользовательский интерфейс для просмотра и редактирования данных.

Основным преимуществом работы с Prisma Client является то, что он позволяет разработчикам мыслить объектами и поэтому предлагает привычный и естественный способ рассуждать о своих данных.

В Prisma Client нет концепции экземпляров модели. Вместо этого он помогает сформировать запросы к базе данных, которые всегда возвращают простые объекты JavaScript [14].

2.5 Выбор среды разработки

Среда разработки должна обеспечивать удобство работы, продуктивность, поддержку необходимых инструментов и расширений. Наиболее популярными в настоящее время являются следующие варианты:

- WebStorm;
- IntelliJ IDEA;
- Visual Studio Code.

2.5.1 WebStorm

WebStorm – это интегрированная среда разработки, разработанная компанией JetBrains. Она предоставляет поддержку современных JavaScript фреймворков, таких как React, Angular и Vue.js.

Webstorm предоставляет множество инструментов для разработки, таких как автодополнение, навигация по коду, отладка, рефакторинг интеграция с системами управления версиями.

Основным недостатком WebStorm является его стоимость, так как эта среда разработки является коммерческим продуктом и требует покупки лицензии. Это может быть ограничивающим фактором для небольших команд или индивидуальных разработчиков.

2.5.2 IntelliJ IDEA

IntelliJ IDEA — это среда разработки, разработанная, как и WebStorm, компанией JetBrains. Эта среда разработки поддерживает широкий спектр языков

					ПГУ 1.090301.29.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

программирования и инструментов для разработки. IntelliJ IDEA также как и Webstorm предлагает широкий набор инструментов для разработки, а основным её преимуществом является поддержка по умолчанию большого количества языков программирования

К недостаткам можно отнести то, что IntelliJ IDEA является коммерческим продуктом, и требует покупки лицензии.

2.5.3 Visual Studio Code

Visual Studio Code — это бесплатный, легковесный и мощный редактор кода, разработанный Microsoft. Он предоставляет широкий набор возможностей для разработки, от базового редактирования текста до интеграции с системами контроля версий и инструментами разработки.

Достоинства Visual Studio Code:

- кроссплатформенность. VS Code доступен для Windows, macOS и Linux, что крайне сильно расширяет аудиторию его пользователей;

- расширяемость. В VS Code доступен обширный каталог расширений, которые позволяют интегрировать поддержку различных языков программирования, фреймворков, инструментов для работы с базами данных и других необходимых утилит;

- интеллектуальная поддержка кода. Благодаря встроенным функциям IntelliSense, VS Code предоставляет интеллектуальные подсказки, автодополнение, рефакторинг кода и поддержку навигации, что ускоряет процесс разработки и улучшает качество кода.

Для разработки приложения была выбрана среда разработки Visual Studio Code. Она предлагает широкий набор инструментов и расширений, поддержку Git, интеллектуальные функции для работы с кодом, а также она не требует покупки лицензии. Эти преимущества делают Visual Studio Code оптимальным выбором в качестве среды разработки для данного проекта.

Таким образом, для разработки проекта были выбраны следующие технологии:

- язык программирования JavaScript;

- фреймворки Vue и Nuxt;
- платформа Node.js, серверный движок Nitro и HTTP-фреймворк h3;
- СУБД PostgreSQL и ORM Prisma;
- среда разработки Visual Studio Code.

2.6 Выбор паттерна проектирования архитектуры

При разработке любого веб-приложения важную роль играет правильный выбор архитектурного паттерна. Самыми популярными паттернами на сегодняшний день являются MVC, MVP и MVVM. Выбор паттерна проектирования зависит от проблемы, которую необходимо решить, а также выбор паттерна взаимосвязан с выбором фреймворка и технологий, которые используются при разработке. Основной задачей любого паттерна является разделение приложения на несколько компонентов, каждый из которых решает разные задачи.

2.6.1 Архитектурный паттерн MVC

Паттерн MVC или Model-View-Controller является фундаментальным шаблоном, который подразумевает разделение проекта на три основные части: модель (Model), представление (View) и контроллер (Controller) [5]. Основная идея использования MVC заключается в том, что контроллер и представления зависят от модели, но модель остается независимым компонентом. Пользователь системы производит некоторые действия с компонентами интерфейса, который предоставляет данную информацию контроллеру. Тот же, в свою очередь, производит определённые действия над моделью данных, после чего представление отображает результат преобразований.

Основные свойства данного паттерна:

- вид осведомлен о модели, т.к. контроллер передает представлению модель;
- вид не имеет ссылки на контроллер, т.е. представление ничего не знает о контроллере;

– отношение между контроллером и видом: множество к одному, это означает, что контроллер может выбрать различные представления для визуализации конкретной задачи.

2.6.2 Архитектурный паттерн MVP

MVP или Model-View-Presenter подобен паттерну MVC, за исключением того, что роль контроллера играет презентер. Данный шаблон часто применяется при создании десктопных интерфейсов. Существует односторонняя привязка между представлением и презентером.

Основные свойства данного паттерна:

- вид не осведомлен о модели, т.к, презентер обновляет представление на основе запрошенных данных;
- вид и презентер осведомлены друг о друге; – отношение между презентером и видом: один к одному.

2.6.3 Архитектурный паттерн MVVM

Шаблон MVVM или Model – View – ViewModel — это следующий этап в развитии MVC. Его отличительными чертами являются модель представления (ViewModel) и связывание данных. MVVM обеспечивает основу для построения клиентских приложений, которые делают взаимодействие с пользователем и обратную связь более отзывчивыми, избегая при этом затратного дублирования кода в рамках всей архитектуры. Этот шаблон упрощает также модульное тестирование. Основные свойства данного паттерна:

- вид не осведомлен о модели;
- модель-представления не знает о виде;
- вид следит за изменениями в модели-представления;
- отношение между моделью-представления и видом – многие ко многим.

Наглядное представление взаимодействия компонентов друг с другом в каждом шаблоне проектирования отражено на рисунке 2.1.

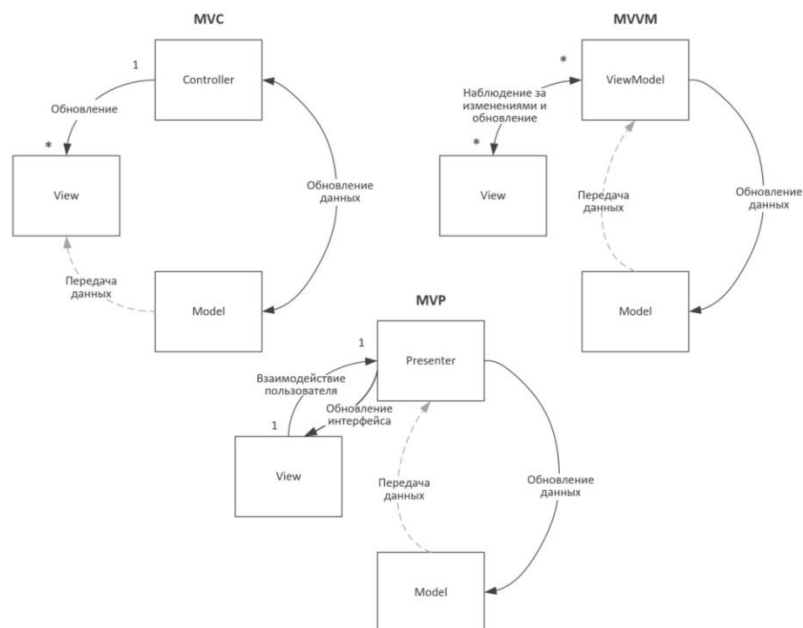


Рисунок 2.1 – Схема взаимодействия компонентов в архитектурных шаблонах проектирования

Для реализации приложения был выбран архитектурный паттерн проектирования MVVM, так как концепции этого паттерна наиболее близки к тому, как работают современные фреймворки, в особенности ранее выбранный фреймворк Vue. Большинство современных фреймворков для разработки веб-приложений работают в парадигме реактивного программирования: отслеживают изменения в своем состоянии, распространяют уведомления об изменениях для всех своих компонентов, автоматически отрисовывают представления в ответ на изменения состояния. Для реализации этих возможностей реактивные веб-приложения применяют принципы проектирования MVVM, асинхронные методики, позволяющие избежать блокирования взаимодействия с пользователем, и элементы функционального программирования. Шаблон проектирования MVVM и реактивное программирование не имеют прямого отношения друг к другу, но у них общая направленность: обе концепции стремятся сделать приложение более отзывчивым и надежным с точки зрения пользователя.

Vue.js предоставляет множество встроенных директив двустороннего связывания данных, тем самым реализуя шаблон MVVM, а новый подход к организации компонентов – Composition API – улучшает структуру и

организацию кода в компонентах, позволяя удобнее разделять представление и модель представления.

В результате, архитектура приложения должна выглядеть следующим образом:

Модель (Model).

В приложении моделью будут данные о книгах, пользователях, бронированиях и других сущностях. Эти данные будут храниться в базе данных.

Представление (View).

Представлениями будут пользовательские интерфейсы страниц приложения, такие как страница поиска книг, страницы с информацией о книгах, страницы личных кабинетов пользователей и прочее. Представления будут реагировать на действия пользователя, следить за изменениями в модели представления и реактивно отображать обновлённые данные, полученные от модели представления.

Модель представления (ViewModel).

Модель представления – это связующее звено между Моделью и Представлением. Она хранит данные в объекте, известном как хранилище. В хранилище находятся данные, которые могут понадобиться приложению в любой момент времени. Это называют состоянием приложения. Связь между представлением и моделью представления реализуется с помощью механизмов связывания данных. Это специальные директивы Vue и прочие методы, которые вызываются представлениями. Также модель представления отвечает за получение и передачу данных в модель. Однако эти операции должны быть асинхронными, чтобы пользователи могли продолжать работать с приложением во время их выполнения.

3 Реализация приложения

В данном разделе описан процесс реализации клиент-серверного приложения «Сервис электронного бронирования книжных изданий». Здесь рассмотрены основные компоненты, алгоритмы взаимодействия и структура базы данных. Цель этого раздела – предоставить полную информацию о разработке приложения

3.1 Описание работы приложения

Топология системы и распределение ее компонентов по физическим узлам представлены на рисунке 3.2.

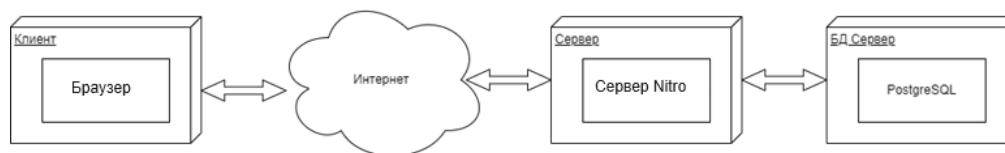


Рисунок 3.2 – UML-диаграмма развёртывания системы

Алгоритм работы приложения заключается в том, что пользователь взаимодействует с клиентской частью, написанной на Vue 3 и Nuxt 3. В зависимости от того, какие действия выполняет пользователь, формируется и отправляется запрос на сервер. Сервер, в свою очередь, обрабатывает запрос. Если в процессе обработки запроса происходит ошибка, сервер сообщает об этом клиенту. В случае успешной обработки запроса сервер взаимодействует с базой данных для выполнения операций, таких как создание, редактирование, получение или удаление данных. После успешного выполнения операции с базой данных сервер возвращает клиенту необходимые данные или подтверждение выполнения операции.

Последовательность выполняемых действий представлена на рисунке 3.1.

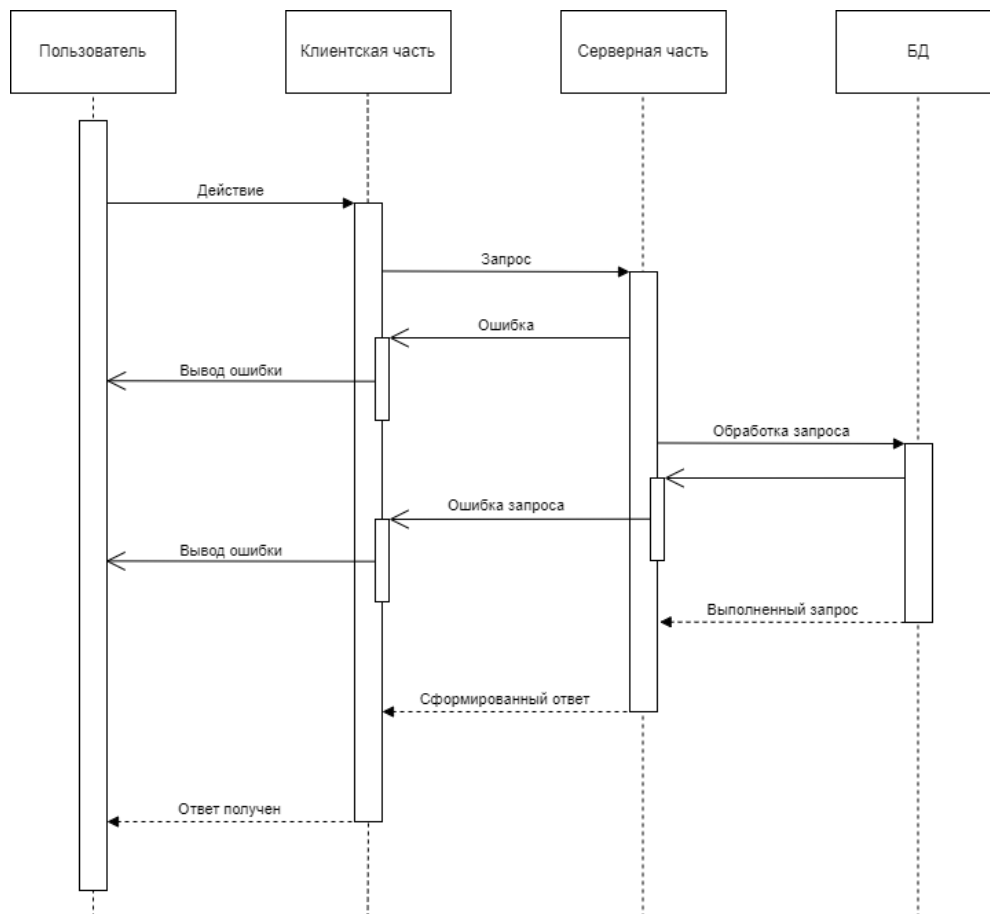


Рисунок 3.1 – UML-диаграмма последовательности функционирования приложения

3.2 Структура базы данных

Для реализации клиент-серверного приложения, способного хранить информацию о пользователях и книгах, необходимо провести анализ и решить, какие сущности и связи будут присутствовать в базе данных. Поскольку структура базы данных является ключевым аспектом ее эффективности и функциональности, в данном разделе рассматриваются основные таблицы, атрибуты и связи между ними, которые были разработаны в рамках данной работы.

Приложение должно предоставлять возможность сотрудникам библиотек быстро и эффективно вносить новые книги в базу данных. Проведя анализ, было решено, что в разрабатываемом приложении будут использоваться следующие таблицы:

Таблица Authors. В этой таблице содержится информация об авторах книг.

Таблица 1 – Authors

Поле	Тип данных	Описание	Ключ
id	TEXT	Идентификатор автора	РК
name	TEXT	Имя автора	
info	TEXT	Информация об авторе	

Таблица Publishers. В данной таблице содержится информация об издательствах.

Таблица 2 – Publishers

Поле	Тип данных	Описание	Ключ
id	TEXT	Идентификатор издательства	РК
name	TEXT	Наименование издательства	
info	TEXT	Информация об издательстве	

Таблица Libraries. В ней будет содержаться информация о библиотеках, включая их адрес.

Таблица 3 – Libraries

Поле	Тип данных	Описание	Ключ
id	TEXT	Идентификатор библиотеки	РК
name	TEXT	Название библиотеки	
info	TEXT	Информация библиотеке	

Поле	Тип данных	Описание	Ключ
address	TEXT	Адрес библиотеки	
latitude	Double Precision	Координаты (широта)	
longitude	Double Precision	Координаты (долгота)	

Таблица Books. Данная таблица хранит записи о книжных изданиях.

Таблица 4 – Books

Поле	Тип данных	Описание	Ключ
id	TEXT	Идентификатор книги	PK
author	TEXT	Автор книги	FK
title	TEXT	Название книги	
coverImage	TEXT	Изображение обложки	
ISBN	TEXT	ISBN	
year	TEXT	Год издания книги	
publisher	TEXT	Издательство	FK
info	TEXT	Дополнительная информация о книге	

Таблица LibraryBook. В этой таблице будет содержаться информация о конкретных экземплярах книг, находящихся в конкретной библиотеке. Именно эти записи должны будут вносить сотрудники библиотек, выбирая из списка свою библиотеку, и книжное издание из таблицы Books.

Таблица 5 – LibraryBook

Поле	Тип данных	Описание	Ключ
id	String	Идентификатор книги	РК
book	TEXT	Книжное издание	FK
library	TEXT	Библиотека	FK
amount	INT	Количество книг в библиотеке	
amountAvailable	INT	Количество незабронированных книг в библиотеке	

Таблица Users. В данной таблице хранятся данные о пользователях.

Таблица 6 – Users

Поле	Тип данных	Описание	Ключ
id	TEXT	Идентификатор пользователя	РК
firstName	TEXT	Имя пользователя	
lastName	TEXT	Фамилия пользователя	
email	TEXT	Электронная почта пользователя	
password	TEXT	Пароль пользователя	
admin	TEXT	Роль пользователя	

createdAt	TimeStamp	Дата создания записи	
updatedAt	TimeStamp	Дата обновления записи	

Таблица RefreshToken. В данной таблице хранятся данные о пользователе, и его токен обновления (RefreshToken)

Таблица 7 – RefreshToken

Поле	Тип данных	Описание	Ключ
id	TEXT	Идентификатор токена	PK
token	TEXT	Значение токена	FK
userId	TEXT	Идентификатор пользователя	FK
createdAt	TimeStamp	Дата создания записи	
updatedAt	TimeStamp	Дата обновления записи	

Таблица BookedBooks. В данной таблице хранятся данные о забронированных книгах.

Таблица 8 – BookedBooks

Поле	Тип данных	Описание	Ключ
id	TEXT	Идентификатор забронированной книги	PK
book	TEXT	Идентификатор книги	FK

Поле	Тип данных	Описание	Ключ
user	TEXT	Идентификатор пользователя	FK
createdAt	TimeStamp	Дата создания записи	
updatedAt	TimeStamp	Дата обновления записи	
received	BOOLEAN	Книга получена или нет	
OrderCode	TEXT	Номер заказа	

Таблица Favorites – в данной таблице хранится информация об избранных книгах пользователей

Таблица 9 – Favorites

Поле	Тип данных	Описание	Ключ
id	TEXT	Идентификатор токена	PK
userId	TEXT	Идентификатор пользователя	FK
bookId	TEXT	Идентификатор книги	FK

Графически структура базы данных показана на рисунке 3.3

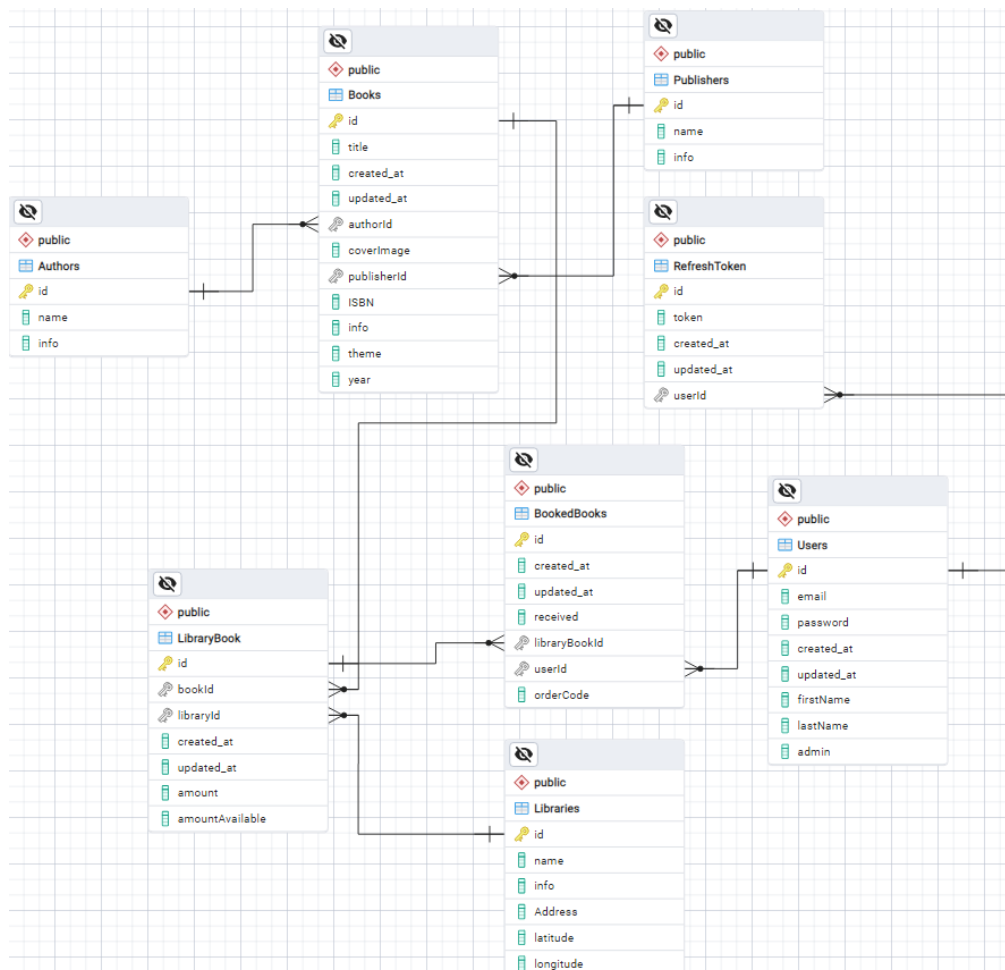


Рисунок 3.3 – Структура базы данных

3.3 Описание компонентов

В соответствии с архитектурой MVVM, приложение имеет три основные части:

- представление (View);
- модель представления (ViewModel);
- модель (Model).

Представление отвечает за визуальное отображение информации на странице, и за предоставление пользователю возможностей ввода информации, например, для авторизации или поиска и бронирования книг.

И представление, и модель представления представлены Vue-компонентами. Vue использует так называемые однофайловые компоненты которые позволяют инкапсулировать содержимое и логику в каждый

компонент [15]. Каждый компонент содержит два блока – блок template, содержащий html разметку и блок script, содержащий JavaScript-код, описывающий поведение компонента.

Некоторые методы модели представления, которые многократно используются разными представлениями вынесены в каталог «composables». В контексте Vue, composable – это функция, использующая Composition API Vue для инкапсуляции и повторного использования логики с отслеживанием состояния [16]. А благодаря Nuxt все функции из этого каталога автоматически импортируются и могут быть использованы в любом компоненте приложения [17].

Модель представляет собой сущность, которая содержит данные, необходимые для отображения на пользовательском интерфейсе. Она отвечает за хранение и управление данными, используемыми в приложении, и содержит всю логику работы с БД.

В данном приложении модель представлена различными объектами Prisma, API, написанными с использованием фреймворка h3, и функциями, осуществляющими доступ к базе данных.

UML-диаграмма компонентов приложения отражена на рисунке X

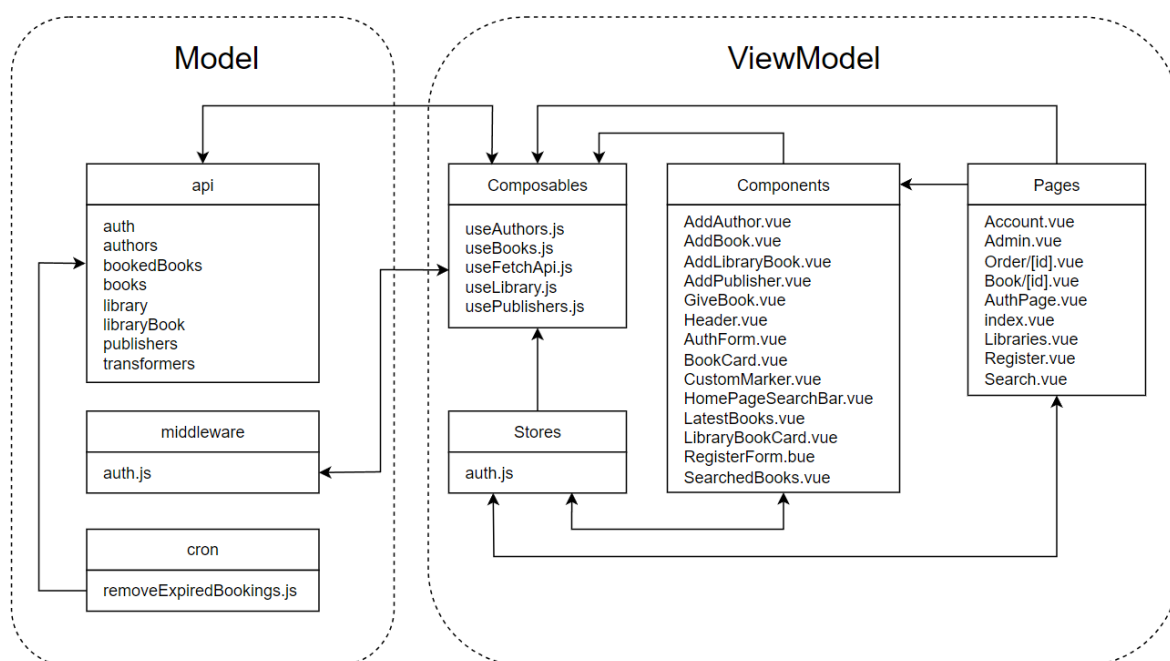


Рисунок 3.4 – UML-диаграмма компонентов приложения

Модель содержит конечные точки API, реализующие HTTP запросы к базе данных, middleware для проверки токена доступа и компонент, использующий планировщик задач cron для регулярного обновления БД

- Auth – регистрация, аутентификация, авторизация пользователя, обновление токена, на листинге 1 представлен метод авторизации;

- Authors – создание и получение автора;

- BookedBooks – представляет всю логику, связанную с бронированием книг;

- Books – создание книги, получение информации книги, а также вся логика, связанная с поиском книг;

- Library – получение информации о библиотеках;

- LibraryBook – представляет методы добавления книг в конкретную библиотеку, получение книг из конкретных библиотек;

- Publishers – создание и получение информации о издателях;

- Transformers – преобразование некоторых результатов, возвращённых базой данных, для более удобного использования представлениями и моделями представлений;

- Middleware – используется для защиты некоторых конечных точек API, реализует проверку авторизации пользователя, получая и расшифровывая токен доступа из заголовков HTTP-запроса;

- RemoveExpiredBookings.js – удаление книг из таблицы забронированных, если их не отметили как «полученные» в нужный срок. Операция удаления реализована с помощью планировщика задач cron, который в назначенное время автоматически запускает функцию проверки срока бронирования всех забронированных книг.

Модели представлений представлены множеством разных компонентов, таких как страницы(pages), composables, компоненты, хранилище.

Composables:

- useAuthors.js – обрабатывает события создания и получения автора;

– useFetchApi.js – получает токен доступа из хранилища и прикрепляет его к заголовкам HTTP запросов, функция представлена на листинге 4;

– useBooks.js – обрабатывает события создания книги, получения информации о книге, поиска книг, бронирования книг;

– usePublishers.js – обрабатывает события получения информации о библиотеках.

Stores:

– auth.js – хранилище (store), которое хранит состояние приложения. Используется для хранения информации о пользователе, в том числе токен доступа. Для сохранения состояния после обновления страницы, использует куки браузера, автоматически сохраняя в них состояние с помощью библиотеки piniapersistedstate.

Components:

– AddAuthor.vue – добавление автора в БД;

– AddBook.vue – добавление книги в БД;

– AddLibraryBook.vue – добавление книги в конкретную библиотеку;

– AddPublisher.vue – добавление издателя в БД;

– GiveBook.vue – компонент, реализующий обработку заказов администратором (сотрудником библиотеки): поиск заказа по номеру, сканирование QR-кода заказа;

– Header.vue – шапка сайта;

– AuthForm.vue – форма авторизации пользователя;

– BookCard.vue – компонент, отрисовывающий информацию об одной книге;

– CustomMarker.vue – компонент кастомного маркера для Яндекс-карт, получает и отображает информацию о библиотеке;

– HomePageSearchBar.vue – строка поиска на главной странице сайта, помещает полученные от пользователя данные в URL-адрес, для последующего использования компонентом Search;

– LatestBooks.vue – компонент, создающий массив компонентов LibraryBookCard на главной странице сайта, для отображения последних поступлений в библиотеки;

– LibraryBookCard.vue – компонент, отрисовывающий информацию об одной книге, существующей в библиотеке;

– RegisterForm.vue – форма регистрации пользователей;

– SearchedBooks.vue – компонент, который создающий массив компонентов BookCard, для отображения результатов поиска книг.

Каждый файл Vue внутри каталога «pages» создает соответствующий URL-адрес (или маршрут), который отображает содержимое файла. Маршрутизация Nuxt основана на vue-router и генерирует маршруты для каждого компонента, созданного в каталоге pages/, на основе имени файла [18]. То есть, при изменении URL на /search, мы попадём в компонент search.vue, если таковой имеется, находящийся в каталоге pages.

Pages:

– Account.vue – страница личного кабинета пользователя, содержит информацию о пользователе и его забронированных книгах;

– Admin.vue – страница администратора, использует компоненты для создания записей в БД и обработки заказов;

– Order/[id].vue – страница заказа. Использует динамическую маршрутизацию, создавая URL-адрес на основе идентификатора заказа;

– Book/[id].vue – страница книги. Использует динамическую маршрутизацию, создавая URL-адрес на основе идентификатора книги;

– AuthPage.vue – страница авторизации пользователя, содержит компонент AuthForm.vue;

– Index.vue – корневая страница приложения;

– Libraries – страница с Яндекс-картой, отображающая библиотеки на карте;

– Register.vue – форма регистрации пользователя;

– Search.vue – страница результатов поиска. Получает строку поиска из URLадреса страницы. Содержит компонент SearchedBooks, который динамически обновляет на основе новых полученных результатов поиска.

Рассмотрим подробнее алгоритмы аутентификации и авторизации

Функция идентификации и аутентификации представлена на листинге 1. В ней происходит идентификация пользователя по введённому email и сравнение хэшей введённого пароля с хэшем, хранимым в базе данных. Функция возвращает информацию о пользователе и токен доступа

Листинг 1. Auth

```
export default define EventHandler(async (event) => {
  const body = await readBody(event)
  const {email, password} = body
  if (!email || !password) {
    return sendError(event, createError({
      statusCode:400,
      statusMessage: 'Invalid params'
    }))
  }
  const user = await getUserByEmail(email)
  if(!user) {
    throw createError({
      statusCode:400,
      statusMessage:'Username or password is invalid',
      message:'Пользовательнезарегистрирован'
    })
  }
  const doesThePasswordMatch = await bcrypt.compare(password,
  user.password)

  if(!doesThePasswordMatch) {
    throw createError({
      statusCode:400,
      statusMessage:'Username or password is invalid',
      message:'Неверный пароль или email'
    })
  }
  const {accessToken, refreshToken} = generateTokens(user)
  await createRefreshToken({
    token: refreshToken,
    userId: user.id
  })
  sendRefreshToken(event, refreshToken)
  return {
    accessToken: accessToken,
    user: userTransformer(user)
  }
})
```

```
}}
```

Функция `useFetchApi` представлена на листинге 2. Она получает токен доступа из хранилища и прикрепляет его к заголовку HTTP-запроса

Листинг 2. `useFetchApi.js`

```
export default (url, options = {}) => {  
  return $fetch(url, {  
    ...options,  
    headers: {  
      ...options.headers,  
      Authorization: `Bearer ${authToken}`  
    }  
  })  
}
```

В хранилище `Pinia` содержатся функции, которые осуществляют запросы к API и выполняют некоторые действия с токенами. Это функции `login`, `register`, `refreshToken`, `getUser`, `reRefreshAccessToken`, `initAuth`, `logout`. Функция `login` представлен на листинге 3, она создает набор из двух токенов – токен доступа и токен обновления. Функция `initAuth` представлена на листинге 4, она представляет собой инициализацию авторизации и вызывается при авторизации и при монтировании корневого компонента `app`, то есть при каждом обновлении страницы. Функция `reRefreshAccessToken` представлена на листинге 5, она нужна для обновления токена доступа, так как он недолговечный. Функция вызывается функцией `initAuth`, а затем вызывается рекурсивно, за минуту до истечения токена доступа. Тем самым сохраняется авторизация пользователя. Для генерации нового токена доступа используется функция `refresh` из компонента модели `Auth`, которая находит в базе данных токен обновления, совпадающий с токеном обновления в куках браузера, если таковой найден, расшифровывает его, проверяет его валидность и генерирует новый токен доступа. Если проверка токена обновления не пройдена, возвращается соответствующее сообщение с ошибкой.

Листинг 3. `Login()`

```
const login = ({ email, password }) => {  
  return new Promise(async (resolve, reject) => {  
    try {  
      const data = await $fetch('/api/auth/login', {
```

```

        method: 'POST',
        body: {
            email,
            password
        }
    })
    setToken(data.access_token)
    setUser(data.user)
    await initAuth()
    resolve(data)
  } catch (error) {
    console.log("err:", error)
    reject(error.data)
  }
})
}

```

Листинг 4. InitAuth()

```

const initAuth = () => {
    return new Promise(async (resolve, reject) => {
    setIsAuthLoading(true)
    try {
        console.log("initAuth")
        await refreshToken()
        await getUser()

        reRefreshAccessToken()

        resolve(true)
    } catch (error) {
        reject(error)
    } finally {
        setIsAuthLoading(false)
    }
    })
}

```

Листинг 5. ReRefreshAccessToken()

```

const reRefreshAccessToken = () => {
    console.log("reRefreshAccessToken")
    if (!authToken.value) {
        return
    }

    const jwt = jwtDecode(authToken.value)
    const newRefreshTime = (jwt.exp * 1000) - Date.now() -
60000
    setTimeout(async () => {
        await refreshToken()
        reRefreshAccessToken()
    }, newRefreshTime);
}

```

Изм.	Лист	№ докум.	Подпись	Дата

Для авторизации используется middleware. Он получает и расшифровывает токен доступа из заголовков HTTP-запроса. Компонент представлен на листинге 6

Листинг 6. Middleware/auth.js

```
export default defineEventHandler(async (event) => {
  const endpoints = [
    '/api/auth/user',
    '/api/bookedBooks'
  ]

  const isHandledByThisMiddleware = endpoints.some(endpoint => {
    const pattern = new UrlPattern(endpoint)

    return pattern.match(event.path)
  })

  if (!isHandledByThisMiddleware) {
    console.log("not handled by this middleware")
    return
  }

  const token = event.node.req.headers['authorization']?.split('
')[1]

  const decoded = decodeAccessToken(token)

  console.log("token:", token)

  if (!decoded) {
    return sendError(event, createError({
      statusCode: 401,
      statusMessage: 'Unauthorized'
    }))
  }

  try {
    const userId = decoded.userId

    const user = await getUserById(userId)

    event.context.auth = { user }

    console.log("middleware auth success")

  } catch (error) {
    return
  }

})
```

Операция бронирования реализована с помощью нескольких функций. Функция `postBookedBook` из `composable useBooks.js` представлена на листинге 7. При бронировании книги должна быть проверена авторизация пользователя, поэтому вызывается функция `useFetchApi`.

Листинг 7. `PostBookedBook()`

```
const postBookedBook = ({ libraryBookId, userId }) => {
  return new Promise(async (resolve, reject) => {
    try {
      await useFetchApi('/api/bookedBooks', {
        method: 'POST',
        body: {
          libraryBookId,
          userId
        }
      })
      resolve(true)
    } catch (error) {
      reject(error.data)
    }
  })
}
```

В листинге 8 приведена одна из функций этого компонента – функция, создающая запись в таблице `bookedBooks` и изменяющая количество доступных книг в таблице `libraryBook`. Эти два действия должны быть атомарной операцией, поэтому они реализованы в рамках одной транзакции.

Листинг 8. `createBookedBook()`

```
export const createBookedBook = (bookData) => {
  return prisma.$transaction([
    prisma.bookedBooks.create({
      data: bookData,
    }),
    prisma.libraryBook.update({
      where: {
        id: bookData.libraryBookId,
      },
      data: {
        amountAvailable: { decrement: 1 }
      }
    })
  ])
}
```

Для реализации адаптивной вёрстки использован CSS-фреймворк Tailwind CSS. С его помощью можно задавать разные стили в зависимости от размера

экрана. Это делается с помощью так называемых «брейкпоинтов». В Tailwind CSS реализуется парадигма «mobile-first». Это означает, что стили по умолчанию – это стили для экранов малой ширины, то есть для мобильных устройств. Для экранов большей ширины используются брейкпоинты md, lg, xl и т.д. Пример реализации адаптивной вёрстки приведён в листинге 9.

Листинг 9. Адаптивная вёрстка.

```
<div class="my-2 h-20 lg:my-2 lg:h-24 w-full content-start
lg:content-center">
  <div class="font-semibold text-xs md:text-sm text-
ellipsis overflow-hidden">
    {{ props.book.title }}
  </div>
  <div class="flex text-xs md:text-sm text-ellipsis
overflow-hidden">
    {{ props.book.author }}
  </div>
  <div class="flex text-xs md:text-sm text-ellipsis
overflow-hidden">
    Издательство: <p class="ml-2 font-semibold"> {{
props.book.publisher }}</p>
  </div>
</div>
```

3.4 Основные алгоритмы взаимодействия клиента и сервера

Для того, чтобы наглядно рассмотреть алгоритмы взаимодействия клиента и сервера были разработаны UML диаграммы деятельности, отображающие последовательности действий между клиентом и сервером.

Первым рассматриваемым алгоритмом является процесс регистрации и авторизации. Для использования всех функций, предлагаемых приложением, необходимо пройти регистрацию и авторизоваться. Если пользователь, не выполнивший вход, пытается воспользоваться функцией, доступной только авторизованным пользователям, он будет автоматически перенаправлен на страницу авторизации. В случае отсутствия аккаунта, пользователь может зарегистрироваться, перейдя на соответствующую страницу и заполнив необходимые поля. Введенные данные отправляются на сервер для обработки и проверки уникальности. Если такие данные уже существуют в базе данных, пользователю будет отправлено соответствующее уведомление. В противном

случае данные будут сохранены в базе данных, и пользователь будет перенаправлен на страницу авторизации.

Если при авторизации введенные данные не совпадают с хранящимися в базе данных, пользователю будет отображено сообщение о некорректных данных. При правильном вводе данных пользователь успешно войдет в систему и получит полный доступ ко всем функциям приложения.

Последовательность описанных действий представлена на рисунке 3.5.

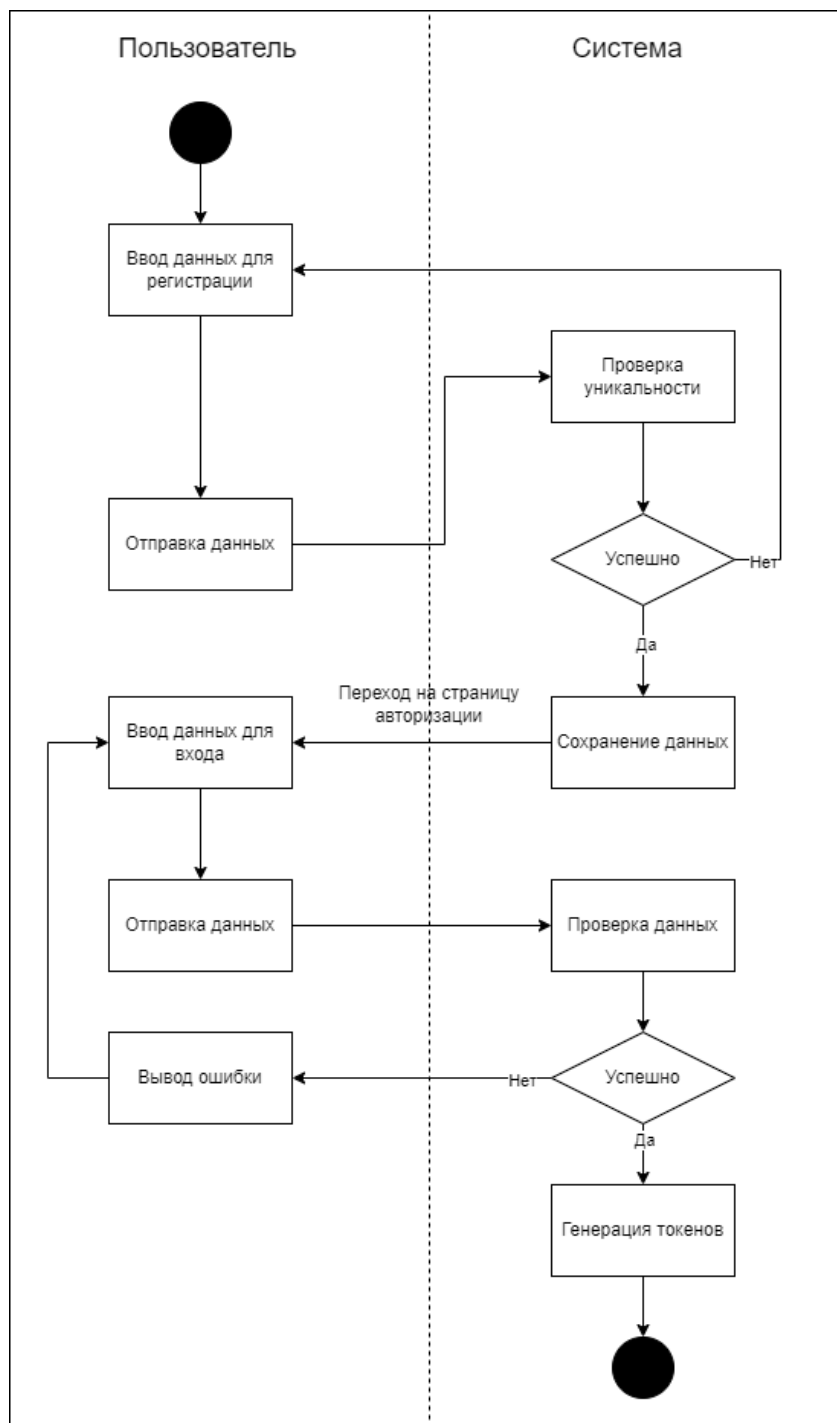


Рисунок 3.5 – UML-диаграмма деятельности регистрации и авторизации

Следующим алгоритмом будет рассмотрен процесс добавления книги в базу данных. Для того, чтобы попасть на страницу добавления книги нужно обладать правами администратора. Права администратора определяются значением true в поле admin таблицы Users.

Как было упомянуто в разделе 3.1 «Структура базы данных», каждая запись в таблицах Издательства и Авторы – уникальна, соответственно, сервер должен выполнять проверку уникальности этих записей и возвращать ошибку в случае, если проверка не пройдена.

Действие «Добавление книги» означает добавление записи в таблицу Books, которая использует идентификаторы издательства и автора в качестве внешнего ключа. При проверке уникальности книги осуществляется проверка существования заданного сочетания издательства, автора и названия и возвращается ошибка в случае, если такая запись найдена в БД.

Действие «Добавление книги в библиотеку» означает добавление записи в таблицу LibraryBook. LibraryBook использует идентификатор книги в качестве внешнего ключа, следовательно, сервер осуществляет проверку существования книги в базе данных и возвращает ошибку, если такой книги не существует в базе данных.

Последовательность описанных действий представлена на рисунке 3.6.

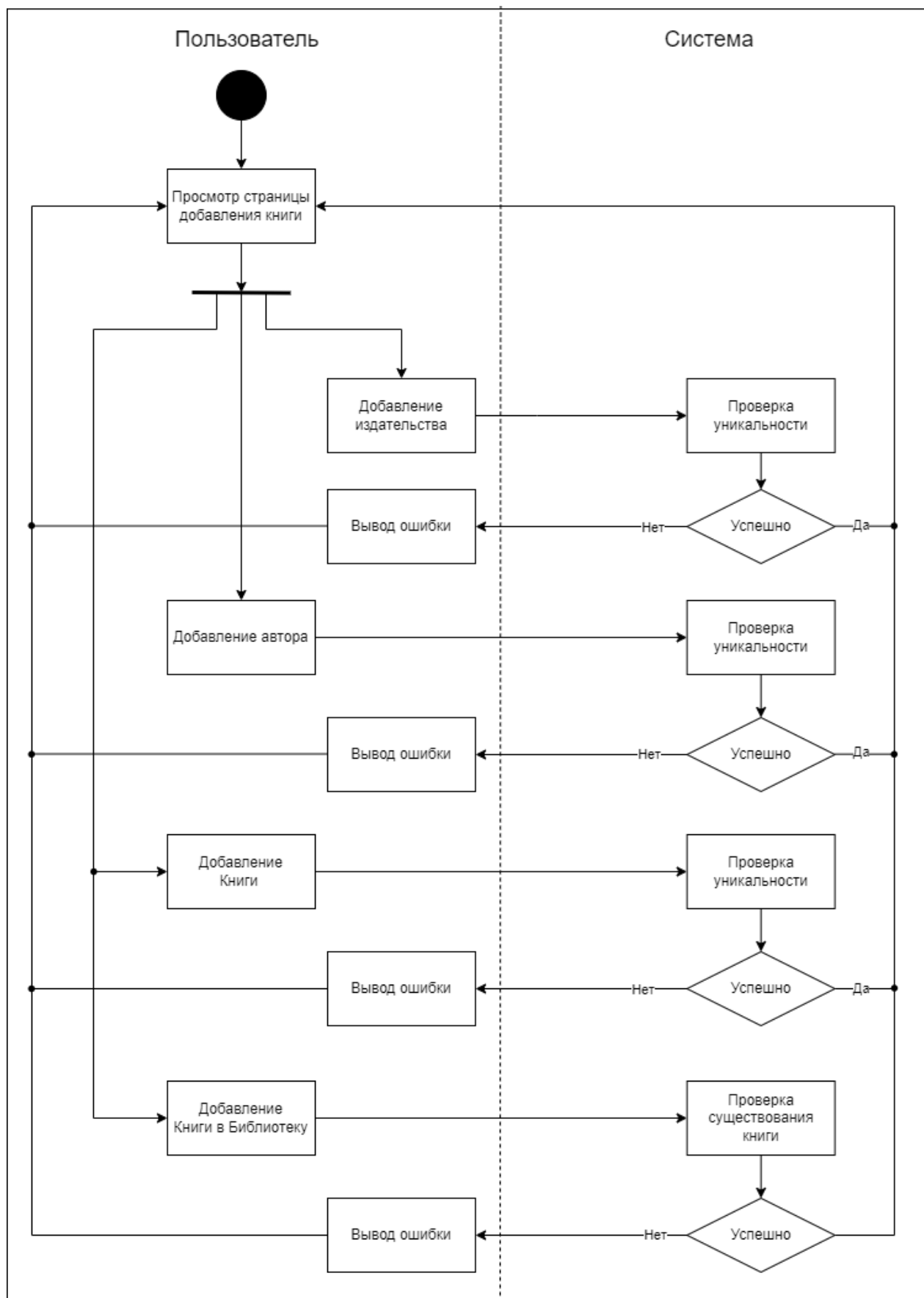


Рисунок 3.6 – UML-диаграмма деятельности добавления записей в БД

Диаграмма деятельности для поиска книг представлена на рисунке 3.7

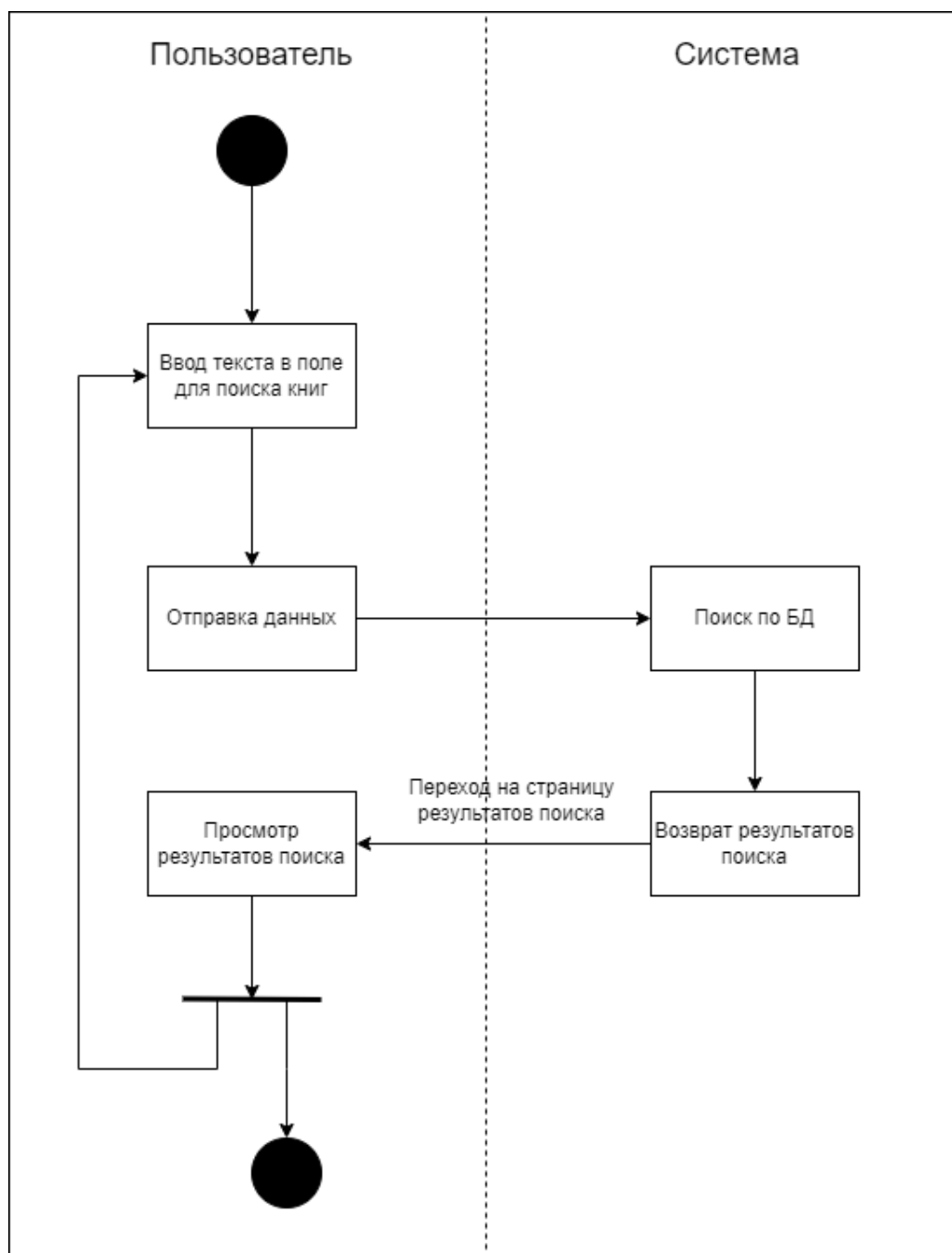


Рисунок 3.7 – UML-диаграмма деятельности поиска книг

Диаграмма деятельности бронирования книги представлена на рисунке 3.8

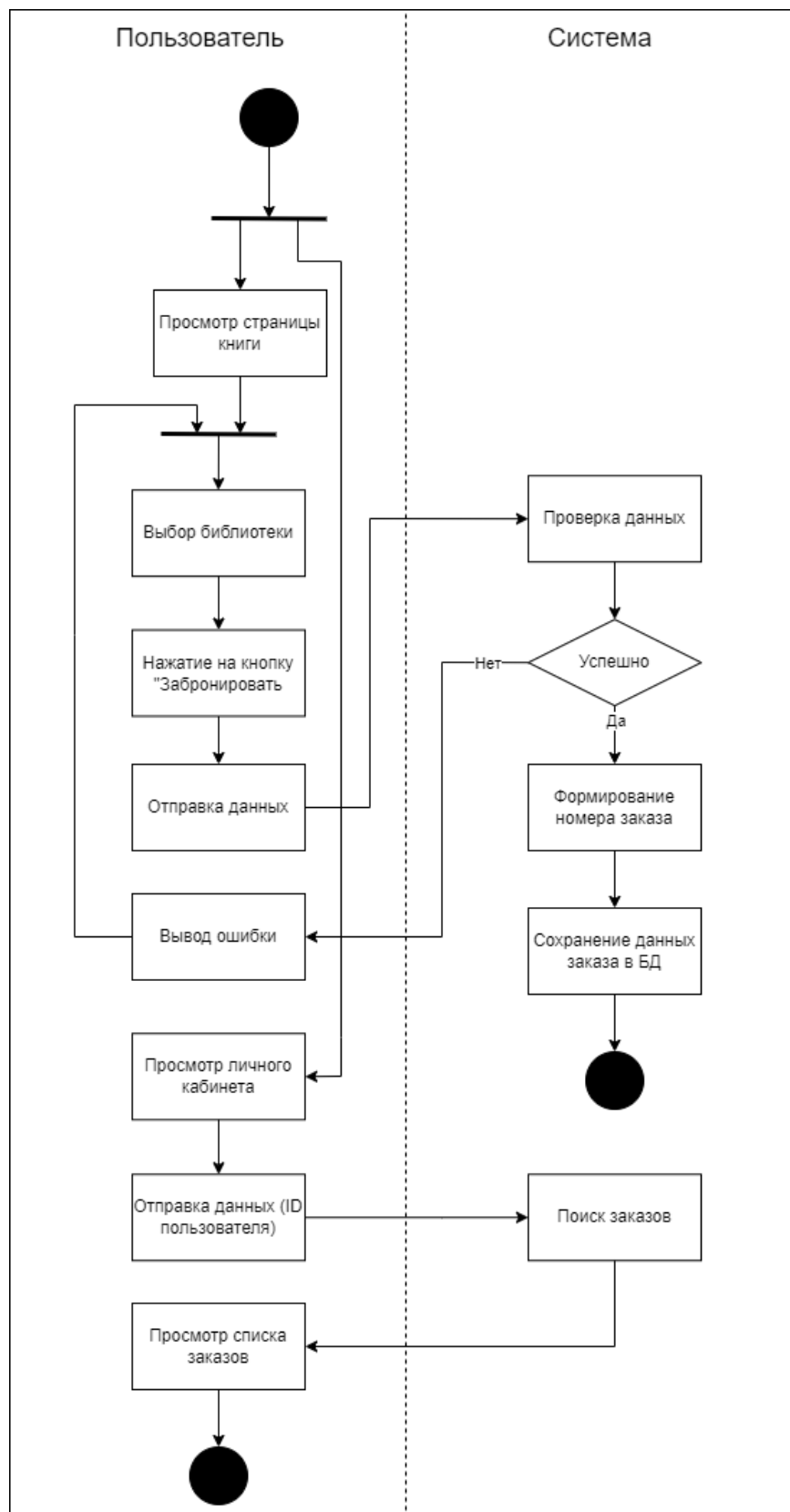


Рисунок 3.8 – UML-диаграмма деятельности бронирования книги

При просмотре забронированной книги и при выдаче заказа бронирования взаимодействуют 3 компонента – пользователь, сотрудник библиотеки и система.

Изм.	Лист	№ докум.	Подпись	Дата

Действие «Ввод номера заказа в поле ввода» означает либо ручной ввод номера, либо ввод путём сканирования QR-кода. Затем система производит поиск заказа по введённому номеру, и возвращает данные сотруднику библиотеки, который производит над заказом необходимую операцию. После чего система обрабатывает полученные данные и сохраняет изменения в БД.

Последовательность описанных действий представлена на рисунке 3.9

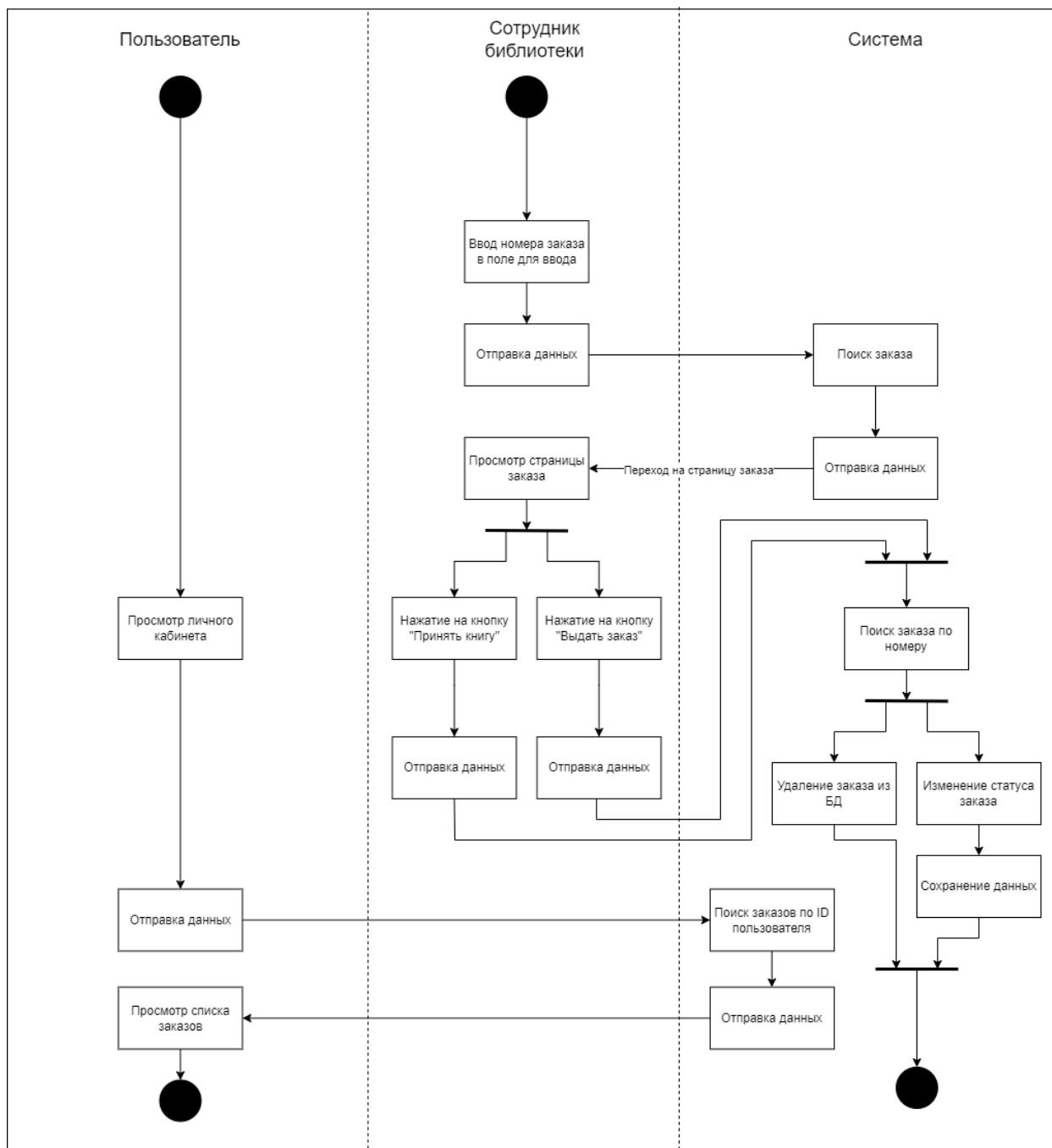


Рисунок 3.9 – UML-диаграмма деятельности работы с заказами

4 Тестирование

Для проверки работоспособности клиент-серверного приложения «Сервис электронного бронирования книжных изданий» необходимо провести его тестирование. Для тестирования были составлены тест-кейсы.

Тест-кейсы составлены для следующих действий:

- регистрация пользователя;
- авторизация пользователя;
- просмотр главной страницы;
- поиск книг по разным параметрам;
- бронирование книги;
- добавление книги в БД;
- добавление книги в библиотеку;
- выдача книги по номеру заказа;
- принятие книги по номеру заказа.

4.1 Регистрация пользователя

На рисунке 4.1 представлен тест-кейс регистрации пользователя.

					ПГУ 1.090301.29.001 ПЗ	Лист
						59
Изм.	Лист	№ докум.	Подпись	Дата		

[ТЕСТОВЫЙ_КОНТУР-1] Регистрация пользователя <small>создан: 27.05.24 изменено: 27.05.24</small>				
Проект:	Сервис электронного бронирования книжных изданий			
Версия:	от 27/05/2024			
Тип:	Тест	Приоритет:	Высокий	
Тестировщик:	Китаев Андрей	Ответственный:	Китаев Андрей Андреевич	
Предварительные действия:	Пользователь перешёл на вкладку «Регистрация»			
Тестовый сценарий:	Шаги	Данные	Ожидаемый результат	Фактический результат
	1 Нажать на кнопку «Зарегистрироваться»	<Все поля ввода пустые>	На экране отобразилась информация о необходимости заполнения всех полей: «Имя: обязательное поле, Фамилия обязательное поле, email обязательное поле»	Совпадает с ожидаемым результатом
	2 Ввести в поля ввода исходные данные, нажать на кнопку «Зарегистрироваться»	Имя: Андрей Фамилия: Китаев Email: test@mail.ru Пароль: 123 Повторите пароль: 123	На экране отобразилось уведомление об ошибке: «Данный пользователь уже зарегистрирован»	Совпадает с ожидаемым результатом
	3 Ввести в поля ввода исходные данные, нажать на кнопку «Зарегистрироваться»	Имя: Андрей Фамилия: Китаев Email: andrey@mail.ru Пароль: 123 Повторите пароль: 123	На экране отобразилась информация о требуемой минимальной длине пароля (8 символов)	Совпадает с ожидаемым результатом
	4 Ввести в поля ввода исходные данные, нажать на кнопку «Зарегистрироваться»	Имя: Андрей Фамилия: Китаев Email: andrey@mail.ru Пароль: 12345678 Повторите пароль: 12345678	На экране появилось уведомление «Вы успешно зарегистрированы»	Совпадает с ожидаемым результатом
Описание				
Тест-кейс предназначен для проверки создания нового пользователя				

Рисунок 4.1 – Тест-кейс регистрации пользователя

4.2 Авторизация пользователя

На рисунке 4.2 представлен тест-кейс авторизации пользователя.

[ТЕСТОВЫЙ_КОНТУР-1] Авторизация пользователя создано: 27.05.24 изменено: 27.05.24				
Проект:	Сервис электронного бронирования книжных изданий			
Версия:	от 27/05/2024			
Тип:	Тест	Приоритет:	Высокий	
Тестировщик:	Китаев Андрей	Ответственный:	Китаев Андрей Андреевич	
Предварительные действия:	Пользователь перешёл на вкладку «Авторизация»			
Тестовый сценарий:	Шаги	Данные	Ожидаемый результат	Фактический результат
	1 Нажать на кнопку «Войти»	<Все поля ввода пустые>	На экране отобразилась информация о необходимости заполнения всех полей	Совпадает с ожидаемым результатом
	2 Ввести в поля ввода исходные данные, нажать на кнопку «Войти»	Email: andrey@mail.ru Пароль: 12345	На экране отобразилось уведомление об ошибке: «Неверный email или пароль»	Совпадает с ожидаемым результатом
	3 Ввести в поля ввода исходные данные, нажать на кнопку «Войти»	Email: andrey@mail.ru Пароль: 12345678	Произошёл переход на главную страницу. На странице можно увидеть шапку сайта в последней вкладке которой отображается имя пользователя, строку поиска, раздел «Недавние поступления»	Совпадает с ожидаемым результатом
Описание				
Тест-кейс предназначен для проверки авторизации пользователя				

Рисунок 4.2 – Тест-кейс авторизации пользователя

4.3 Поиск книг, просмотр страницы книги, бронирование

На рисунке 4.3 представлен тест-кейс поиска книг. На рисунке 4.4 представлен тест-кейс просмотра страницы книги и бронирования книги.

[ТЕСТОВЫЙ_КОНТУР-1] Поиск книг создано: 27.05.24 изменено: 27.05.24				
Проект:	Сервис электронного бронирования книжных изданий			
Версия:	от 27/05/2024			
Тип:	Тест	Приоритет:	Высокий	
Тестировщик:	Китаев Андрей	Ответственный:	Китаев Андрей Андреевич	
Предварительные действия:	Авторизованный пользователь перешёл на главную страницу			
Тестовый сценарий:	Шаги	Данные	Ожидаемый результат	Фактический результат
	1 Оставить поле поиска пустым, нажать на кнопку «Найти»	<Все поля ввода пустые>	На экране отобразилась информация о необходимости заполнения строки поиска	Совпадает с ожидаемым результатом
	2 Ввести в поле ввода исходные данные, нажать на кнопку «Найти»	«Пушкин»	На экране отобразился результат в виде списка книг, найденных по автору «Пушкин». Выше списка отображается количество найденных результатов и пагинация	Совпадает с ожидаемым результатом
	3 Ввести в поле ввода исходные данные, нажать на кнопку «Найти»	«АСТ»	На экране отобразился результат в виде списка книг издательства, введённого в поле ввода. Выше списка отображается количество найденных результатов и пагинация	Совпадает с ожидаемым результатом
Описание				
Тест-кейс предназначен для проверки поиска книг				

Рисунок 4.3 – Тест-кейс поиска книг

[ТЕСТОВЫЙ_КОНТУР-1] Бронирование книги создано: 27.05.24 изменено: 27.05.24				
Проект:	Сервис электронного бронирования книжных изданий			
Версия:	от 27/05/2024			
Тип:	Тест	Приоритет:	Высокий	
Тестирущик:	Китаев Андрей	Ответственный:	Китаев Андрей Андреевич	
Предварительные действия:	Авторизованный пользователь перешёл на страницу книги, нажав на заглавие одной из книг			
Тестовый сценарий:	Шаги	Данные	Ожидаемый результат	Фактический результат
	1 Оставить поле выбора библиотеки пустым, нажать на кнопку «Забронировать»	<Все поля ввода пустые>	На экране отобразилась информация об ошибке: «Сначала выберите библиотеку»	Совпадает с ожидаемым результатом
	2 Нажать на маркер библиотеке на карте		На карте отобразилось модальное окно с информацией о библиотеке (Название, информация, кол-во экземпляров книги, кнопка «Выбрать»)	Совпадает с ожидаемым результатом
	3 Нажать на кнопку «Выбрать» в модальном окне на карте		На экране появилось модальное окно с информацией о выбранной библиотеке и книге и кнопка «Забронировать»	Совпадает с ожидаемым результатом
	4 Нажать на кнопку «Забронировать» в модальном окне		На экране появилось уведомление «Вы забронировали книгу {название книги}»	Совпадает с ожидаемым результатом
	5 Нажать на имя пользователя в шапке сайта		Произошёл переход на страницу «Личный кабинет». На экране отображается информация о пользователе и список забронированных книг (заказов). Список заказов содержит данные книги, библиотеки, статус заказа («Забронирована» или «Выдана»)	Совпадает с ожидаемым результатом
	6 Нажать на один из «заказов»		На экране появилось модальное окно с QR-кодом, содержащим информацию о заказе	Совпадает с ожидаемым результатом
Описание				
Тест-кейс предназначен для проверки бронирования книги				

Рисунок 4.4 – Тест-кейс бронирования книги

4.4 Тестирование функционала администратора

На рисунке 4.5 представлен тест-кейс обработки заказа бронирования книжного издания. На рисунке 4.6 представлен тест-кейс добавления записей в базу данных.

[ТЕСТОВЫЙ_КОНТУР-1] Выдача и приём заказа создано: 27.05.24 изменено: 27.05.24				
Проект:	Сервис электронного бронирования книжных изданий			
Версия:	от 27/05/2024			
Тип:	Тест	Приоритет:	Высокий	
Тестировщик:	Китаев Андрей	Ответственный:	Китаев Андрей Андреевич	
Предварительные действия:	Пользователь авторизован как пользователь с ролью «admin» и перешёл на страницу личного кабинета администратора. Логин: adm@adm, пароль: adm			
Тестовый сценарий:	Шаги	Данные	Ожидаемый результат	Фактический результат
	1	Навести веб-камеру на QR код «не выданного» заказа	QR-код успешно распознан и прочитан, отобразилась кнопка для перехода по ссылке	Совпадает с ожидаемым результатом
	2	Нажать на кнопку «посмотреть заказ»	На экране отобразилась страница обработки заказов. Отображается полная информация о книге, статус заказа, кнопка «Выдать»	Совпадает с ожидаемым результатом
	3	Нажать на кнопку «Выдать»	На экране появилось уведомление «Книга выдана»	Совпадает с ожидаемым результатом
	4	Навести веб-камеру на QR код «выданного» заказа	QR-код успешно распознан и прочитан, отобразилась кнопка для перехода по ссылке	Совпадает с ожидаемым результатом
	5	Нажать на кнопку «посмотреть заказ»	На экране отобразилась страница обработки заказов. Отображается полная информация о книге, статус заказа, кнопка «Выдать»	Совпадает с ожидаемым результатом
	6	Нажать на кнопку «Принять»	На экране появилось уведомление «Книга принята»	Совпадает с ожидаемым результатом
Описание				
Тест-кейс предназначен для проверки Выдачи и Принятия заказа по QR коду				

Рисунок 4.5 – Тест-кейс обработки заказа

[ТЕСТОВЫЙ_КОНТУР-1] Добавление данных в БД создано: 27.05.24 изменено: 27.05.24				
Проект:	Сервис электронного бронирования книжных изданий			
Версия:	от 27/05/2024			
Тип:	Тест	Приоритет:	Высокий	
Тестировщик:	Китаев Андрей	Ответственный:	Китаев Андрей Андреевич	
Предварительные действия:	Авторизованный пользователь с ролью «admin» перешёл в личный кабинет			
Тестовый сценарий:	Шаги	Данные	Ожидаемый результат	Фактический результат
	1 Оставить все поля ввода пустыми, нажать на кнопку «Добавить книгу в библиотеку»	<Все поля ввода пустые>	На экране отобразилось уведомление о том, что необходимо заполнить все поля	Совпадает с ожидаемым результатом
	2 Ввести в поле ввода исходные данные, нажать на кнопку «Добавить книгу в библиотеку»	Библиотека: Городская библиотека №9 Книга «Евгений Онегин, Эксмо» кол-во экземпляров 10	На экране отобразилась информация об ошибке: «Книга уже существует в этой библиотеке»	Совпадает с ожидаемым результатом
	3 Ввести в поле ввода исходные данные, нажать на кнопку «Добавить книгу в библиотеку»	Библиотека: Городская библиотека №4 Книга «Евгений Онегин, Эксмо» кол-во экземпляров 10	На экране отобразилась информация об успешном действии: «Книга успешно добавлена»	Совпадает с ожидаемым результатом
Описание				
Тест-кейс предназначен для проверки добавления записей в БД				

Рисунок 4.6 – Тест-кейс добавления записей в БД

5 Руководство пользователя

5.1 Результат работы приложения

Сервис электронного бронирования книжных изданий представляет собой многостраничный сайт, каждая страница которого состоит из нескольких логических блоков. Шапка сайта повторяется на каждой странице, в ней располагаются кнопки навигации: кнопка перехода на главную страницу, кнопка перехода на страницу библиотек, кнопка перехода на страницу авторизации или в личный кабинет и кнопка переключения цветовой схемы сайта.

На главной странице, помимо шапки, располагается два блока: строка поиска и раздел «Последние поступления», в которых можно увидеть несколько последних книг, появившихся в библиотеках. Главная страница отображена на рисунке 5.1.

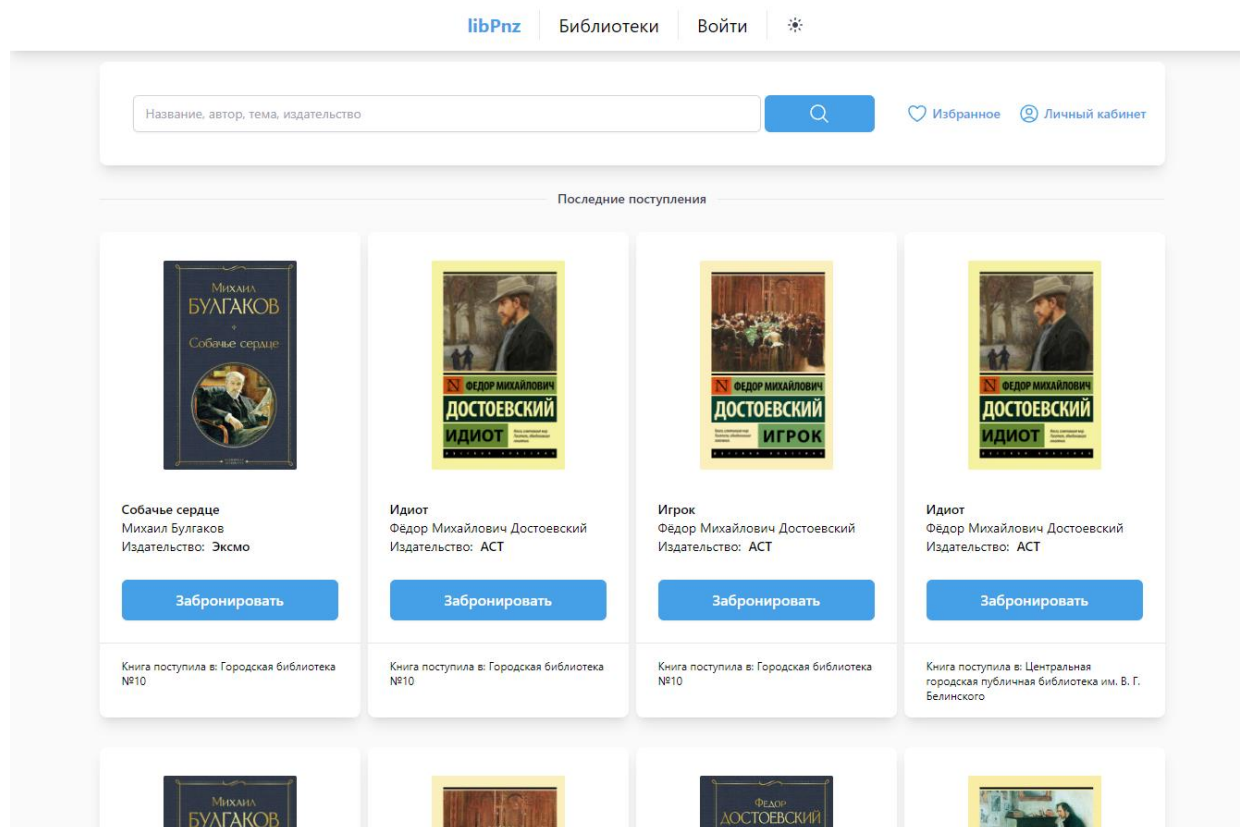


Рисунок 5.1 – Главная страница сайта

Ввод в строку поиска и нажатие на кнопку поиска осуществляет переход на страницу результатов поиска (рисунок 5.2). В результатах поиска отображается

массив книг, для каждой из которых отображается информация о доступности, в скольких библиотеках присутствует данная книга, если её нет в библиотеках, отображается красная надпись «Нет в наличии»

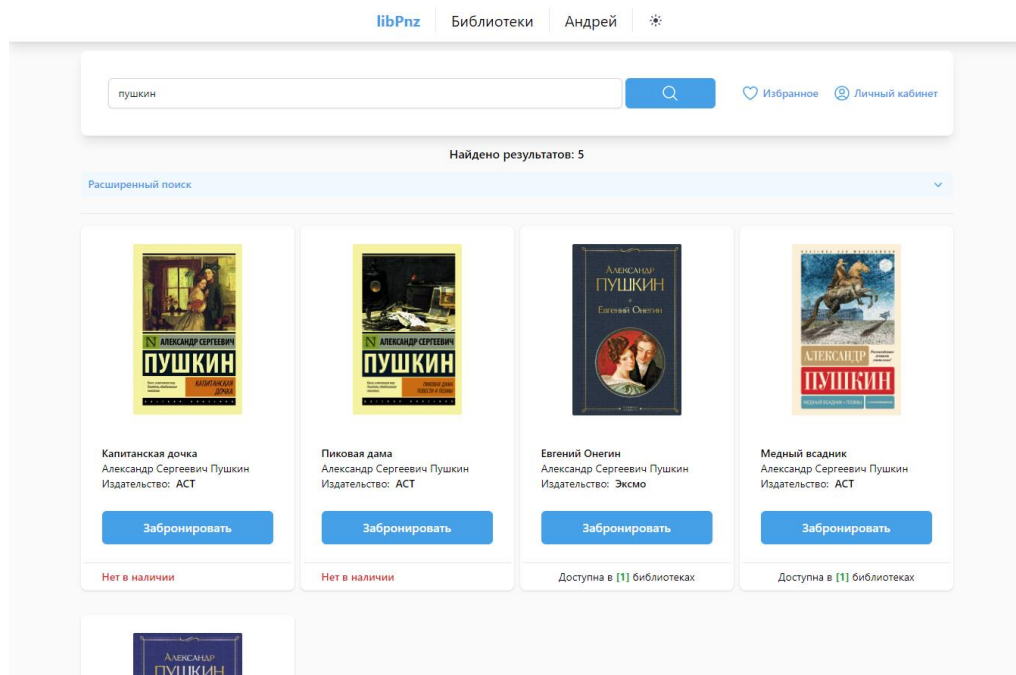


Рисунок 5.2 – Страница результатов поиска

В раскрывающемся меню отображается расширенный поиск, который позволяет к уже имеющемуся поисковому запросу добавить фильтры. Расширенный поиск представлен на рисунке 5.3

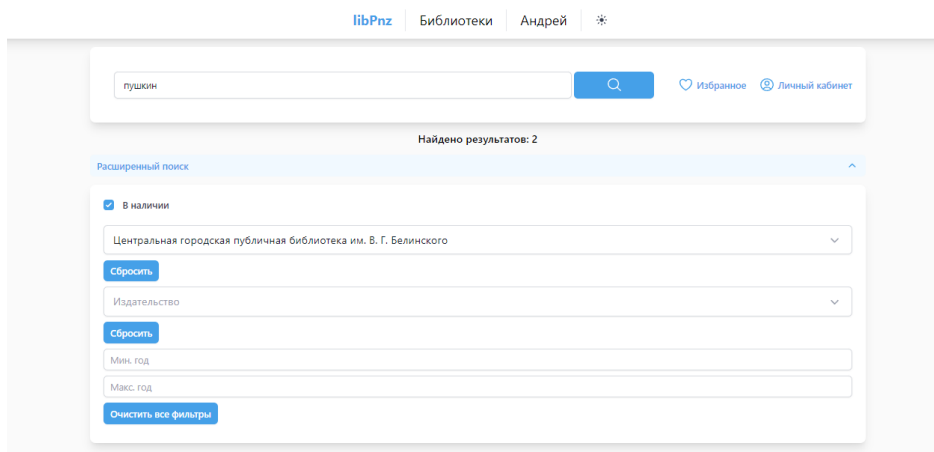


Рисунок 5.3 – Расширенный поиск

Для бронирования книги пользователь должен нажать на кнопку «Забронировать» в нижней части карточки книги. Произойдёт переход на

страницу книги. На странице книги отображается вся информация о книге, внизу страницы – карта, на которой маркерами отмечены библиотеки, в которых наличествует данная книга. Маркеры интерактивные, предоставляют информацию о библиотеке, и кнопку для бронирования именно в той библиотеке, на которой стоит маркер. Для выбора библиотеки, в которой будет забронирована книга, можно воспользоваться либо маркером на карте, либо выпадающим списком. Страница книги отражена на рисунке 5.4.

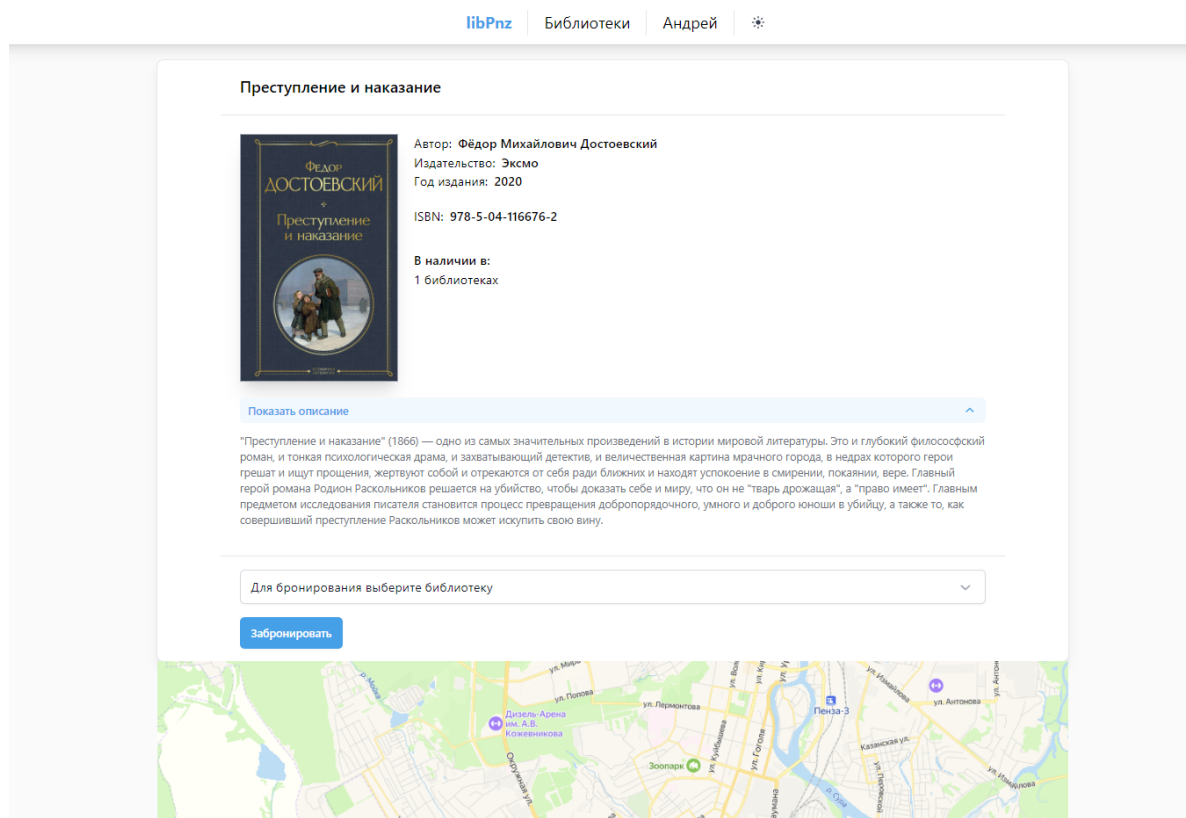


Рисунок 5.4 – Страница книги

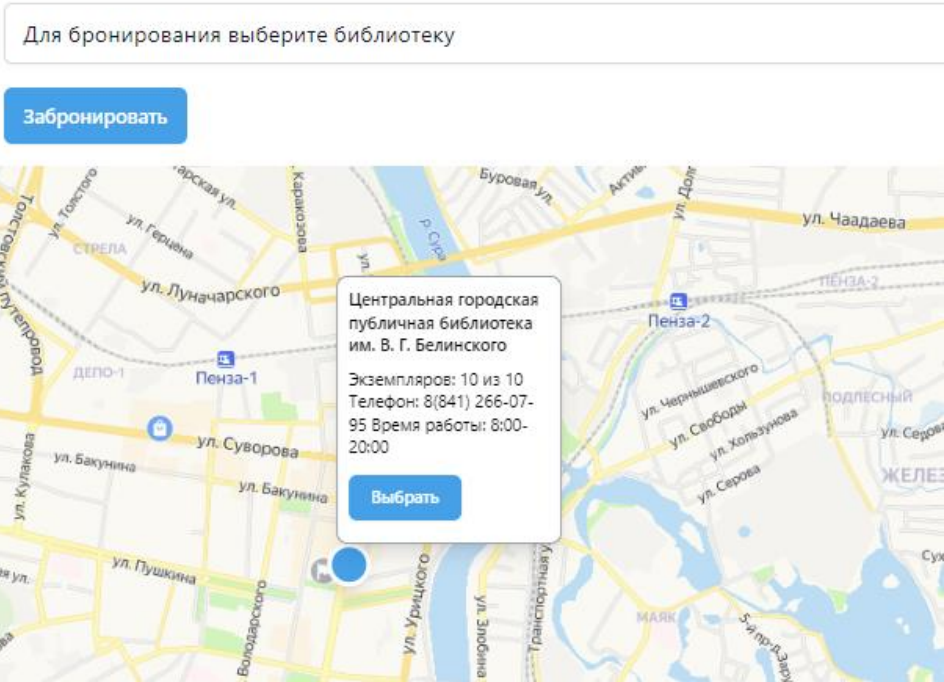


Рисунок 5.5 – Интерактивный маркер

После успешного бронирования книги книга будет отображаться в личном кабинете, при нажатии на раскрывающееся меню можно увидеть информацию о бронировании. Личный кабинет отображен на рисунке 5.5.

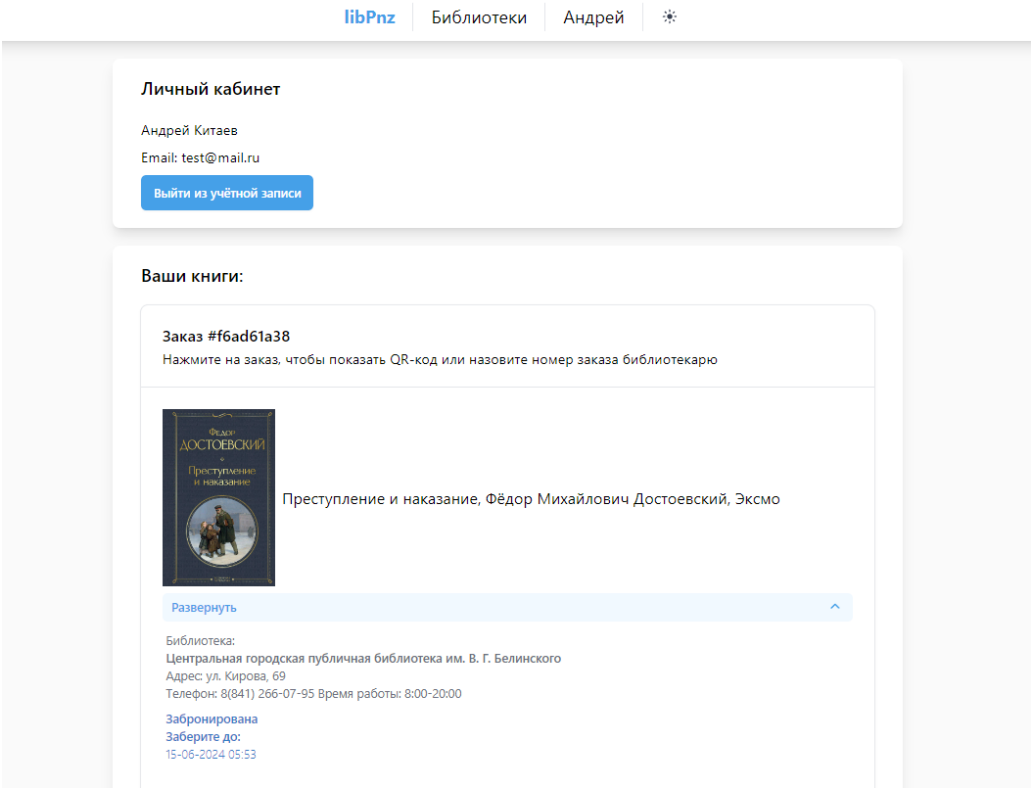


Рисунок 5.6 – Личный кабинет

При нажатии на забронированную книгу в личном кабинете на экране появится модальное окно с QR-кодом, содержащим уникальный идентификатор заказа (рисунок 5.7). Этот QR-код необходимо предоставить сотруднику библиотеки для получения книги.



Рисунок 5.7 – Модальное окно с QR-кодом

Личный кабинет администратора предоставляет сотрудникам библиотек функции добавления записей в БД, и функции обработки заказов. Личный кабинет администратора представлен на рисунке 5.8.

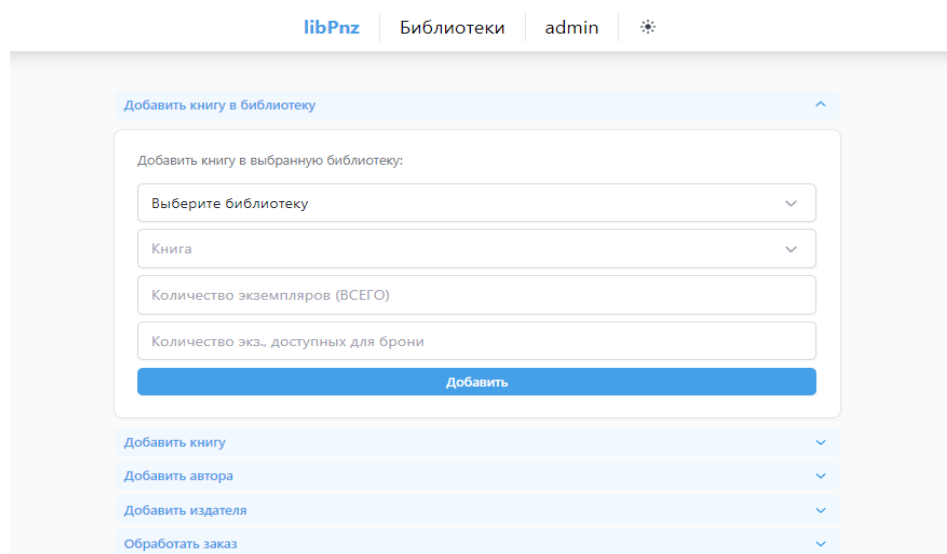


Рисунок 5.8 – Личный кабинет администратора

Для сканирования QR-кода нужно воспользоваться вкладкой «Обработать заказ». При нажатии на кнопку «Отсканировать» браузер запросит разрешение на использование веб-камеры, после чего включится камера и поток с неё будет отображаться в прямоугольном окне. Сканирование QR-кода представлено на рисунке 5.9.

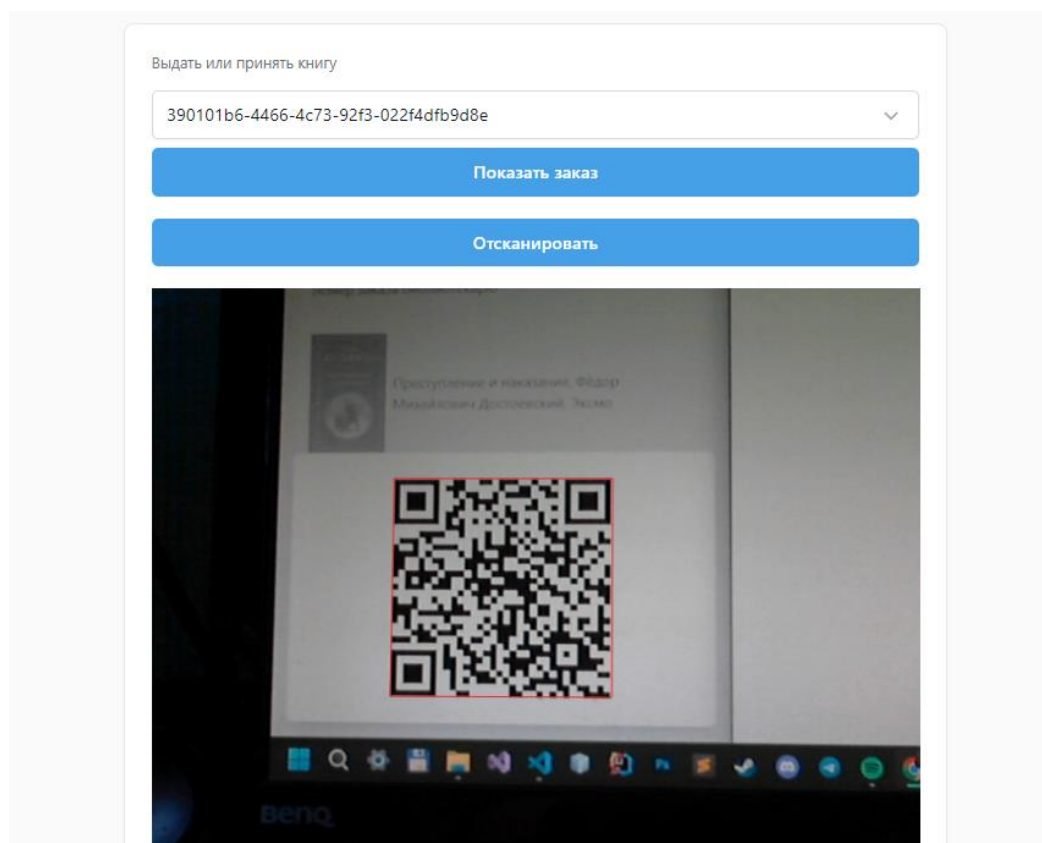


Рисунок 5.9 – Сканирование QR-кода

После нажатия на кнопку «Показать заказ» произойдёт переход на страницу заказа, на которой будет отображена вся информация о заказе, а также кнопки «Выдать» и «Принять», доступной будет только одна из них, в зависимости от текущего статуса заказа – выдан или нет. Страница заказа представлена на рисунке 5.10.

f6ad61a38



Автор: Фёдор Михайлович Достоевский
Издательство: Эксмо
Библиотека: Центральная городская публичная библиотека им. В. Г. Белинского
ISBN: 978-5-04-116676-2
Количество всего: 10
Забронированных / на руках: 1

Выдать

Принять

Рисунок 5.10 – Страница заказа

После успешной выдачи заказа, в личном кабинете пользователя обновится информация об этом заказе, и будет отображена информация о том, когда книга выдана, и когда её нужно вернуть, а также статус бронирования – просрочена книга или нет. Вид заказа после его получения представлен на рисунке 5.11.

Ваши книги:

Заказ #f6ad61a38
Нажмите на заказ, чтобы показать QR-код или назовите номер заказа библиотекарю



Преступление и наказание, Фёдор Михайлович Достоевский, Эксмо

Развернуть

Библиотека:
Центральная городская публичная библиотека им. В. Г. Белинского
Адрес ул. Кирова, 69
Телефон: 8(841) 266-07-95 Время работы: 8:00-20:00
Выдана:
10-06-2024 06:11
Вернуть до:
24-06-2024 06:11

Не просрочена

Рисунок 5.11 – Информация о полученном заказе

6 Безопасность жизнедеятельности

При взаимодействии с компьютером, необходимо соблюдать меры безопасности, чтобы не было неприятных последствий для здоровья после использования. Если пользователь ПК будет пренебрегать техникой безопасности, то проблемы со здоровьем будут неизбежны.

Искусственное освещение, используемое в помещениях, где сотрудники непосредственно связаны с персональными электронно-вычислительными машинами, обязано обеспечивать условия для безопасности зрения, а также соответствовать оптимальным условиям зрительное восприятие, какое бывает при естественном освещении. Для этого выделяют следующие требования к освещению на рабочих местах, оборудованных ПЭВМ:

- освещенность рабочего места должна быть порядка 300 – 500 лк. Запрещается работать в темном или полутёмном пространстве, а также в помещении с избыточным количеством света;

- искусственное освещение в помещениях для эксплуатации ПЭВМ должно быть распределено равномерно;

- естественный свет должен падать преимущественно слева от рабочего стола. Нельзя располагать оборудованное дисплеем рабочее место таким образом, чтобы в поле зрения оператора не попадали окна или осветительные приборы, а также они не должны находиться непосредственно за спиной пользователя;

- яркость бликов на экране ПЭВМ не должна превышать 40 кд / м² и яркость потолка не должна превышать 200 кд / м².

В качестве источников общего искусственного освещения лучше использовать люминесцентные лампы типа ЛБ с рассеивателями или экранирующими решетками, поскольку они создают равномерную освещенность путем рассеянного или отраженного распределения света и исключают блики на экране монитора и клавиатуре. Соотношение яркости экрана и непосредственно ближайшего окружения не должно превышать 3:1.

					ПГУ 1.090301.29.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

Кроме того, на рабочие места необходимо устанавливать локальные источники света – настольные лампы с регулируемой яркостью и изменением расположения плафона.

Эти требования необходимо учитывать, поскольку без правильного размещения источников света на рабочих местах зрение сотрудников может падать, что приводит к хронической переутомляемости и усталости, а также утрате заинтересованности в результатах работы.

Помимо условий относительно освещенности помещения существуют правила организации рабочего стола:

- расстояние между рабочими столами с видеомониторами должно быть не менее 2,0 м, а расстояние между боковыми поверхностями видеомониторов – не менее 1,2 м;

- экран видеомонитора должен находиться от глаз пользователя на расстоянии 600 - 700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов;

- конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работы;

- рабочее кресло должно быть подъемно-поворотным, регулируемым по высоте и углам наклона сиденья и спинки, а также расстоянию спинки от переднего края сиденья, при этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию;

- сиденье и спинка стула должны быть покрыты не электризующимися полумягкими материалами;

Кроме того, применение дисплеев с антибликовым, антирадиационным покрытием или защитных экранов поможет снизить вредное воздействие на здоровье человека при использовании персонального компьютера.

Для комфортной работы с персональным компьютером необходимо обеспечить оптимальные значения температуры воздуха в помещении (19-23 градуса цельсия), а также относительную влажность воздуха (55%). Помещение,

где находятся компьютеры, должно быть достаточно просторным и хорошо проветриваемым.

В связи с напряженностью трудового процесса у пользователей ПЭВМ рекомендуется организовать рабочую смену путем чередования работ с использованием ПЭВМ и без него. Продолжительность непрерывной работы не должна превышать 1 часа, а продолжительность перерыва – от 10 до 15 минут. При работе с ПЭВМ в ночную смену (с 22 до 6 ч.) продолжительность регламентированных перерывов следует увеличить на 30%.

Помимо этого, рекомендуется проводить упражнения на глаза, снижающих напряжение. Поскольку, смотря в монитор, люди реже моргают, слизистая оболочка не получает требуемого количества влаги и пересыхает.

Существует общие рекомендации, позволяющие снизить усталость зрительного аппарата:

- часто поморгайте в течение пары минут;
- открывайте и закрывайте глаза с периодичностью 3-5 секунд;
- Легкими движениями нажимайте на веки;
- переводите взгляд с удалённых объектов на близко расположенные предметы, фокусируя взгляд на протяжении пяти секунд.

Заключение

В ходе выполнения выпускной квалификационной работы реализовано клиент-серверное веб-приложение, для чего использован современный стек технологий в области веб-разработки, включающий фреймворки и библиотеки для создания пользовательских интерфейсов и серверной логики, исполняющейся в среде Node.

Результатом работы является клиент-серверное веб-приложение «Сервис электронного бронирования книжных изданий», а также пояснительная записка дипломного проекта, в которой представлены результаты тестирования приложения, описание разработки и обзор его функциональных возможностей.

Разработанное приложение предоставляет пользователю современный интерфейс, который позволяет пользователю получать информацию о доступности книжных изданий в библиотеках, бронировать книги и получать их в библиотеке посредством предоставления номера заказа сотруднику библиотек.

В ходе тестирования было подтверждено, что приложение соответствует всем поставленным в начале работы требованиям.

Задание на дипломное проектирование выполнено в полном объеме.

В дальнейшем приложение может быть усовершенствовано, путём расширения функционала сотрудников библиотек: добавлением возможности просмотра забронированных, выданных, просроченных книг, генерацией отчетов и статистики.

Список использованных источников

1. Стратегия развития библиотечного дела в Российской Федерации на период до 2030 года [Электронный ресурс]/Режим доступа: <https://sudact.ru/law/rasporiazhenie-pravitelstva-rf-ot-13032021-n-608-r/strategiia-razvitiia-bibliotechnogo-dela-v/?ysclid=llunyzgibe86730440>, свободный. (Дата обращения 25.04.2024).
2. Развитие информационных технологий и цифровая трансформация деятельности библиотек [Электронный ресурс]/Режим доступа: <https://sudact.ru/law/rasporiazhenie-pravitelstva-rf-ot-13032021-n-608-r/strategiia-razvitiia-bibliotechnogo-dela-v/iii/3/>, свободный. (Дата обращения 25.04.2024).
3. Основные показатели работы отрасли [Электронный ресурс]/Режим доступа: <https://stat.mkrf.ru/indicators/>
4. Как часто россияне ходят в библиотеку [Электронный ресурс]/Режим доступа: <https://journal.tinkoff.ru/library-stat>, свободный. (Дата обращения 25.04.2024).
5. Недостаточный уровень информатизации библиотек [Электронный ресурс]/Режим доступа: <https://sudact.ru/law/rasporiazhenie-pravitelstva-rf-ot-13032021-n-608-r/strategiia-razvitiia-bibliotechnogo-dela-v/i/2/nedostatochnyi-uroven-informatizatsii-bibliotek/>, свободный. (Дата обращения 25.04.2024)
6. Документация Python [Электронный ресурс]/Режим доступа: <https://www.python.org/about/apps/#web-and-internet-development>, свободный. (Дата обращения 29.04.2025)
7. Флэнаган Д. JavaScript. Карманный справочник. Сделайте веб-страницы интерактивными! / Перевод А.Г. Сысолюк. — Москва.: Издательский дом "Вильямс", 2015. — 320 с.
8. Бэнкс Алекс, Порселло Ева. React и Redux: функциональная веб-разработка. — СПб.: «Питер», 2018. — 336 с.

9. Бэнкс Алекс, Порселло Ева. GraphQL: язык запросов для современных веб-приложений. — СПб.: «Питер», 2019. — 240 с.

10. Nuxt.js: a Minimalist Framework for Creating Universal Vue.js Apps [Электронный ресурс]/Режим доступа:<https://www.sitepoint.com/nuxt-js-universal-vue-js/>, свободный. (Дата обращения 05.05.2024)

11. Getting Started with Server-Side Rendering Using Nuxt.js [Электронный ресурс]/Режим доступа: <https://alligator.io/vuejs/server-side-rendering-with-nuxtjs/>, свободный (Дата обращения 06.05.2024)

12. Документация Nuxt [Электронный ресурс]/Режим доступа: <https://nuxt.com/docs/guide>, свободный. (Дата обращения 12.05.2024)

13. Документация Nuxt [Электронный ресурс]/Режим доступа: <https://nuxt.com/docs/guide/concepts/server-engine>, свободный. (Дата обращения 12.05.2024)

14. Документация Prisma [Электронный ресурс]/Режим доступа: <https://www.prisma.io/docs/orm>, свободный. (Дата обращения 13.05.2024)

15. Документация Vue.js [Электронный ресурс]/Режим доступа: <https://vuejs.org/guide/scaling-up/sfc.html>, свободный. (Дата обращения 01.05.2024)

16. Документация Vue.js [Электронный ресурс]/Режим доступа: <https://vuejs.org/guide/reusability/composables>, свободный. (Дата обращения 01.05.2024)

17. Документация Nuxt [Электронный ресурс]/Режим доступа: <https://nuxt.com/docs/guide/directory-structure/composables>, свободный. (Дата обращения 03.05.2024)

18. Документация Nuxt [Электронный ресурс]/Режим доступа: <https://v2.nuxt.com/docs/directory-structure/pages/>, свободный. (Дата обращения 03.05.2024)

Приложение А

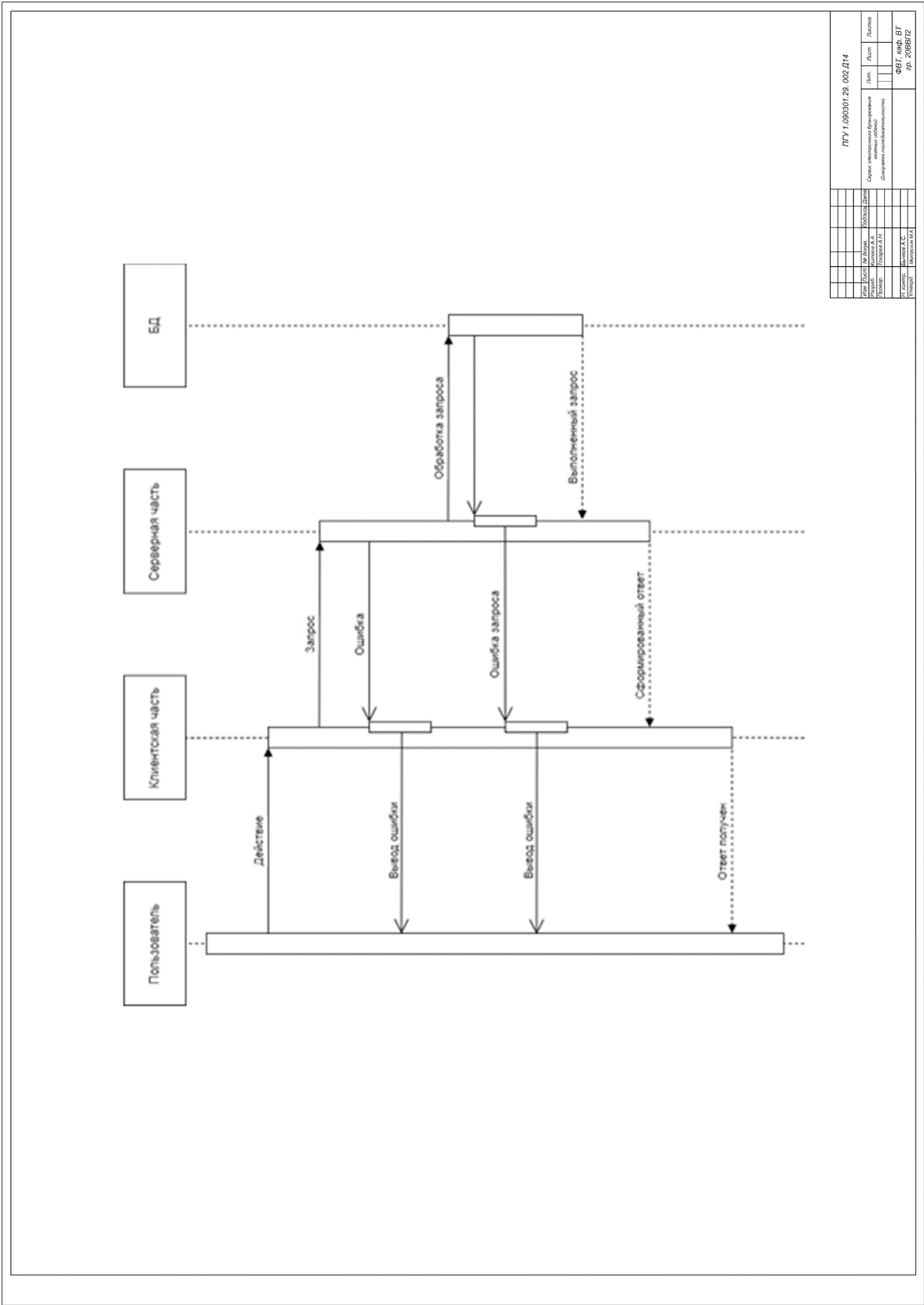
(Обязательное)

Клиент серверное приложение «Книжный интернет-магазин»

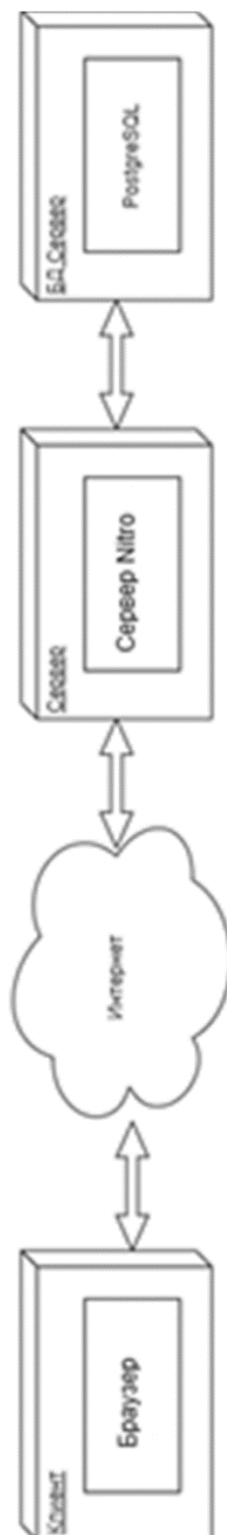
UML-диаграммы

					ПГУ 1.090301.29.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1–Диаграмма последовательностей



Приложение А.2—Диаграмма развертывания

[illegible]

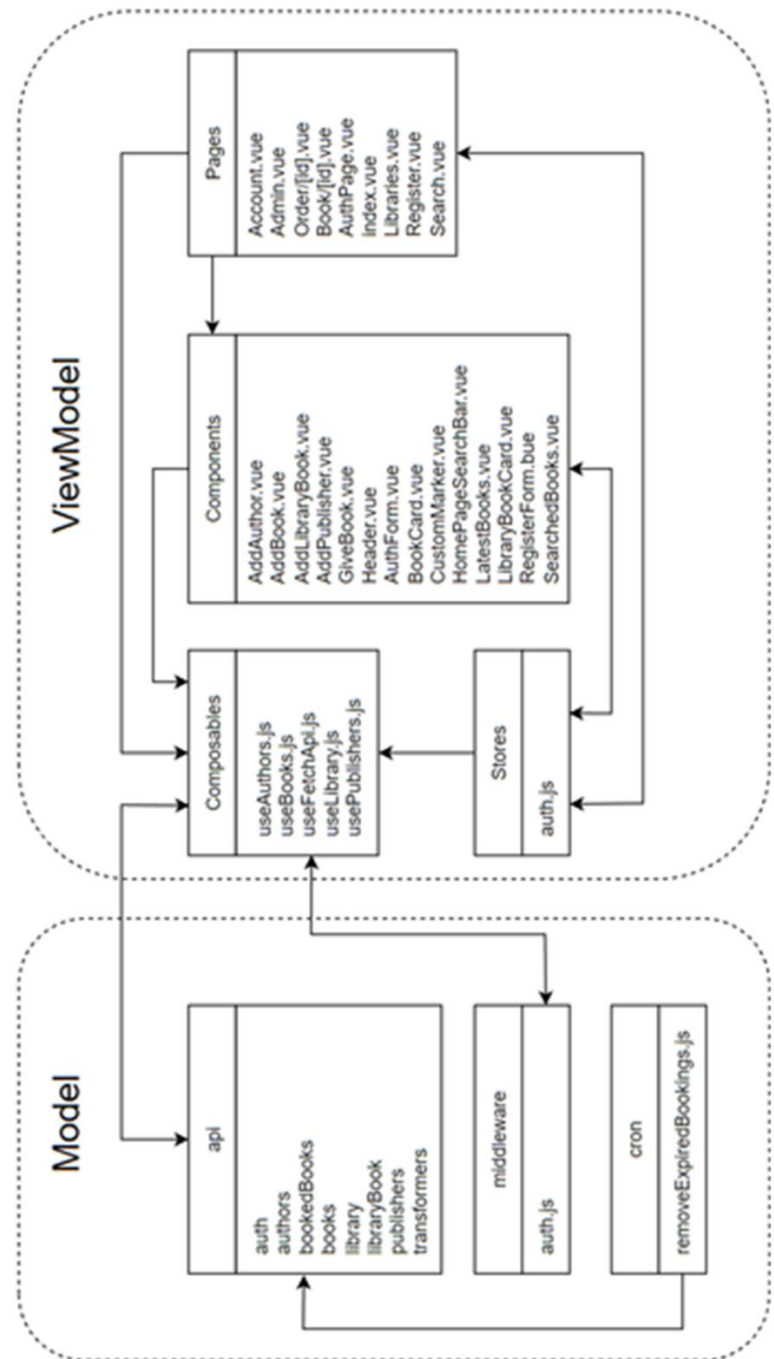
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.29.001 ПЗ

Лист

81

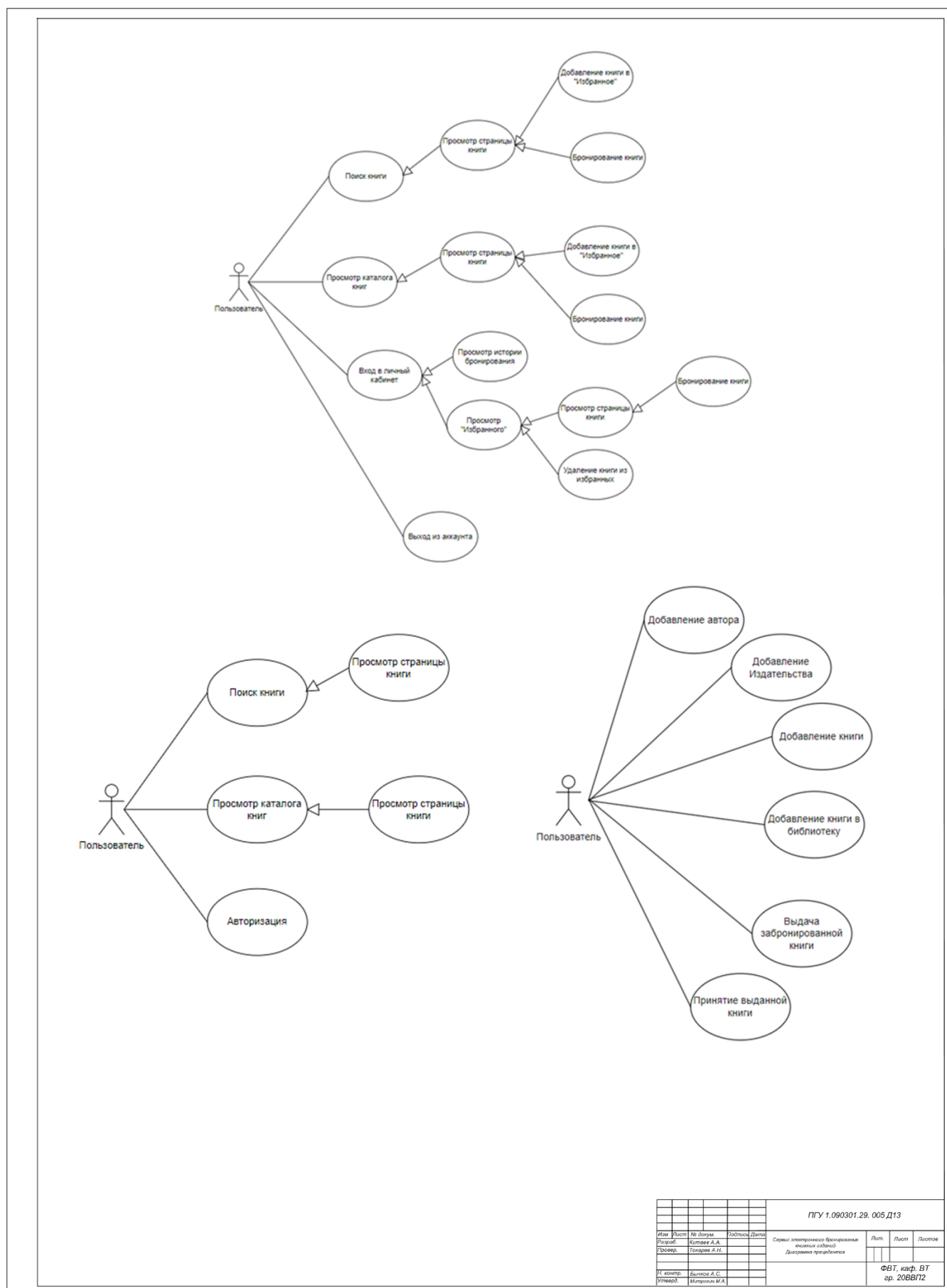
Приложение А.3–Диаграмма компонентов



ПГУ 1.090301.29.001.D18									
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.4—Диаграмма прецедентов



					ПГУ 1.090301.29. 005 Д13		
Изм.	Лист	№ докум.	Подпись	Дата	Сервис электронного бронирования личных копий Дополнительно предоставляется		
Разработчик	Колесов А.А.	Проверен	Голубев А.Н.				
Ил. контр.	Белкин А.С.	Утвержден	Матроскин М.А.		ФВТ, каф. ВТ гр. 20ВВ72		
					Лит.	Лист	Листов

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.29.001 ПЗ

Лист

83

Приложение Б

(Обязательное)

Сервис электронного бронирования книжных изданий

Листинг программы

Файл BookCard.vue

```
<template>
  <UCard class="ring-0 px-1 ">

    <div class="grid justify-items-center">
      <div class="shrink-0 w-20 sm:w-20 md:w-28 lg:w-36 md:h-48
lg:h-60 md:content-start content-center ">
        
      </div>
      <div class="my-2 h-20 lg:my-2 lg:h-24 w-full content-
start lg:content-center">
        <div class="font-semibold text-xs md:text-sm text-
ellipsis overflow-hidden">
          {{ props.book.title }}
        </div>
        <div class="flex text-xs md:text-sm text-ellipsis
overflow-hidden">
          {{ props.book.author }}
        </div>
        <div class="flex text-xs md:text-sm text-ellipsis
overflow-hidden">
          Издательство: <p class="ml-2 font-semibold"> {{
props.book.publisher }}</p>
        </div>
      </div>
    </div>
    <UButton size="xl" :to="bookUrl" class="w-full justify-
center">
      Забронировать
    </UButton>

    <template #footer v-if="props.book.libraryBookCount > 0">
      <div class="h-1 flex items-center justify-center">
        <p class="flex-none text-xs md:text-sm">Доступна в </p>
        <p class="flex-none text-green-600 font-bold mx-1 text-
xs md:text-sm">[{{ props.book.libraryBookCount }}]</p>
        <p class="flex-none text-xs md:text-sm">библиотеках</p>
      </div>
    </template>
    <template #footer v-if="props.book.libraryBookCount == 0">
      <div class="h-1 grid content-center font-semibold text-
red-600 text-xs md:text-sm">
        Нет в наличии
      </div>
    </template>
  </UCard>
</template>

<script setup>
const props = defineProps({
```

```

    book: {
      type: Object,
      required: true
    }
  })

  const bookUrl = computed(() => `/Book/${props.book.id}`)
</script>

```

Файл HomePageSearchBar.vue

```

<template>
  <div class="lg:grid lg:grid-cols-12 lg:p-5 lg:w-full shadow-
lg rounded-lg mt-3 mb-3 bg-white dark:bg-gray-900">
    <form @submit="handleSearch" class="justify-center grid
grid-cols-12 lg:col-span-9 lg:grid lg:grid-cols-12 ">
      <UInput v-model="search" :loading="loading" class="col-
span-10 lg:col-span-10 pl-4 py-4" size="lg"
placeholder="Название, автор, тема, издательство"
required />
      <UButton type="submit" @submit="handleSearch"
lg:label="Найти"
class="col-span-2 lg:col-span-2 my-4 ml-1 mr-4 content-
center justify-center">

        <template #trailing>
          <UIcon name="i-heroicons-magnifying-glass-solid"
class="w-6 h-6" />
        </template>
      </UButton>
    </form>
    <div class="grid grid-cols-2 pb-4 pt-2 mx-2 lg:m-0 lg:p-0
justify-items-center md:justify-items-center content-center lg:col-
span-3 rounded-lg ">
      <div class="text-sky-500 hover:text-sky-600 font-
semibold text-md flex content-center text-sm lg:text-md ">
        <UIcon name="i-heroicons-heart" class="w-5 h-5 lg:w-6
lg:h-6 mr-1 shrink-0" />
        <div class="content-center ">
          Избранное
        </div>
      </div>
      <NuxtLink v-if="authStore.authUser" to="Account" class="
text-sky-500 hover:text-sky-600 font-semibold flex content-center
text-sm lg:text-md mr-2 lg:mr-0">
        <UIcon name="i-heroicons-user-circle" class="w-5 h-5
lg:w-6 lg:h-6 mr-1 shrink-0" />
        <div class="content-center">Личный кабинет</div>
      </NuxtLink>
      <NuxtLink v-else to="AuthPage" class="text-sky-500
hover:text-sky-600 font-semibold flex content-center text-sm
lg:text-md mr-2 lg:mr-0">

```

```

        <UIcon name="i-heroicons-user-circle" class="w-5 h-5
lg:w-6 lg:h-6 mr-1 shrink-0" />
        <div class="content-center">Личный кабинет</div>
      </NuxtLink>
    </div>
  </div>
</template>
<script setup>
const authStore = useAuthStore()
const search = ref('')
const loading = ref(false)
const inStock = ref(false)
async function handleSearch(e) {
  e.preventDefault()

  useRouter().push({
    path: '/search',
    query: {
      q: search.value
    }
  })
}
</script>

```

Файл SearchedBooks.vue

```

<template>
  <div class="grid grid-cols-2 md:grid-cols-3 lg:grid-cols-4
gap-2">
    <BookCard v-for="book in props.books"
      :book="book"
      :key="book.id"
      class="text-sm lg:text-md mb-6 col-span-1" />
  </div>
</template>
<script setup>
const props = defineProps({
  books: {
    type: Array,
    required: true
  }
})
</script>
Файл useBooks.js
export default () => {
  const loading = ref(false)
  const error = ref(null)
  const fetchBooks = async (params = {}) => {
    if (loading.value) return
    loading.value = true
    error.value = null
  }
}

```

```

    try {
      const response = await useFetchApi('/api/books', {
        method: 'GET',
        params
      })
      return {
        books: response.books,
        total: response.total
      }
    }
    catch (err) {
      error.value = err.message
    }
    finally {
      loading.value = false
    }
  }
}

const fetchLibraryBooks = async () => {
  if (loading.value) return
  loading.value = true
  error.value = null
  try {
    const response = await
$fetch('/api/libraryBook/getLibBooks', {
      method: 'GET',
    })
    return {
      books: response.books
    }
  }
  catch (err) {
    error.value = err.message
  }
  finally {
    loading.value = false
  }
}

const fetchBookedBooks = async (params = {}) => {
  if (loading.value) return
  loading.value = true
  error.value = null
  try {
    const response = await useFetchApi('/api/bookedBooks', {
      method: 'GET',
      params
    })
    return {
      bookedBooks: response
    }
  }
  catch (err) {
    error.value = err.message
  }
}

```

```

        finally {
            loading.value = false
        }
    }
    const getOrderById = (orderId) => {
        return new Promise(async (resolve, reject) => {
            try {
                const response = await
useFetchApi(`/api/bookedBooks/${orderId}`)
                resolve(response)
            } catch (error) {
                reject(error.data)
            }
        })
    }
    const getBookById = (bookId) => {
        loading.value = true
        return new Promise(async (resolve, reject) => {
            try {
                const response = await
useFetchApi(`/api/books/${bookId}`)
                resolve(response)
            } catch (error) {
                reject(error.data)
            } finally {
                loading.value = false
            }
        })
    }
    const postBook = ({ publisherId, authorId, title, year, info,
ISBN, coverImage }) => {
        return new Promise(async (resolve, reject) => {
            try {
                await $fetch('/api/books', {
                    method: 'POST',
                    body: {
                        publisherId,
                        authorId,
                        title,
                        year,
                        info,
                        ISBN,
                        coverImage
                    }
                })
                resolve(true)
            } catch (error) {
                reject(error.data)
            }
        })
    }
}

```

```

const postLibraryBook = ({ bookId, libraryId, amount,
amountAvailable }) => {
  return new Promise(async (resolve, reject) => {
    try {
      await $fetch('/api/libraryBook/postLibBook', {
        method: 'POST',
        body: {
          bookId,
          libraryId,
          amount,
          amountAvailable
        }
      })
      resolve(true)
    } catch (error) {
      reject(error.data)
    }
  })
}

const postBookedBook = ({ libraryBookId, userId }) => {
  return new Promise(async (resolve, reject) => {
    try {
      await useFetchApi('/api/bookedBooks', {
        method: 'POST',
        body: {
          libraryBookId,
          userId
        }
      })
      resolve(true)
    } catch (error) {
      reject(error.data)
    }
  })
}

const giveOrder = ({ orderId }) => {
  return new Promise(async (resolve, reject) => {
    try {
      await useFetchApi('/api/bookedBooks', {
        method: 'PUT',
        body: {
          orderId
        }
      })
      resolve(true)
    } catch (error) {
      reject(error.data)
    }
  })
}

const takeOrder = ({ orderId }) => {
  return new Promise(async (resolve, reject) => {
    try {

```

```

        await useFetchApi('/api/bookedBooks', {
          method: 'DELETE',
          body: {
            orderId
          }
        })
        resolve(true)
      } catch (error) {
        reject(error.data)
      }
    })
  }
  const putLibraryBook = ({ libraryBookId, value }) => {
    return new Promise(async (resolve, reject) => {
      try {
        await $fetch('/api/libraryBook', {
          method: 'PUT',
          body: {
            libraryBookId,
            value
          }
        })
        resolve(true)
      } catch (error) {
        reject(error.data)
      }
    })
  }
  return {
    fetchBooks,
    loading,
    error,
    getBookById,
    fetchLibraryBooks,
    postBook,
    postLibraryBook,
    postBookedBook,
    fetchBookedBooks,
    giveOrder,
    putLibraryBook,
    takeOrder,
    getOrderById
  }
}

```

Файл Order/[id].vue

```

<template>
  <UContainer v-if="authStore.authUser &&
authStore.authUser.admin" class=" rounded-lg w-2/3 py-5">
    <UCard v-if="order" class="px-20 shadow-lg">
      <template #header>

```

					ПГУ 1.090301.29.001 ПЗ	Лист
						91
Изм.	Лист	№ докум.	Подпись	Дата		

```

        <div class="font-semibold text-xl">
            {{ order.orderCode }}
        </div>
    </template>
    <div class="flex">
        <div class="w-40 content-center border border-gray-400
shadow-xl">
            
        </div>
        <div class=" w-full pl-5 content-center">
            <div class="flex">
                Автор: <p class="ml-2 font-semibold"> {{
order.libraryBook.book.author.name }}</p>
            </div>
            <div class="flex">
                Издательство: <p class="ml-2 font-semibold"> {{
order.libraryBook.book.publisher.name }}</p>
            </div>
            <div class="flex">
                Библиотека: <p class="ml-2 font-semibold"> {{
order.libraryBook.library.name }}</p>
            </div>
            <div class="flex">
                ISBN: <p class="ml-2 font-semibold"> {{
order.libraryBook.book.ISBN }}</p>
            </div>
            <div class="flex">
                Количество всего: <p class="ml-2 font-semibold"> {{
order.libraryBook.amount }}</p>
            </div>
            <div class="flex">
                Забронированных / на руках: <p class="ml-2 font-
semibold"> {{ order.libraryBook.amount -
order.libraryBook.amountAvailable }}</p>
            </div>
        </div>
    </div>
    <template #footer>
        <UButton @click="handleGiveOrder" size="xl"
class="my-2" :loading="loading" :disabled="order.received">
            Выдать
        </UButton>
        <UButton @click="handleTakeOrder" size="xl"
class="my-2 mx-5" :loading="loading" :disabled="!order.received">
            Принять
        </UButton>
    <!-- </div> -->
    </template>
</UCard>
</UContainer>
</template>

```

```

<script setup>
const authStore = useAuthStore()
const loading = ref(false)
const { getOrderById, giveOrder, putLibraryBook, takeOrder } =
useBooks()
const route = useRoute([])
const orderId = computed(() => route.params.id)
const order = ref([])
const toast = useToast()
async function getOrder() {
  loading.value = true
  try {
    const response = await getOrderById(orderId.value)
    order.value = response.order
    console.log("order:", order.value)
  } catch (error) {
    console.log(error)
  }
  loading.value = false
}
await getOrder()
async function handleGiveOrder() {
  loading.value = true
  try {
    await giveOrder({
      orderId: order.value.id
    })
    toast.add({
      title: 'Успешно выдан заказ:',
      description: order.value.orderCode,
      icon: 'i-heroicons-check-circle',
      color: 'green'
    })
  } catch (error) {
    console.log(error)
    toast.add({
      title: 'Ошибка:',
      description: error,
      icon: 'i-heroicons-x-mark',
      color: 'red'
    })
  } finally {
    loading.value = false
    useRouter().push({
      path: '/Admin',
    })
  }
}
}
async function handleTakeOrder() {
  loading.value = true
  try {
    await putLibraryBook({
      libraryBookId: order.value.libraryBookId,

```

```

        value:1
      })
      await takeOrder({
        orderId: order.value.id
      })
      toast.add({
        title: 'Успешно принят заказ:',
        description: order.value.orderCode,
        icon: 'i-heroicons-check-circle',
        color: 'green'
      })
    } catch (error) {
      console.log(error)
    } finally {
      loading.value = false
      useRouter().push({
        path: '/Admin',
      })
    }
  }
}
</script>

```

Файл Book/[id].vue

```

<template>
  <UContainer v-if="loading" class=" rounded-lg lg:w-full py-
5">
    <UCard class="px-0 lg:px-20 shadow-lg">
      <template #header>
        <div class="font-semibold text-xl">
          <USkeleton class="h-8" />
        </div>
      </template>
      <div class="flex">
        <div class="w-48 lg:w-64 lg:content-start">
          <!--  -->
          <USkeleton class="h-48 lg:h-80" />
        </div>
        <div class=" w-full pl-5 space-y-3">
          <USkeleton class="h-4 w-80" />
          <USkeleton class="h-4 w-64" />
          <USkeleton class="h-4 w-64" />
          <USkeleton class="h-4 w-48" />
        </div>
      </div>
      <div class="mt-5">
        <USkeleton class="h-8 w-200" />
      </div>
      <div class="mt-10">
        <USkeleton class="h-12 w-200" />
      </div>
    </UCard>
  </UContainer>

```

```

        <template #footer>
        </template>
    </UCard>
</UContainer>
<UContainer v-if="book" class=" rounded-lg lg:w-full py-5">
    <UCard class="px-0 lg:px-20 shadow-lg">
        <template #header>
            <div class="font-semibold text-xl">
                {{ book.title }}
            </div>
        </template>
        <div class="flex">
            <div class="w-48 lg:w-64 lg:content-start">
                
            </div>
            <div class=" w-full pl-5 ">
                <div class="lg:flex">
                    Автор: <p class="lg:ml-2 font-
semibold"> {{ book.author }}</p>
                </div>
                <div class="lg:flex">
                    Издательство: <p class="lg:ml-2 font-
semibold"> {{ book.publisher }}</p>
                </div>
                <div class="lg:flex">
                    Год издания: <p class="lg:ml-2 font-
semibold"> {{ book.year }}</p>
                </div>
                <div class="lg:flex lg:mt-5">
                    ISBN: <p class="lg:ml-2 font-semibold">
{{ book.isbn }}</p>
                </div>

                <div class="mt-5 lg:mt-8">
                    <p class="font-semibold">
                        В наличии в:
                    </p>
                    <p>{{ libraryBooks.length }}
библиотеках</p>
                </div>
            </div>
        </div>
    </UContainer>
    <UAccordion :items="items" class="mt-5">
        <template #bookInfo>
            <div class="flex">
                {{ book.info }}
            </div>
        </template>
    </UAccordion>
    <template #footer>
        <USelectMenu v-model="selectedLib"
:options="libraryBooks"

```

```

placeholder="Для бронирования выберите
библиотеку" value-attribute="id"
option-attribute="libraryName" size="xl" />

<div v-if="authStore.authUser">
  <UButton @click="handleBooking"
:loading="loading" :disabled="loading" size="lg" class="mt-4">
    Забронировать
  </UButton>
</div>
</template>
</UCard>
<yandex-map v-model="map" :settings="{
  location: {
    center: [44.985491, 53.163541],
    zoom: 13,
  },
  theme: colorMode.value,
}" height="500px" :show-all-markers="true">
  <yandex-map-default-scheme-layer />
  <yandex-map-default-features-layer />
  <yandex-map-marker v-for="(marker, index) in
markers" :key="index" :settings="marker" offset="15">
    <!-- <div class="bg-white rounded-lg shadow-lg
text-xs content-center py-1 px-1 mb-3 border border-gray-600"><p
class="font-bold">{{ marker.library.amountAvailable }} экз.
</p></div>

    <button @click="isOpen = true"
      class="transform -translate-x-1/2 -
translate-y-1/2 text-white text-md font-semibold border-4 shadow-lg
border-white bg-sky-500 hover:bg-sky-600 rounded-full px-3 py-3">
    </button> -->
    <CustomMarker :library="marker.library"
:user="authStore.authUser"
      @selectLibrary="handleSelectLibrary" />
  </yandex-map-marker>
</yandex-map>
</UContainer>
<UModal v-model="isOpen">
  <!-- <UContainer class="p-4"> -->
  <div v-if="selectedLib && selectedLibInfo" class="px-7
py-5">
    <p>Книга будет забронирована в библиотеке:</p>
    <p class="font-semibold">{{
selectedLibInfo.libraryName }}</p>
    <p>По адресу: {{ selectedLibInfo.address }}</p>
    <p>Книга: {{ book.title }}</p>
    <p>Издательство: {{ book.publisher }}</p>
    <UButton :loading="loading" :disabled="loading"
@click="handleBooking"
      class="text-white text-md font-semibold bg-sky-
500 hover:bg-sky-600 rounded-lg px-4 py-2 mt-3 mb-2">
      Забронировать

```

```

        </UIButton>
    </div>
    <!-- </UContainer> -->
</UModal>
</template>
<script setup lang="ts">
    // import { YMapMarkerProps } from '@yandex/ymaps3-
types/imperative/YMapMarker';
    import type { YMap } from '@yandex/ymaps3-types';
    import { shallowRef } from 'vue';
    import { YandexMap, YandexMapDefaultSchemeLayer,
YandexMapDefaultFeaturesLayer, YandexMapMarker } from 'vue-yandex-
maps';
    const authStore = useAuthStore()
    const markers = ref([])
    const map = shallowRef<null | YMap>(null);
    const items = [{
        label: 'Показать описание',
        defaultOpen: false,
        slot: 'bookInfo'
    }]
    const colorMode = useColorMode()
    const isOpen = ref(false)
    const { getBookById, loading } = useBooks()
    const book = ref(null)
    const libraryBooks = ref(null)
    const selectedLib = ref()
    const selectedLibInfo = ref()
    const route = useRoute()
    const postLoading = ref(false)
    const toast = useToast()
    const error = ref()
    const bookId = computed(() => route.params.id)
    async function getBook() {
        try {
            const response = await getBookById(bookId.value)
            book.value = response.book
            libraryBooks.value = response.libs
            markers.value = response.libs.map(library => ({
                coordinates: [library.longitude, library.latitude],
                library
            }));
        } catch (error) {
            console.log(error)
        }
    }
    async function handleBooking() {
        console.log("booking user id", authStore.authUser.id)
        const { postBookedBook, putLibraryBook } = useBooks()
        postLoading.value = true
        isOpen.value = false
        error.value = null
        try {

```

```

        await postBookedBook({
            libraryBookId: selectedLib.value,
            userId: authStore.authUser.id,
        })
        toast.add({
            title: 'Вы забронировали книгу:',
            description: '' + book.value.title + '',
            icon: 'i-heroicons-check-circle',
        })
    } catch (err) {
        if (err.message) {
            toast.add({
                title: 'Ошибка:',
                description: '' + err.message + '',
                icon: 'i-heroicons-x-circle',
                color: 'orange'
            })
        } else {
            toast.add({
                title: 'Ошибка:',
                description: 'Что-то пошло не так :/',
                icon: 'i-heroicons-x-circle',
                color: 'red'
            })
        }
    }

    } finally {
        postLoading.value = false
    }
}

getBook()
function handleSelectLibrary(libraryId) {
    const selectedLibrary = libraryBooks.value.find(library =>
library.id === libraryId);
    selectedLib.value = selectedLibrary.id;
    selectedLibInfo.value = selectedLibrary;
    isOpen.value = true;
}
</script>

```

Файл index.vue

```

<template>
  <UContainer class="grid lg:grid">
    <HomePageSearchBar />
    <!-- <div class="grid grid-cols-2 md:grid-cols-3 lg:grid-
cols-4 gap-2">
      <USkeleton v-if="loading"
        class="text-sm lg:text-md mb-6 col-span-1" />
    </div> -->

    <LatestBooks :books="searchBooks" />

```

```

    <div>
      <div class="grid grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-2">
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-span-1 rounded-lg" />
      </div>
      <!-- <USkeleton v-if="loading" class="h-80 w-full rounded-lg" /> -->
    </div>
    <!-- <div v-if="loading">Загрузка</div> -->
  </UContainer>
</template>
<script setup>
const { fetchLibraryBooks, loading, error } = useBooks()
const searchBooks = ref([])
async function getBooks() {
  const books = await fetchLibraryBooks()
  if (books) {
    searchBooks.value = books.books
  }
}
getBooks()
</script>

```

Файл Search.vue

```

<template>
  <UContainer>
    <HomePageSearchBar />
  </UContainer>
</template>

```

```

<div class="mb-3 mt-2 font-semibold flex justify-
center">Найдено результатов: {{ totalPages }}</div>

<UAccordion :items="items" class="">
  <template #filters>
    <div class="bg-white dark:bg-gray-900 shadow-lg
rounded-lg p-5">
      <div class="pb-5">
        <UCheckbox v-model="filters.inStock" name="inStock"
label="В наличии" />
      </div>
      <USelectMenu v-model="filters.libraryId"
:options="libs" placeholder="Библиотека" value-attribute="id"
option-attribute="name" class="" size="xl" required
/>
      <UButton :loading="loading" :disabled="loading"
@click="resetLibraryFilter" class="justify-center my-2">
        Сбросить
      </UButton>
      <UInputMenu v-model="filters.publisherId"
:search="searchPublishers" :options="publishers" :loading="loading"
size="xl" placeholder="Издательство" option-
attribute="name" trailing value-attribute="id" required />
      <UButton :loading="loading" :disabled="loading"
@click="resetPublisherFilter" class="justify-center my-2">
        Сбросить
      </UButton>
      <UInput v-model.lazy="filters.minYear"
placeholder="Мин. год" required class="mb-2" />
      <UInput v-model.lazy="filters.maxYear"
placeholder="Макс. год" required />
      <UButton :loading="loading" :disabled="loading"
@click="clearAllFilters" class="justify-center my-2">
        Очистить все фильтры
      </UButton>
    </div>
  </template>
</UAccordion>
<UDivider class="my-4"></UDivider>
<div class="grid grid-cols-2 md:grid-cols-3 lg:grid-cols-4
gap-2">
  <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
  <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
  <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
  <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
  <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
  <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />

```

```

        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
        <USkeleton v-if="loading" class="h-[380px] w-70 mb-6 col-
span-1 rounded-lg" />
    </div>
    <SearchedBooks :books="searchBooks" />
    <div class="flex justify-center mb-20" v-if="searchBooks">
        <UPagination v-model="page" :page-count="pageSize"
:total="totalPages" :active-button="{ color: 'sky' }" />
    </div>
</UContainer>
</template>
<script setup>
const { fetchBooks, loading, error } = useBooks()
const { fetchLibs } = useLibrary()
const searchBooks = ref([])
const route = useRoute()
const searchQuery = computed(() => route.query.q)
const filters = ref({
    inStock: null,
    libraryId: null,
    publisherId: null,
    minYear: null,
    maxYear: null,
})
const libs = ref([])
const publishers = ref([])
const items = [{
    label: 'Расширенный поиск',
    defaultOpen: false,
    slot: 'filters'
}]
const page = ref(route.query.page ? route.query.page : 1)
const totalPages = ref()
const pageSize = 12
watch([searchQuery, page, filters.value], async () => {
    console.log("watch")
    console.log(filters.value)
    getBooks()
})
async function getBooks() {
    const skip = (page.value - 1) * pageSize
    const params = {
        query: searchQuery.value,
        skip: skip,
    }

```

```

        take: pageSize,
        ...filters.value
    }
    const books = await fetchBooks(params)
    if (books) {
        searchBooks.value = books.books
        totalPages.value = books.total
    }
}
getBooks()
const getLibs = async (q) => {
    if (q === '') return []
    const response = await fetchLibs()
    libs.value = response.libraries
    return libs.value
}
const searchPublishers = async (q) => {
    if (q === '') return []
    const response = await $fetch('/api/publishers/', {
        method: 'GET',
        query: {
            query: q
        }
    })
    publishers.value = response
    return publishers.value
}
getLibs()
function resetLibraryFilter() {
    filters.value.libraryId = null
}
function resetPublisherFilter() {
    filters.value.publisherId = null
}
function clearAllFilters() {
    filters.value.publisherId = null
    filters.value.libraryId = null
    filters.value.inStock = false
    filters.value.libraryId = null
    filters.value.minYear = null
    filters.value.maxYear = null
}
</script>

```

Файл login.post.js

```

import { getUserByEmail } from "~/server/db/users"
import bcrypt from "bcrypt"
import { generateTokens, sendRefreshToken } from
~/server/utils/jwt.js"
import { createRefreshToken } from "~/server/db/refreshTokens"

```

```

import { userTransformer } from
"~/server/api/transformers/user"
import { sendError } from "h3"
export default defineEventHandler(async (event) => {
  const body = await readBody(event)
  const {email, password} = body
  if (!email || !password) {
    return sendError(event, createError({
      statusCode:400,
      statusMessage: 'Invalid params'
    }))
  }
  const user = await getUserByEmail(email)
  if(!user) {
    throw createError({
      statusCode:400,
      statusMessage:'Username or password is invalid',
      message:'Пользователь не зарегистрирован'
    })
  }
  const doesThePasswordMatch = await bcrypt.compare(password,
user.password)

  if(!doesThePasswordMatch) {
    throw createError({
      statusCode:400,
      statusMessage:'Username or password is invalid',
      message:'Неверный пароль или email'
    })
  }
  const {accessToken, refreshToken} = generateTokens(user)
  await createRefreshToken({
    token: refreshToken,
    userId: user.id
  })
  sendRefreshToken(event, refreshToken)
  return {
    accessToken: accessToken,
    user: userTransformer(user)
  }
})

```

Файл refresh.get.js

```

import { getRefreshTokenByToken } from
"~/server/db/refreshTokens";
import { getUserId } from "~/server/db/users";
import { decodeRefreshToken, generateTokens } from
"../../utils/jwt";
export default defineEventHandler(async (event) => {
  const cookies = parseCookies(event)

```

```

const refreshToken = cookies?.refresh_token
console.log("refreshToken:", refreshToken)
if (!refreshToken) {
  return sendError(event, createError({
    statusCode: 401,
    statusMessage: "Refresh token is invalid"
  }))
}
const rToken = await getRefreshTokenByToken(refreshToken)
if (!rToken) {
  return sendError(event, createError({
    statusCode: 401,
    statusMessage: "Refresh token is invalid 2"
  }))
}
const token = decodeRefreshToken(refreshToken)
try {
  const user = await getUserById(token.userId)
  const { accessToken } = generateTokens(user)
  return {
    access_token: accessToken
  }
} catch (error) {
  return sendError(event, createError({
    statusCode: 500,
    statusMessage: "Something went wrong"
  }))
}
})

```

Файл authors.post.js

```

import { createAuthor } from "~/server/db/author"
export default defineEventHandler(async (event) => {
  const body = await readBody(event)
  const { name, info } = body
  const authorData = {
    name,
    info
  }
  console.log("author add")
  const author = await createAuthor(authorData)
  return {
    author: author
  }
})

```

Файл bookedBooks/[id].get.js

```

import { getBookedBooks, getOrderById } from
"~/server/db/booking"

```

					ПГУ 1.090301.29.001 ПЗ	Лист
						104
Изм.	Лист	№ докум.	Подпись	Дата		

```

export default defineEventHandler(async (event) => {
  const { id } = event.context.params
  try {
    const order = await getOrderById(id)
    return {
      order: order
    }
  }
  catch (error) {
    throw createError({
      statusCode: 500,
      statusMessage: 'Error finding orders',
    })
  }
})

```

Файл bookedBooks.delete.js

```

import { deleteBookedBook } from "~/server/db/booking"
import { sendError } from "h3"
export default defineEventHandler(async (event) => {
  const body = await readBody(event)
  const { orderId } = body
  const prismaQuery = {
    where: {
      id: orderId,
    },
  }
  const newBookedBook = await deleteBookedBook(prismaQuery)
  return {
    bookedBook: newBookedBook
  }
})

```

Файл bookedBooks.get.js

```

import {getBookedBooks} from "~/server/db/booking"
export default defineEventHandler(async (event) => {
  const query = getQuery(event)
  if (!query.query) {
    throw createError({
      statusCode: 400,
      statusMessage: 'Query string should not be empty',
    })
  }
  let prismaQuery = {
    include: {
      libraryBook: {
        include: {
          library: true,
          book: {

```

```

        include: {
          author:true,
          publisher:true
        }
      }
    },
  },
  where: {
    OR: [
      {
        orderCode: {
          search: query.query
        }
      },
      {
        userId: {
          search: query.query
        }
      }
    ]
  }
}
}
try {
  const orders = await getBookedBooks(prismaQuery)
  return orders
}
catch (error) {
  throw createError({
    statusCode: 500,
    statusMessage: 'Error finding orders',
  })
}
})

```

Файл bookedBooks.post.js

```

import { createBookedBook, getBookedBooks, generateOrderCode }
from "~/server/db/booking"
import { getLibraryBookById } from "~/server/db/book"
export default defineEventHandler(async (event) => {
  const body = await readBody(event)
  const { libraryBookId, userId } = body
  const dueDate = new Date()
  dueDate.setDate(dueDate.getDate() + 5)
  const bookData = {
    libraryBookId,
    userId,
    received: false,
    orderCode: '',
    booked_due_date: dueDate
  }

```

```

    }
    if (!bookData.libraryBookId) {
      throw createError({
        statusCode:400,
        statusMessage:'No available books',
        message:'Сначала выберите библиотеку',
      })
    }
    const libraryBook = await
getLibraryBookById(bookData.libraryBookId)
    if(libraryBook.amountAvailable == 0) {
      throw createError({
        statusCode:400,
        statusMessage:'No available books',
        message:'Книги нет в наличии'
      })
    }
    const query = {
      where: {
        AND: [
          {libraryBookId: bookData.libraryBookId},
          {userId: bookData.userId}
        ]
      }
    }
    if(await getBookedBooks(query) != 0) {
      throw createError({
        statusCode:400,
        statusMessage:'You cant order the same book twice',
        message:'Вы не можете забронировать одну книгу дважды'
      })
    } else {
      bookData.orderCode = generateOrderCode(userId,
libraryBookId)
      const bookedBook = await createBookedBook(bookData)
      return {
        book: bookedBook
      }
    }
  })
})

```

Файл bookedBooks.put.js

```

import { putBookedBook } from "~/server/db/booking"
import { sendError } from "h3"
export default defineEventHandler(async (event) => {
  const body = await readBody(event)
  const { orderId } = body
  const dueDate = new Date()
  dueDate.setDate(dueDate.getDate() + 14)
  const prismaQuery = {
    where: {

```

```

        id: orderId,
      },
      data: {
        received: true,
        received_at: new Date(),
        due_date: dueDate
      }
    }
  }
  const newBookedBook = await putBookedBook(prismaQuery)
  return {
    bookedBook: newBookedBook
  }
})

```

Файл books/[id].get.js

```

import { getBookById, getLibraryBooks } from "~/server/db/book"
import { bookTransformer } from "../transformers/book"
import { libTransformer } from "../transformers/library"
import { libraryBookTransformer } from
"../transformers/libraryBook"
export default defineEventHandler(async (event) => {
  const { id } = event.context.params
  console.log(id)
  const book = await getBookById(id)
  const query = {
    select: {
      id: true,
      library: true,
      amount: true,
      amountAvailable: true
    },
    where: {
      book: {
        id: {
          search: id
        }
      }
    },
  },
  const LB = await getLibraryBooks(query)
  return {
    book: bookTransformer(book),
    libs: LB.map(libTransformer)
  }
})

```

Файл books.get.js

```

import { getBooks, getBooksCount, getTotalAvailableBooks } from
"~/server/db/book";

```

					ПГУ 1.090301.29.001 ПЗ	Лист
						108
Изм.	Лист	№ докум.	Подпись	Дата		

```

import { bookTransformer } from "../transformers/book";
export default defineEventHandler(async (event) => {
  const { query, skip, take, publisherId, inStock, libraryId,
minYear, maxYear } = getQuery(event);
  const searchConditions = [];
  if (query) {
    const searchTerms = query.trim().split(" ");
    const checkTermResults = async (term) => {
      const condition = {
        OR: [
          {
            author: {
              name: {
                search: term,
                mode: 'insensitive'
              }
            },
            {
              title: {
                search: term,
                mode: 'insensitive'
              }
            },
            {
              publisher: {
                name: {
                  search: term,
                  mode: 'insensitive'
                }
              }
            }
          ]
        };
      const count = await getBooksCount({ where:
condition });
      return count > 0;
    };
    const validTerms = [];
    for (const term of searchTerms) {
      if (await checkTermResults(term)) {
        validTerms.push(term);
      }
    }
    if (validTerms.length === 0) {
      return {
        books: [],
        total: 0
      };
    }
    if (validTerms.length > 0) {
      searchConditions.push({
        AND: validTerms.map((term) => ({

```

```

        OR: [
            {
                author: {
                    name: {
                        search: term,
                        mode: 'insensitive'
                    }
                },
            },
            {
                title: {
                    search: term,
                    mode: 'insensitive'
                },
            },
            {
                publisher: {
                    name: {
                        search: term,
                        mode: 'insensitive'
                    }
                },
            },
        ]
    })))
    });
}

console.log("Backend inStock:", inStock)
console.log("Backend libraryId:", libraryId)
console.log("ALL:", query, skip, take, publisherId,
inStock, libraryId, minYear, maxYear)
if (inStock === 'true') {
    searchConditions.push({
        LibraryBook: {
            some: {
                amountAvailable: {
                    gt: 0
                }
            }
        }
    });
}
if (libraryId) {
    searchConditions.push({
        LibraryBook: {
            some: {
                libraryId: libraryId
            }
        }
    });
}
if (publisherId) {

```

```

        searchConditions.push({
          publisher: {
            id: publisherId,
          }
        });
      }
    }
    if (minYear || maxYear) {
      const yearCondition = {};
      if (minYear) {
        yearCondition.gte = minYear;
      }
      if (maxYear) {
        yearCondition.lte = maxYear;
      }
      searchConditions.push({
        year: yearCondition
      });
    }
    let prismaQuery = {
      include: {
        author: true,
        publisher: true,
        LibraryBook: true, // Для фильтрации по количеству
        доступных книг
        _count: {
          select: {
            LibraryBook: true
          }
        }
      },
      skip: +skip,
      take: +take,
      where: searchConditions.length > 0 ? { AND:
searchConditions } : {}
    };
    const books = await getBooks(prismaQuery);
    const total = await getBooksCount({
      where: prismaQuery.where
    });

    const totalAvailable = await
getTotalAvailableBooks(prismaQuery.where);
    return {
      books: books.map(bookTransformer),
      total: total,
      totalAvailable: totalAvailable
    };
  });
}

```

Файл books.post.js

```
import { createBook, getBookByInfo } from "~/server/db/book"
```

					ПГУ 1.090301.29.001 ПЗ	Лист
						111
Изм.	Лист	№ докум.	Подпись	Дата		

```

export default defineEventHandler(async (event) => {
  const body = await readBody(event)
  const { publisherId, authorId, title, year, info, ISBN,
coverImage } = body
  const bookData = {
    publisherId,
    authorId,
    title,
    year,
    info,
    ISBN,
    coverImage
  }
  if (!publisherId || !authorId || !title || !year || !info
|| !ISBN || !coverImage) {
    throw createError({
      statusCode:400,
      statusMessage: 'Invalid params',
      message: 'Неполная информация :('
    })
  }
  const checkBook = await getBookByInfo(bookData)
  if(checkBook) {
    throw createError({
      statusCode:400,
      statusMessage: 'Book already exists',
      message: 'Книга уже существует'
    })
  }
  const book = await createBook(bookData)
  return {
    book: book
  }
})

```

Файл getLibBooks.get.js

```

import { getLibraryBooks } from "~/server/db/book"
import { libraryBookTransformer } from
"../transformers/libraryBook"
export default defineEventHandler(async () => {
  let prismaQuery = {
    skip:0,
    take:8,
    include: {
      book: {
        include: {
          author: true,
          publisher: true
        }
      },
    },
  },

```

```

        library: true
      },
      orderBy: [
        {
          created_at: 'desc'
        }
      ]
    }
  }
  const books = await getLibraryBooks(prismaQuery)
  return {
    books: books.map(libraryBookTransformer)
  }
})

```

Файл libraryBook.put.js

```

import { getLibraryBookById, putLibraryBook } from
"~/server/db/book"
import { sendError } from "h3"
export default defineEventHandler(async (event) => {
  const body = await readBody(event)
  const { libraryBookId, value } = body
  const libraryBook = await getLibraryBookById(libraryBookId)
  if (libraryBook.amountAvailable == 0 && value == -1) {
    return sendError(event, createError({
      statusCode: 400,
      statusMessage: 'Book is not available'
    }))
  }
  const newAmountAvailable = libraryBook.amountAvailable +
value
  const prismaQuery = {
    where: {
      id: libraryBookId,
    },
    data: {
      amountAvailable: newAmountAvailable
    }
  }
  const newLibraryBook = await putLibraryBook(prismaQuery)
  return {
    libraryBook: newLibraryBook
  }
})

```

Файл removeExpiredBookings.js

```

import cron from 'node-cron'
import { PrismaClient } from '@prisma/client'
const prisma = new PrismaClient()

```

```

const removeExpiredBookings = async () => {
  const now = new Date()
  const expiredBookings = await prisma.bookedBooks.findMany({
    where: {
      booked_due_date: {
        lt: now,
      },
      received: false,
    },
  })
  for (const booking of expiredBookings) {
    await prisma.$transaction([
      prisma.libraryBook.update({
        where: { id: booking.libraryBookId },
        data: { amountAvailable: { increment: 1 } },
      }),
      prisma.bookedBooks.delete({
        where: { id: booking.id },
      }),
    ])
  }
}
const scheduleCronJobs = () => {
  cron.schedule('0 0 * * *', () => {
    removeExpiredBookings()
      .then(() => {
        console.log('Expired bookings removed')
      })
      .catch((error) => {
        console.error('Error removing expired bookings:',
error)
      })
  })
}
export { scheduleCronJobs }

```

Файл cron.js

```

import { scheduleCronJobs } from './removeExpiredBookings'
export const initCronJobs = () => {
  if (process.env.NODE_ENV !== 'production') {
    console.log('Scheduling cron jobs')
    scheduleCronJobs()
  }
}

```

Файл book.js

```

import { prisma } from ".";
export const createBook = (bookData) => {
  return prisma.books.create({

```

```

        data: bookData,
      })
    }
    export const createLibraryBook = (bookData) => {
      return prisma.libraryBook.create({
        data: bookData
      })
    }
    export const getBooks = (params = {}) => {
      return prisma.books.findMany({
        ...params
      })
    }
    export const getBooksCount = (params = {}) => {
      return prisma.books.count({
        ...params
      });
    };
    export const getLibraryBooks = (params = {}) => {
      return prisma.libraryBook.findMany({
        ...params
      })
    }
    export const getBookByInfo = (bookData) => {
      return prisma.books.findFirst({
        where: {
          AND: [
            {
              title: bookData.title
            },
            {
              author: {
                id: bookData.authorId
              }
            },
            {
              publisher: {
                id: bookData.publisherId
              }
            }
          ]
        }
      })
    }
    export const getLibraryBookByInfo = (bookData) => {
      return prisma.libraryBook.findFirst({
        where: {
          AND: [
            {
              bookId: bookData.bookId
            },
            {
              libraryId: bookData.libraryId
            }
          ]
        }
      })
    }
  }
}

```

```

        },
      ],
    },
  })
}
export const getBookById = (bookId) => {
  return prisma.books.findUnique({
    where: {
      id: bookId
    },
    include: {
      author: true,
      publisher: true,
      _count: {
        select: {
          LibraryBook: true
        }
      }
    },
  })
}
export const getLibraryBookById = (bookId) => {
  try {
    return prisma.libraryBook.findUnique({
      where: {
        id: bookId
      }
    })
  } catch (err) {
  }
}
export const putLibraryBook = (params) => {
  return prisma.libraryBook.update({
    ...params
  })
}
export const getTotalAvailableBooks = async (where) => {
  const result = await prisma.libraryBook.aggregate({
    _sum: {
      amountAvailable: true
    },
    where: {
      book: where
    }
  });
  return result._sum.amountAvailable || 0;
};
export const getBooksWithAvailability = async (params = {}) =>
{
  const books = await prisma.books.findMany({
    ...params,
    include: {
      author: true,

```

```

        publisher: true,
        LibraryBook: {
          select: {
            amountAvailable: true
          }
        },
        _count: {
          select: {
            LibraryBook: true
          }
        }
      }
    });
    let totalAvailable = 0;
    books.forEach(book => {
      book.LibraryBook.forEach(libraryBook => {
        totalAvailable += libraryBook.amountAvailable;
      });
    });
    return {
      books,
      totalAvailable
    };
  };
};

```

Файл booking.js

```

import { prisma } from ".";
import crypto from 'crypto'
export const createBookedBook = (bookData) => {
  return prisma.$transaction([
    prisma.bookedBooks.create({
      data: bookData,
    }),
    prisma.libraryBook.update({
      where: {
        id: bookData.libraryBookId,
      },
      data: {
        amountAvailable: { decrement: 1 }
      }
    })
  ])
}
export const getBookedBooks = (params = {}) => {
  return prisma.bookedBooks.findMany({
    ...params
  })
}
export const generateOrderCode = (userUuid, bookUuid) => {
  const combined = userUuid + bookUuid

```

```

    const hash =
crypto.createHash('sha256').update(combined).digest('hex')
    return hash.substring(0, 9) // Первые 8 символов хэша
  }
  export const putBookedBook = (params) => {
    return prisma.bookedBooks.update({
      ...params
    })
  }
  export const deleteBookedBook = (params) => {
    return prisma.bookedBooks.delete({
      ...params
    })
  }
  export const getOrderByIdAndBookId = (params) => {
    return prisma.bookedBooks.findUnique({
      ...params
    })
  }
  export const getOrderById = (orderId) => {
    return prisma.bookedBooks.findUnique({
      where: {
        id: orderId
      },
      include: {
        libraryBook: {
          include: {
            book: {
              include: {
                author: true,
                publisher: true
              }
            },
            library: true
          }
        }
      }
    })
  }
}

```

Файл refreshTokens.js

```

import { prisma } from "."
export const createRefreshToken = (refreshToken) => {
  return prisma.refreshToken.create({
    data: refreshToken
  })
}
export const getRefreshTokenByToken = (token) => {
  return prisma.refreshToken.findUnique({
    where: {
      token
    }
  })
}

```

```

    }
  })
}
export const removeRefreshToken = (token) => {
  return prisma.refreshToken.delete({
    where: {
      token: token
    }
  })
}
}

```

Файл app.vue

```

<template>
  <UContainer class="min-w-full max-w-full grid justify-items-
center bg-white dark:bg-gray-900 shadow-lg">

    <Header/>
  </UContainer>
  <UNotifications />
  <UContainer class="px-0">
    <NuxtPage />
  </UContainer>
</template>
<script setup>
console.log("APP VUE")
import { useAuthStore } from '~/stores/auth'
const authStore = useAuthStore()
onBeforeMount( () => {
  authStore.initAuth()
})
</script>

```

Файл nuxt.config.ts

```

import { initCronJobs } from './server/cron'
export default defineNuxtConfig({
  devtools: { enabled: false },
  modules: [
    '@nuxt/ui',
    ['@pinia/nuxt',
      {
        autoImports: ['defineStore'],
      }
    ],
    'vue-yandex-maps/nuxt',
    '@pinia-plugin-persistedstate/nuxt',
  ],
  build: {
    transpile: ['vue-qr-code-reader'],
  },

```

```

yandexMaps: {
  apikey: process.env.YANDEX_MAPS_API_KEY,
},
colorMode: {
  preference: 'dark'
},
runtimeConfig: {
  jwtAccessSecret: process.env.JWT_ACCESS_TOKEN_SECRET,
  jwtRefreshSecret: process.env.JWT_REFRESH_TOKEN_SECRET,
},
typescript: {
  tsConfig: {
    compilerOptions: {
      verbatimModuleSyntax: true
    }
  }
},
hooks: {
  'listen': () => {
    if (process.env.NODE_ENV !== 'production') {
      initCronJobs()
    }
  }
},
css: [
  '~/assets/css/main.css'
],
})

```

Приложение В

(Обязательное)

Сервис электронного бронирования книжных изданий

Презентация

Сервис электронного бронирования книжных изданий

Выполнил студент группы 20ВВП2
Китаев А.А.
Научный руководитель
к.т.н. доцент кафедры ВТ Токарев А.Н.

Пенза, 2024

1

Цель работы

Разработать сервис электронного бронирования книжных изданий

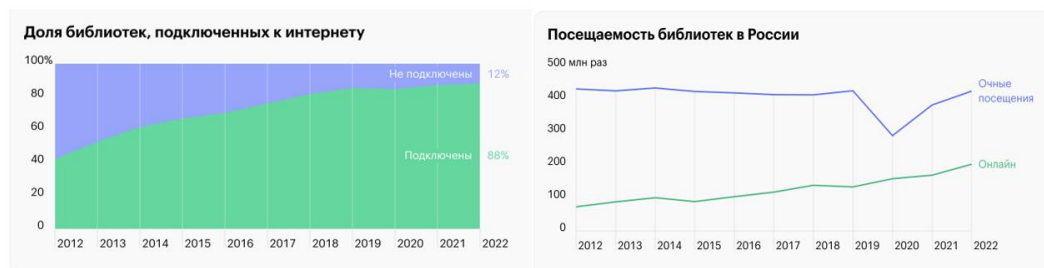
Объект работы

Приложение, позволяющее:

- Искать книги
- Видеть, в каких библиотеках книга доступна
- Бронировать книги
- Получать книги по номеру заказа

2

Актуальность темы

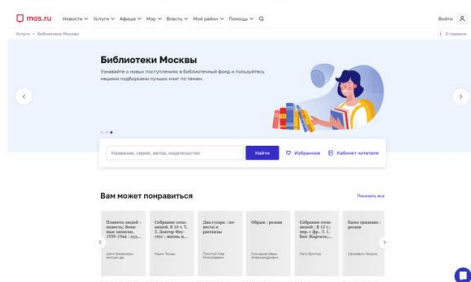


Источник: Минкультуры РФ

3

Анализ аналогов

«Библиотеки Москвы»

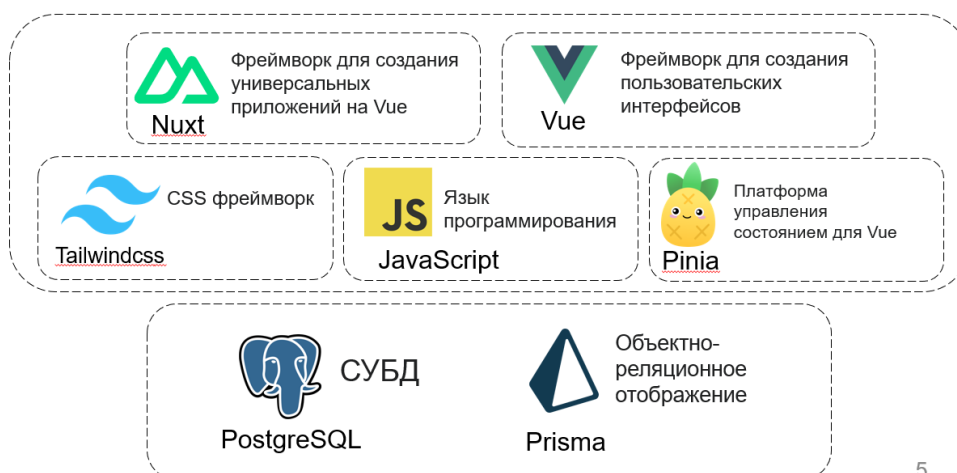


«Библиотеки Санкт-Петербурга»



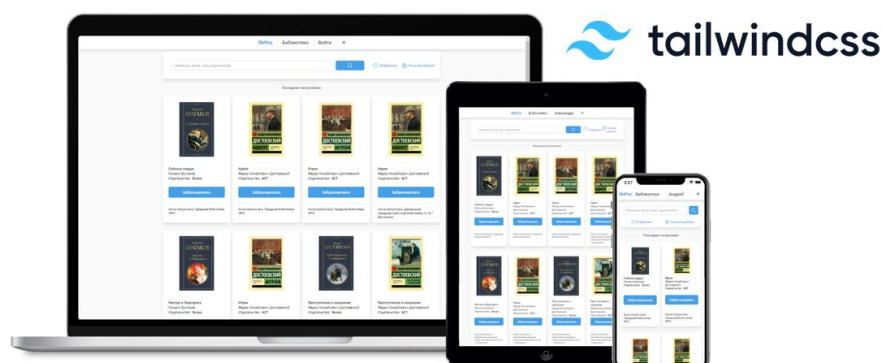
4

Стек технологий



5

Адаптивная вёрстка



6

Изм.	Лист	№ докум.	Подпись	Дата	ПГУ 1.090301.29.001 ПЗ	Лист
						123

Принципы доступности



5.1.7.7 ГОСТ Р 52872-2019

Масштабируемая верстка

(увеличение размера на 500% без появления горизонтальной полосы прокрутки)

5.1.7.3 ГОСТ Р 52872-2019

Контрастность текста и фона

(коэффициент контрастности не менее 4,5:1)

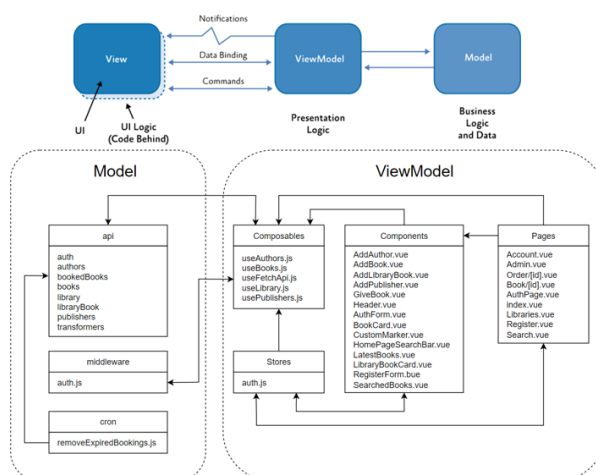
Contrast	AA	AAA
8.28	✓	✓
8.28	✓	✓

5.2.1.1 ГОСТ Р 52872-2019

Возможность работы с помощью клавиатуры

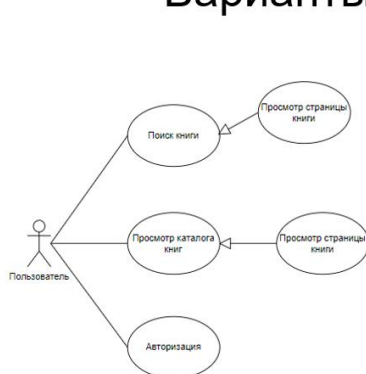
7

Архитектура приложения - MVVM

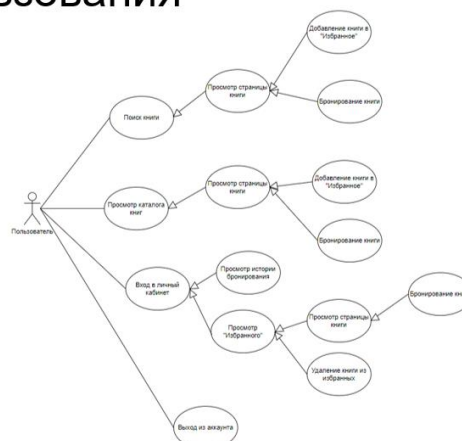


8

Варианты использования



Для неавторизованного пользователя



Для авторизованного пользователя

9

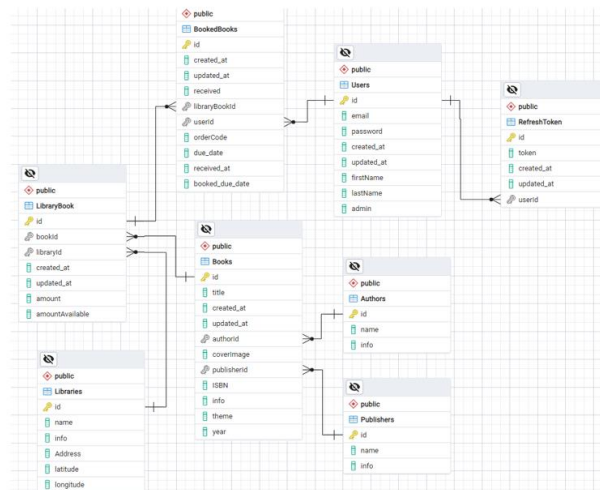
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.29.001 ПЗ

Лист

124

Структура БД



11

Результат работы

Результатом работы является клиент-серверное веб-приложение «Сервис электронного бронирования книжных изданий», которое позволяет:

- получать информацию о доступности книжных изданий в библиотеках
- бронировать книги
- получать книги в библиотеке посредством предоставления номера заказа сотруднику библиотеки

13

Спасибо за внимание!

14

					<p style="text-align: center;"><i>ПГУ 1.090301.29.001 ПЗ</i></p>	Лист
						125
Изм.	Лист	№ докум.	Подпись	Дата		