Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Система арендования книг в библиотеках»

Студент ИУ7-22М (Группа) _____ (Подпись, дата) И. А. Цветков (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) А. А. Ступников (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире, стремительное развитие технологий оказывает значительное влияние на повседневную жизнь человека. Одной из ключевых областей, где технологии играют важную роль, является доступ к информации и культурным ресурсам. Библиотеки, как центры знаний и культурного наследия, остаются важным элементом образовательной и интеллектуальной среды. Однако традиционная модель функционирования библиотек сталкивается с рядом вызовов: быстрый рост цифровых ресурсов, изменение пользовательских предпочтений, а также необходимость оптимизации процесса обслуживания и предоставления доступа к ним.

Одной из современных тенденций является популяризация аренды книг через цифровые платформы, что предоставляет удобство и упрощает доступ к библиотечным ресурсам. Это особенно актуально в условиях городской жизни, где время и доступность ресурсов играют важную роль. Разработка веб-сайтов и мобильных приложений для аренды книг может значительно улучшить взаимодействие пользователей с библиотеками, увеличивая их посещаемость и привлекая новые категории пользователей.

Примером успешного применения подобных технологий является проект цифровой платформы «Московская электронная библиотека», которая предоставляет доступ к сотням тысяч книг, журналов и других материалов. Пользователи могут брать книги в аренду онлайн и читать их как в электронном, так и в традиционном формате. Это показывает, что библиотеки не теряют своей актуальности, а, напротив, приспосабливаются к новым условиям и сохраняют свою роль в обществе.

Актуальность данной темы обусловлена также изменением культурных и потребительских привычек пользователей, которые все больше предпочитают онлайн-сервисы для удовлетворения своих потребностей. В условиях пандемии COVID-19 был зафиксирован резкий рост интереса к удаленным библиотечным услугам, что подчеркнуло необходимость развития и совершенствования цифровых решений для библиотек.

Данный курсовой проект нацелен на разработку веб-сайта аренды книг в библиотеках города, который будет решать задачи оптимизации работы библиотек, упрощения доступа к книгам. В рамках проекта будет создан интерфейс, удобный для поиска и аренды книг, а также для управления

личными данными пользователей и их библиотечными активностями.

Целью работы является разработка системы бронирования книг в библиотеках города. Для ее достижения необходимо выполнить следующие задачи.

1. Произвести анализ аналогичных решений.
2. Проанализировать предметную область.
3. Спроектировать архитектуру распределенной системы.
4. Произвести выбор стека технологий для реализации.
5. Реализовать распределенную систему аренды книг.

1 Аналитическая часть

1.1 Разбор аналогов

Для успешной разработки веб-сайта аренды книг в библиотеках города важно проанализировать существующие аналогичные решения. Это позволит выделить сильные и слабые стороны уже реализованных проектов и использовать полученные данные для создания функционального и удобного ресурса. Рассмотрим несколько популярных примеров аналогов.

1. **ЛитРес: Библиотека** [1] Это один из крупнейших российских онлайн-сервисов, предоставляющий доступ к электронной библиотеке, включая аренду книг для библиотек. Он активно сотрудничает с государственными библиотеками, предлагая пользователям доступ к бесплатным и платным материалам.

   Плюсы:

   — Широкий выбор книг. Пользователи могут арендовать книги из огромной коллекции, включая новинки.

   — Удобный интерфейс. Сайт и мобильное приложение имеют интуитивно понятный дизайн и структуру.

   — Синхронизация с библиотеками. Возможность брать книги в аренду через конкретные библиотеки.

   — Многоформатность. Книги доступны как в электронном, так и в аудиоформатах.

   Минусы:

   — Ограниченный бесплатный доступ. Большая часть контента платная, что может ограничивать круг пользователей.

   — Отсутствие персонализации. Недостаточно индивидуальных рекомендаций и настроек под пользователя.

   — Нет работы с физическими книгами. Платформа ориентирована в основном на электронные ресурсы, что не всегда удовлетворяет потребности пользователей, предпочитающих бумажные издания.

2. **Московская электронная библиотека** [2] Государственный проект, предоставляющий жителям Москвы бесплатный доступ к библиотечным фондам, включая электронные книги и периодику.

   Плюсы:

   — Бесплатный доступ. Пользователи могут бесплатно брать в аренду книги, что делает услугу доступной для всех.

   — Интеграция с городскими библиотеками. Прямое взаимодействие с реальными библиотеками города, что упрощает получение информации о наличии книг.

   — Широкий выбор. Библиотека содержит множество произведений различной тематики, включая учебную литературу.

   Минусы:

   — Устаревший интерфейс. Сайт выглядит морально устаревшим, что затрудняет навигацию и снижает пользовательский опыт.

   — Малый функционал. Нет удобных инструментов для личного кабинета, отсутствуют продвинутые функции поиска и рекомендации.

   — Отсутствие интеграции с физическими книгами. Хотя проект предоставляет электронные ресурсы, нет механизма аренды бумажных книг.

3. **OverDrive (международный аналог)** [3] Это популярная платформа для аренды книг, используемая библиотеками по всему миру. Она позволяет пользователям брать в аренду как электронные, так и аудиокниги через местные библиотеки.

   Плюсы:

   — Интернациональность. Поддержка множества языков и сотрудничество с библиотеками по всему миру.

   — Широкий ассортимент книг. Включает не только художественную литературу, но и учебные и исследовательские работы.

— Интеграция с мобильными устройствами. Поддержка мобильных приложений для различных платформ, что упрощает доступ к арендованным материалам.

Минусы:

— Ограниченное количество копий. Каждая библиотека ограничена количеством цифровых копий, что может привести к дефициту книг.

— Требование регистрации в местных библиотеках. Пользователю нужно быть зарегистрированным в библиотеке, чтобы получить доступ к ресурсам.

— Проблемы с доступом к физическим книгам. Платформа ориентирована на электронные и аудиокниги, а не на физические издания.

Изучив плюсы и минусы существующих решений, можно выделить несколько ключевых моментов для создания более эффективного веб-сайта аренды книг в библиотеках города.

1. Современный и удобный интерфейс. Нужно избегать устаревших интерфейсов и предложить интуитивно понятный и современный дизайн.

2. Бесплатный доступ и интеграция с библиотеками. Необходимо предоставить пользователям возможность арендовать книги бесплатно через городские библиотеки.

Эти элементы помогут создать ресурс, способный конкурировать с существующими решениями и удовлетворить требования современных пользователей.

1.2 Описание системы

Разрабатываемый портал должен представлять собой систему для аренды книг в библиотеках города.

Если пользователь хочет оформить заказ, то ему нужно пройти регистрацию, указав следующую информацию: фамилия, имя, номер телефона, адрес электронной почты, пароль. Для неавторизованных пользователей доступен только просмотр общей информации сайта: списка библиотек и книг в них.

1.3 Функциональные требования к системе с точки зрения пользователя

Портал должен обеспечивать реализацию следующих функций.

1. Система должна обеспечивать регистрацию и авторизацию пользователей с валидацией вводимых данных.
2. Аутентификация пользователей.
3. Разделение всех пользователей на 3 роли:
   - неавторизованный пользователь (гость);
   - авторизированный пользователь (пользователь);
   - администратор.
4. Предоставление возможностей **гостю, пользователю, администратору** представленных в таблице 1.1.

Таблица 1.1 – Функции пользователей

Гость	1. Просмотр списка библиотек (включая фильтрацию по городу); 2. Просмотр списка книг в выбранной библиотеке; 3. Регистрация в системе; 4. Авторизация в системе.
Пользователь	1. Авторизация в системе; 2. Просмотр списка библиотек (включая фильтрацию по городу); 3. Просмотр списка книг в выбранной библиотеке; 4. Получение информации о данных текущего аккаунта; 5. Просмотр всех своих арендованных книг во всех библиотеках с фильтрацией по статусу аренды 6. Получение детальной информации по конкретной аренде на имя текущего пользователя; 7. Оформление аренды книги на имя авторизованного пользователя; 8. Отмена аренды (возврат) на имя авторизованного пользователя.
Администратор	1. Функции пользователя; 2. Просмотр статистики по сайту.

1.4 Входные данные

Входные параметры системы представлены в таблице 1.2.

Таблица 1.2 – Входные данные

Сущность	**Входные данные**
Регистрация пользователя	1. *фамилия* не более 256 символов; 2. *имя* не более 256 символов; 3. *логин* не более 256 символов; 4. *пароль* не более 128 символов; 5. *номер телефона* в формате (+7XXXXXXXXXX); 6. *роль* администратор или пользователь; 7. *электронная почта* в формате (*@*.*).
Аутентификация пользователя	1. *логин* не более 256 символов; 2. *пароль* не более 128 символов.
Аренда книги	1. *идентификатор книги*; 2. *идентификатор библиотеки*; 3. *дата возврата* в формате ДД/ММ/ГГГГ.
Возврат книги	1. *идентификатор аренды*; 2. *состояние книги* прекрасное / хорошее / плохое; 3. *дата возврата* в формате ДД/ММ/ГГГГ.
Фильтр и пагинация библиотек	1. *город* не более 256 символов; 2. *номер страницы* не менее 1; 3. *объектов на странице* не менее 1.
Фильтр и пагинация книг	1. *показать закончившиеся* да / нет; 2. *номер страницы* не менее 1; 3. *объектов на странице* не менее 1.
Фильтр и пагинация аренд	1. *статус аренды* в аренде / возвращена вовремя / возвращена после срока; 2. *номер страницы* не менее 1; 3. *объектов на странице* не менее 1.

1.5 Выходные параметры

Выходными параметрами системы являются web-страницы. В зависимости от запроса и текущей роли пользователя они содержат следующую информацию (таблица 1.3).

Таблица 1.3 – Выходные параметры

Гость	1. Список библиотек: • *название*; • *адрес*; • *город*.
	2. Список книг: • *название*; • *автор*; • *жанр*; • *состояние*; • *количество в библиотеке*.
	3. О сайте: • *общая информация о сайте*; • *правила сайта*; • *контактная информация поддержки*.
Пользователь	1. Список библиотек: • *название*; • *адрес*; • *город*.
	2. Список книг: • *название*; • *автор*; • *жанр*; • *состояние*; • *количество в библиотеке*.
	3. О сайте: • *общая информация о сайте*; • *правила сайта*; • *контактная информация поддержки*.

	4. Детальная информация о пользователе, вошедшем в систему; • *фамилия*; • *имя*; • *логин*; • *номер телефона*; • *рейтинг* число от 0 до 100, характеризующее количество книг, которое может взять пользователь; • *электронная почта.*
	5. Список взятых в аренду книг пользователя, вошедшего в систему: • *библиотека, в которой арендована книга* в соответствии с пунктом 1; • *арендованная книга* в соответствии с пунктом 2; • *дата взятия книги в аренду*; • *дата возврата книги*; • *статус аренды.*
Администратор	1. Список библиотек: • *название*; • *адрес*; • *город.*
	2. Список книг: • *название*; • *автор*; • *жанр*; • *состояние*; • *количество в библиотеке.*
	3. О сайте: • *общая информация о сайте*; • *правила сайта*; • *контактная информация поддержки.*

	4. Детальная информация о пользователе, вошедшем в систему; • *фамилия*; • *имя*; • *логин*; • *номер телефона*; • *рейтинг* число от 0 до 100, характеризующее количество книг, которое может взять пользователь; • *роль в системе*; • *электронная почта.*
	5. Список взятых в аренду книг пользователя, вошедшего в систему: • *библиотека, в которой арендована книга* в соответствии с пунктом 1; • *арендованная книга* в соответствии с пунктом 2; • *дата взятия книги в аренду*; • *дата возврата книги*; • *статус аренды.*
	6. Статистика по порталу, собранная через сервис статистики: • *идентификатор*; • *метод запроса* GET/POST/PATCH/DELETE/OPTIONS; • *url запроса*; • *числовой статус выполнения запроса*; • *время выполнения запроса* в формате ДД/ММ/ГГГГ (ЧЧ:ММ).

1.6 Состав системы

Система будет состоять из фронтенда и 9 подсистем:

— сервис-координатор;

— сервис регистрации и авторизации;

— сервис библиотек;

— сервис рейтинга;

— сервис аренды;

— сервис статистики;

— сервис kafka;

— сервис consumer;

— сервис zookeeper.

1.6.1 Фронтенд

Фронтенд – принимает запросы от пользователя по протоколу HTTP и возвращает ответ в виде HTML страниц, файлов стилей и TypeScript.

1.6.2 Сервис-координатор

Сервис-координатор – сервис, который отвечает за координацию запросов внутри системы. Все сервисы портала (кроме сервиса регистрации и авторизации) должны взаимодействовать друг с другом через сервис-координатор, запросы с фронтенда в том числе сначала должны приходить на сервис-координатор, а затем перенаправляться на нужный сервис. При этом сервис-координатор отвечает за следующие действия.

1. получения списка библиотек в городе с пагинацией от сервиса библиотек 1.6.4;

2. получения списка книг в библиотеке с пагинацией от сервиса библиотек 1.6.4;

3. получения списка книг, арендованных пользователем с пагинацией от сервисов библиотек 1.6.4 и аренды 1.6.6;

4. получения рейтига пользователя от сервиса рейтинга 1.6.5;

5. оформление аренды книги через сервисы библиотек 1.6.4 и аренды 1.6.6 с учетом рейтинга пользователя из сервиса рейтинга 1.6.5 (книг в аренде может быть не больше, чем рейтинг пользователя);

6. возврат книги через сервисы библиотек 1.6.4 и аренды 1.6.6 и с изменением данных в сервисе рейтинга 1.6.5 (если книга возвращена в более плохом состояни и/или позже заявленного срока возврата, то за каждое снимается 10 рейтинга, иначе прибавляется 1 рейтинг).

1.6.3 Сервис регистрации и авторизации

Сервис регистрации и авторизации отвечает за следующие действия.

1. Регистрацию нового пользователя;
2. Аутентификацию пользователя;
3. Авторизацию пользователя;
4. Получение данных пользователей;
5. Изменение данных о пользователе;
6. Удаление пользователя.

Взаимодействие сервиса регистрации и авторизации с остальными сервисами должно осуществляться по протоколу OpenID Connect. Сам сервис представляет из себя Identity Provider [4]. Сервис регистрации и авторизации в своей работе используют базу данных, которая хранит следующую информацию:

— Пользователь:

  – *уникальный идентификатор*;
  – *логин*;
  – *имя*;
  – *фамилия*;
  – *захешированный пароль*;
  – *номер телефона*;
  – *электронная почта*;
  – *роль*.

1.6.4 Сервис библиотек

Сервис библиотек реализует следующие функции.

1. Получение списка всех библиотек с фильтрацией и пагинацией;
2. Получение информации о конкретной библиотеке;
3. Создание библиотеки;
4. Изменение библиотеки;
5. Удаление библиотеки;
6. Получение списка всех книг с фильтрацией и пагинацией;
7. Получение информации о конкретной книге;
8. Создание книги;
9. Изменение книги;
10. Удаление книги;
11. Получение списка всех связей книг и библиотек с фильтрацией и пагинацией;
12. Получение информации о конкретной связи библиотеки и книги;
13. Создание связи библиотеки и книги
14. Изменение связи библиотеки и книги
15. Удаление связи библиотеки и книги.

Сервис использует в своей работе базу данных:

— Библиотека:

  – *уникальный идентификатор*;
  – *название*;
  – *город*;

- *адрес*.

— Книга:

- *уникальный идентификатор*;
- *название*;
- *автор*;
- *жанр*;
- *состояние*.

— Связь библиотеки и книги:

- *уникальный идентификатор*;
- *уникальный идентификатор библиотеки*;
- *уникальный идентификатор книги*;
- *количестов* книг в библиотеке по связии.

1.6.5 Сервис рейтинга

Сервис рейтинга реализует следующие функции.

1. Получение списка рейтингов всех пользователей с фильтрацией и пагинацией;
2. Получение информации о конкретной рейтинге;
3. Создание рейтинга;
4. Изменение рейтинга;
5. Удаление рейтинга.

Сервис использует в своей работе базу данных:

— Рейтинг:

- *уникальный идентификатор*;
- *логин пользователя*;
- *рейтинг* число от 0 до 100.

1.6.6 Сервис аренды

Сервис аренды реализует следующие функции.

1. Получение списка аренд всех пользователей с фильтрацией и пагинацией;
2. Получение информации о конкретной аренде;
3. Создание аренды;
4. Изменение аренды;
5. Удаление аренды.

Сервис использует в своей работе базу данных:

— Аренда:

- *уникальный идентификатор*;
- *логин пользователя*;
- *уникальный идентификатор библиотеки*;
- *уникальный идентификатор книги*;
- *статус* в аренде / возвращена в срок / возвращена после срока;
- *дата взятия книги в аренду*;
- *дата возврата книги.*

1.6.7 Сервис статистики

Сервис статистики – сервис, который отвечает за запись событий сервиса координатора в базу данных для осуществления возможности быстрого обнаружения, локализации и воспроизведения ошибки в случае её возникновения. Дает возможность получить статистику с пагинацией.

Сервис использует в своей работе базу данных:

— Статистика:

- *уникальный идентификатор*;
- *метод запроса* GET/POST/PATCH/DELETE/OPTIONS;

– *url запроса*;

– *числовой статус выполнения запроса*;

– *время выполнения запроса.*

1.6.8 Сервис kafka

Сервис kafka [5] – сервис, который необходим для сервиса статистики для сбора и обработки данных в реальном времени, что позволяет анализировать пользовательскую активность. Kafka поддерживает высокие объёмы данных и легко масштабируется, обеспечивая надёжную работу даже при значительных нагрузках. Благодаря встроенной отказоустойчивости и гарантии доставки сообщений, система статистики не потеряет важные данные при сбоях.

1.6.9 Сервис consumer

Сервис consumer – сервис, который нужен kafka для получения, обработки и анализа данных, поступающих от producer в реальном времени. Kafka действует как посредник, обеспечивая доставку сообщений между различными сервисами, что позволяет consumer-серверам асинхронно получать данные и обрабатывать их по мере поступления. Это важно для поддержания высокой производительности и отказоустойчивости, так как Kafka распределяет нагрузку между несколькими consumer-серверами, помогая избежать перегрузки. Также Kafka гарантирует надёжную доставку сообщений, что позволяет consumer корректно обрабатывать каждое сообщение без риска потери данных. Наконец, она обеспечивает возможность параллельной обработки данных, что ускоряет анализ больших объёмов информации.

1.6.10 Сервис zookeeper

Сервис zookeeper [6] – сервис, который нужен kafka для управления и координации различных компонентов в своей распределённой системе. Вот ключевые задачи, которые решает Zookeeper в Kafka.

1. Координация кластеров: Zookeeper помогает координировать работу брокеров (серверов Kafka) внутри кластера, отслеживая их состояние.

Он сообщает Kafka о том, какие узлы доступны и активно работают, обеспечивая бесперебойное взаимодействие между ними.

2. Управление метаданными: Zookeeper хранит важную информацию о топиках, партициях и распределении лидеров партиций среди брокеров. Это нужно для того, чтобы потребители (consumers) и производители (producers) могли эффективно взаимодействовать с нужными данными в кластере.

3. Обнаружение лидера: Zookeeper определяет лидера для каждой партиции Kafka, который отвечает за запись и чтение данных. В случае сбоя одного из брокеров Zookeeper автоматически выбирает нового лидера для партиции, чтобы поддерживать непрерывную работу.

4. Отказоустойчивость: Zookeeper обеспечивает высокую доступность и надёжность кластера Kafka, помогая восстанавливать компоненты после сбоев и поддерживать согласованное состояние всех узлов системы. Управление доступом: Zookeeper управляет доступом клиентов к Kafka и координирует изменения конфигурации, обеспечивая стабильность и безопасность работы кластера.

5. Таким образом, Zookeeper является критически важным компонентом для обеспечения координации, отказоустойчивости и управления Kafka-кластером.

1.7 Требования к программной реализации

1. Требуется использовать СОА (сервис-ориентированную архитектуру) для реализации системы.

2. Система состоит из микросервисов. Каждый микросервис отвечает за свою область логики работы приложения и должны быть запущены изолированно друг от друга.

3. При необходимости, каждый сервис имеет своё собственное хранилище, запросы между базами запрещены.

4. При разработке базы данных необходимо учитывать, что доступ к ней должен осуществляться по протоколу TCP.

5. Необходимо реализовать один web-интерфейс для фронтенда. Интерфейс должен быть доступен через тонкий клиент (браузер).

6. Для межсервисного взаимодействия использовать HTTP (придерживаться RESTful).

7. Выделить Gateway Service как единую точку входа и межсервисной коммуникации. В системе не должно осуществляться горизонтальных запросов.

8. Необходимо предусмотреть авторизацию пользователей через интерфейс приложения.

9. Код хранить на Github, для сборки использовать Github Actions.

10. Каждый сервис должен быть завернут в docker.

1.8 Функциональные требования к подсистемам

Фронтенд – серверное приложение, предоставляет пользовательский интерфейс и внешний API системы, при разработке которого нужно учитывать следующее:

— должен принимать запросы по протоколу HTTP и формировать ответы пользователям в формате HTML;

— в зависимости от типа запроса должен отправлять последовательные запросы в соответствующие микросервисы;

— запросы к микросервисам необходимо осуществлять по протоколу HTTP;

— данные необходимо передавать в формате JSON;

— целесообразно использовать Tailwind для упрощения написания стилей.

Сервис-координатор – это серверное приложение, которое должно отвечать следующим требованиям по разработке:

— обрабатывать запросы в соответствии со своим назначением, описанным в топологии системы;

— принимать и возвращать данные в формате JSON по протоколу HTTP;

— использовать очередь для отложенной обработки запросов (например, при временном отказе одного из сервисов);

— осуществлять деградацию функциональности в случае отказа некритического сервиса (зависит от семантики запроса);

— уведомлять сервис статистики о событиях в системе.

Сервис регистрации и авторизации, сервис библиотек, сервис рейтинга, сервис аренды, сервис статистики – это серверные приложения, которые должны отвечать следующим требованиям по разработке:

— обрабатывать запросы в соответствии со своим назначением, описанным в топологии системы;

— принимать и возвращать данные в формате JSON по протоколу HTTP;

— осуществлять доступ к СУБД по протоколу TCP.

Сервис kafka, сервис consumer, сервис zookeeper – это серверное приложение, которое должно отвечать следующим требованиям по разработке:

— обрабатывать запросы в соответствии со своим назначением, описанным в топологии системы;

— принимать и возвращать данные в формате JSON по протоколу HTTP.

1.9 Пользовательский интерфейс

Для реализации пользовательского интерфейса должен быть использован подход MVC (Model-View-Controller). Этот подход к проектированию интерфейса является популярным шаблоном проектирования, который помогает разделить логику приложения на три основных компонента: Модель (Model), Представление (View) и Контроллер (Controller). Этот подход позволяет улучшить структуру приложения, облегчить его тестирование и управление, а также разработка фронтенда и бекенда могут быть полностью разделены между собой, то есть можно вести независимую разработку.

Пользовательский интерфейс в разрабатываемой системе должен обладать следующими характеристиками:

— Кроссбраузерность – способность интерфейса работать практически в любом браузере любой версии.

— «Плоский» дизайн» – дизайн, в основе которого лежит идея отказа от объемных элементов (теней элементов, объемных кнопок и т.д.) и замены их плоскими аналогами.

— Расширяемость – возможность легко расширять и модифицировать пользовательский интерфейс.

1.10 Сценарий взаимодействия с приложением

Приведем пример работы портала на примере выполнения запроса от пользователя на получение списка аренд пользователя.

1. На фронтенд приходит запрос пользователя.

2. Если пользователь был авторизован, то происходит получение токена аторизации. Затем выполняется запрос к сервису-координатору. Если данные корректны (данные поступили в ожидаемом формате) и проверка JWT-токена (проверка того, что токен был подписан известным серверу ключом и того, что срок действия токена ещё не истёк) (если он истёк или оказался некорректным, пользователю возвращается ошибка), то отправляется запрос на сервисы аренды и библиотек аренд.

3. Выполняются запросы к соответствующим эндпоинтам сервиса библиотек для получения даннных о библиотеках и книгах, осуществляется проверка корректности полученных данных (данные поступили в ожидаемом формате) и проверка JWT-токена (проверка того, что токен был подписан известным серверу ключом и того, что срок действия токена ещё не истёк). Если он истёк или оказался некорректным, пользователю возвращается ошибка. При успешной проверке токена сервис возвращает списки библиотек и книг сервису-координатору, который агрегирует полученные данные в одну таблицу и происходит возврат результата на фронтенд.

4. Если ошибки нигде не произошло, то производится генерация HTML содержимого страницы ответа пользователю с использованием данных,

полученных от сервиса-координатора. В ином случае генерируется страница с описанием ошибки.

2 Конструкторская часть

2.1 Концептуальный дизайн

Для создания функциональной модели портала, отражающей его основные функции и потоки информации наиболее наглядно использовать нотацию IDEF0. На рисунке 2.1 приведена концептуальная модель системы. На рисунке 2.2 представлена детализированная концептуальная модель системы в нотации IDEF0.

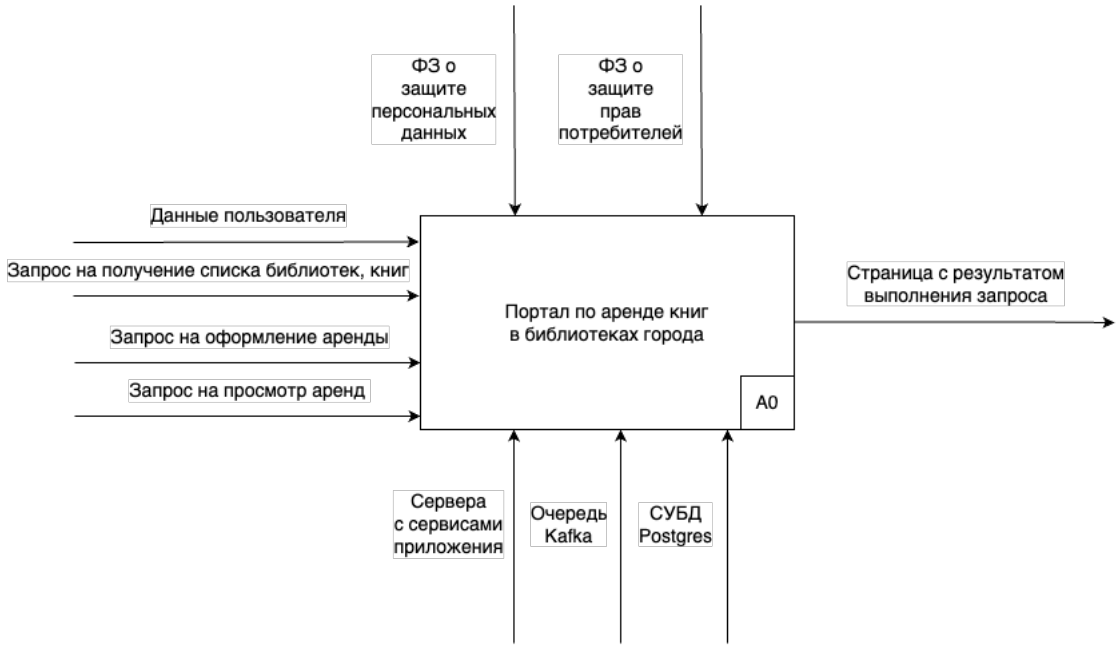

Рисунок 2.1 – Концептуальная модель системы в нотации IDEF0

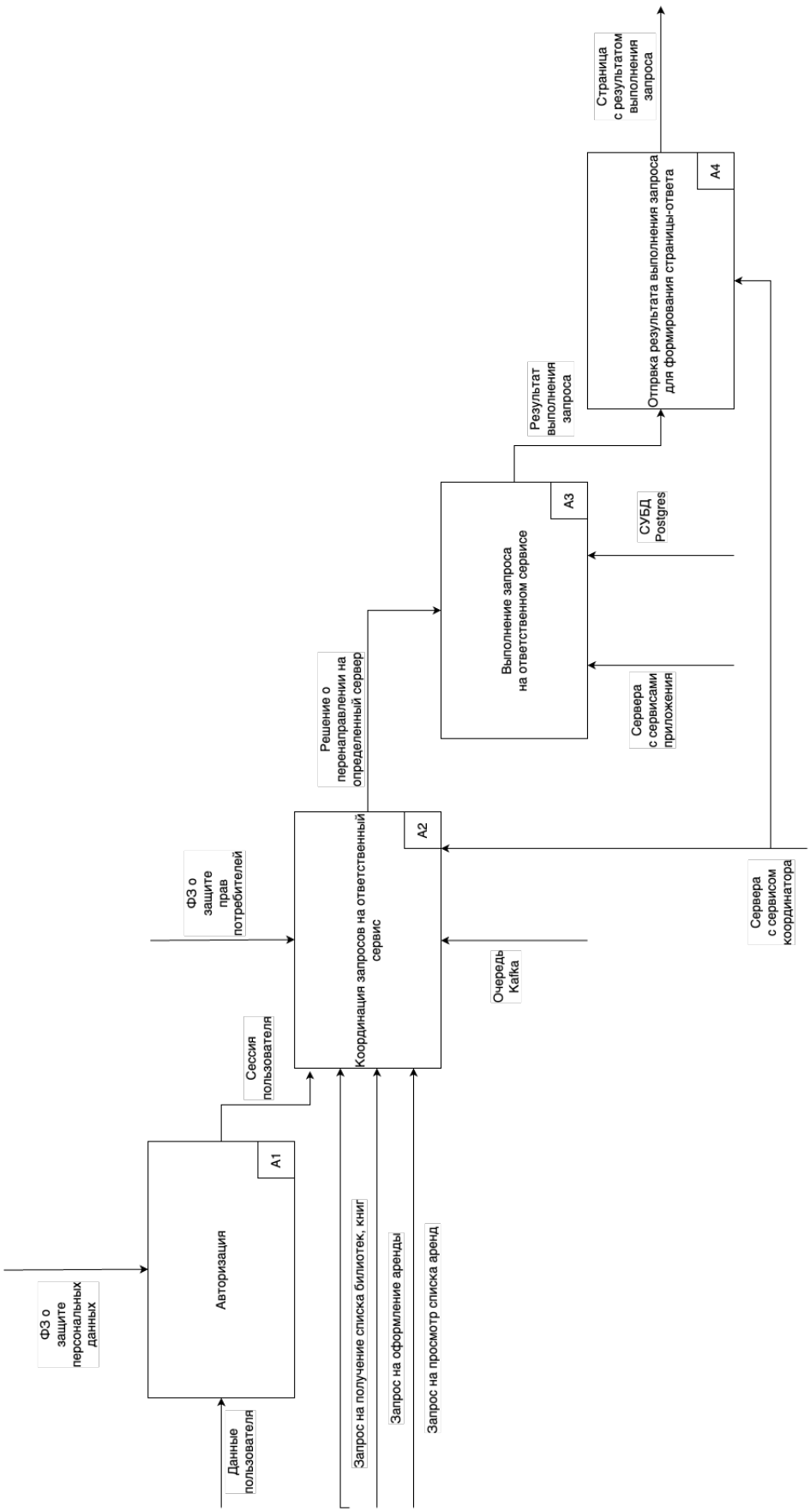

Рисунок 2.2 – Детализированная концептуальная модель системы в нотации IDEF0

2.2 Сценарии функционирования системы

Регистрация пользователя

1. Пользователь нажимает на кнопку «Войти» в интерфейсе.
2. Так как у пользователя нет аккаунта, он нажимает на кнопку «Зарегистрироваться».
3. Пользователь перенаправляется на страницу, которая содержит поля для заполнения его данных.
4. Пользователь вводит данные в форму и для завершения регистрации нажимает на кнопку «Готово», тем самым подтверждая верность своих данных, а также согласие на их обработку и хранение.
5. Если пользователь с введенным для регистрации логином, почтой или номером телефона уже существует, то клиент получает сообщение об ошибке. При успешной регистрации клиент попадает на главную страниицу сайта.

Авторизация клиента

1. Пользователь нажимает на кнопку «Войти» в интерфейсе.
2. Пользователь перенаправляется на страницу авторизации, которая содержит поля для заполнения логина и пароля.
3. Пользователь завершает работу с формой авторизации нажатием кнопки «Готово».
4. При обнаружении ошибки в данных, пользователь получает сообщение об ошибке; при совпадении данных с записью в базе данных аккаунтов пользователь получает доступ к системе и перенаправляется на гланый экран сайта.

Оформление аренды книги

1. Пользователь на главной странице видит список всех библиотек. При желании он может отфильтровать их по городу расположения.

2. Пользователь нажимает на кнопку «Посмотреть книги» у понравившейся библиотеки.

3. Пользователь выбирает книгу в выбранной библиотеке и нажимает кнопку «Забронировать».

4. Пользователь попадает на страницу, на которой он видит полную информацию о выбранной библиотеке, выбранной книге. Там же ему нужно ввести дату возврата книги и подтвердить согласие с правилами сайта.

5. Если все верно и пользователь ввел все данные, он нажимает на кнопку «Готово». Если же книга в момент аренды не закончилась в библиотеке, пользователь аутентифицирован и его рейтинг больше, чем количество уже арендованных книг, то он получает соответсвующее сообщение об ошибке. Иначе книга попадает в список забронированных.

Возврат книги из аренды

1. Пользователь нажимает на иконку профиля.

2. Ему выпадает список вкладок сайта, доступных аутентифицированному пользователю.

3. Пользователь нажимает на вкладку «Бронирования».

4. Пользователь попадает на страницу, на которой он видит все свои аренды книг. Есть возможность отфильтровать по статусу бронирования.

5. Пользователь выбирает аренду, котороую он хочет завершить и нажимает на соответсвующее окно бронирования.

6. Пользователь получает информацию о библиотеке, в которой эта книга арендована, и информацию о самой книге, а также о датах взятия книги в аренду и дате возврата.

7. Пользователь нажимает на кнопку «Вернуть книгу» и получает модальное окно, в котором необходимо ввести состояние книги и согласиться с правилами сайта.

8. Если пользователь вернул книгу в более плохом состоянии, чем она была или позже срока, то за каждое условие он получит минус 10 к рейтингу, что уменьшит количество книг, которое он сможет арендовать в дальнейшем. Иначе пользователь получит плюс 1 к рейтингу.

9. Затем пользователь увидит, что статус аренды книги изменился на «Возвращена в срок» или «Возвращена после срока».

2.3 Диаграммы прецедентов

В системе выделены 3 роли: Неавторизованный пользователь, Авторизованный пользователь, Администратор. На рисунках 2.3-2.5 представлены диаграммы прецедентов для каждой из ролей.

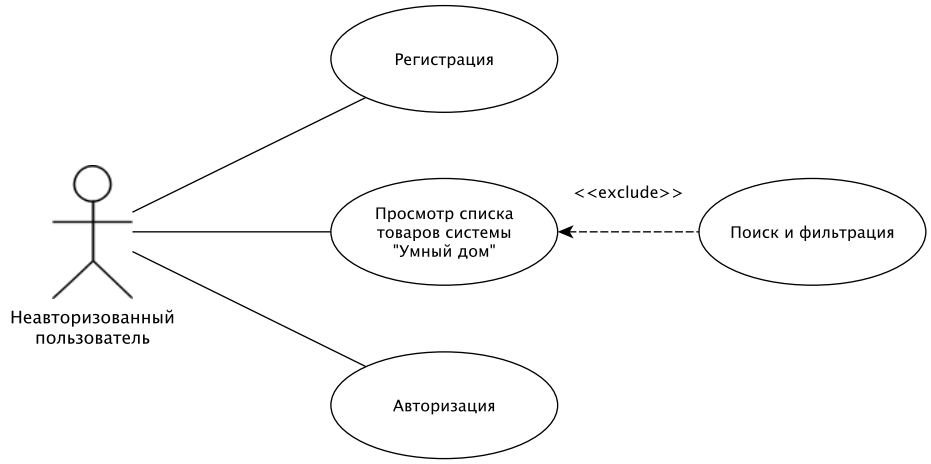

Рисунок 2.3 – Диаграмма прецедентов с точки зрения Неавторизованного пользователя

Рисунок 2.4 – Диаграмма прецедентов с точки зрения Авторизованного пользователя

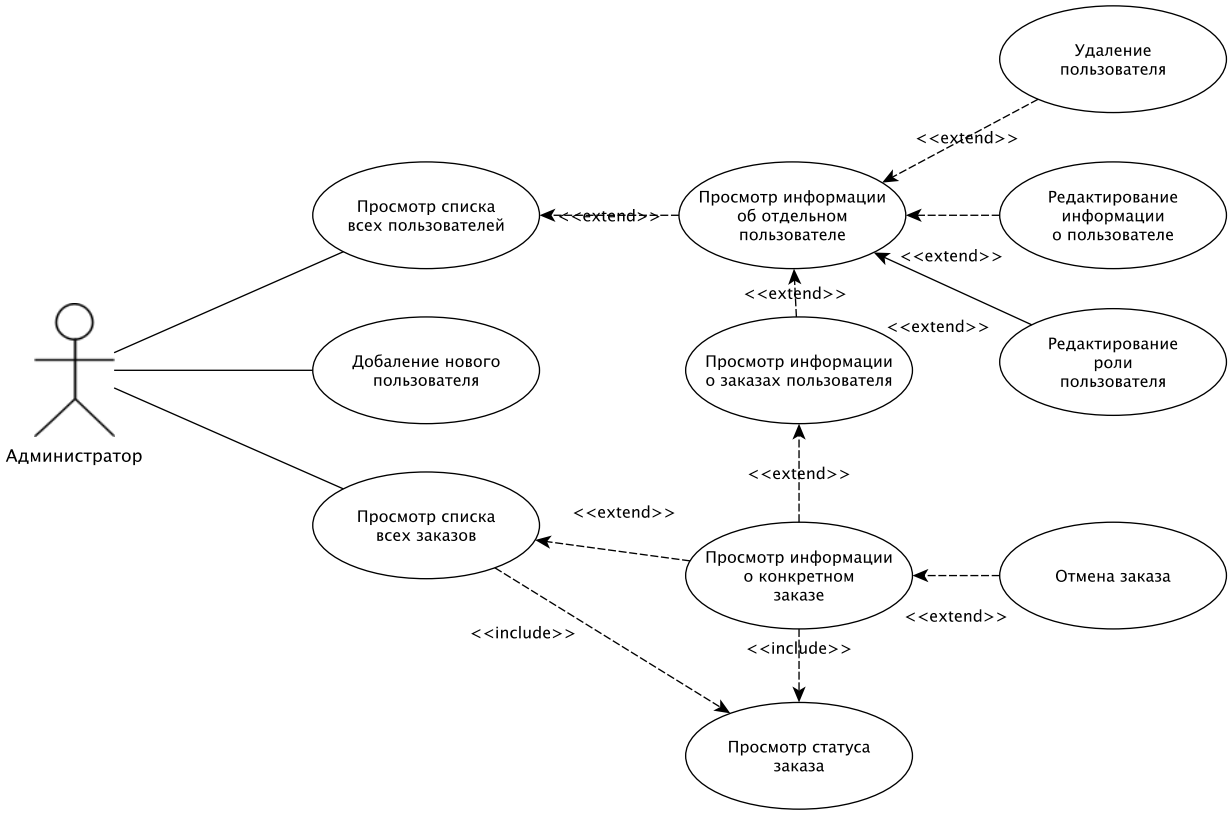

Рисунок 2.5 – Диаграмма прецедентов с точки зрения Администратора

2.4 Высокоуровневый дизайн пользовательского интерфейса

Пользовательский интерфейс в разрабатываемой системе представляет собой web-интерфейс, доступ к которому осуществляется через браузер (тонкий клиент). Страница системы состоит из «шапки», основной части.

Обобщенно структуру страниц системы можно представить следующим образом:

— страница авторизации;

— страница регистрации;

— главная страница со списком библиотек и информацией о каждой из них;

— страница со списком книг и информацией о каждой из них;

— страница с арендой книги;

— страница со всеми арендованными книгами;

— страница о сайте с правилами и контактной информацией.

Также на рисунках 2.6-?? приведены для примеры страниц: «Главная», «Авторизация», «Бронирование книги» и «Все бронирования».

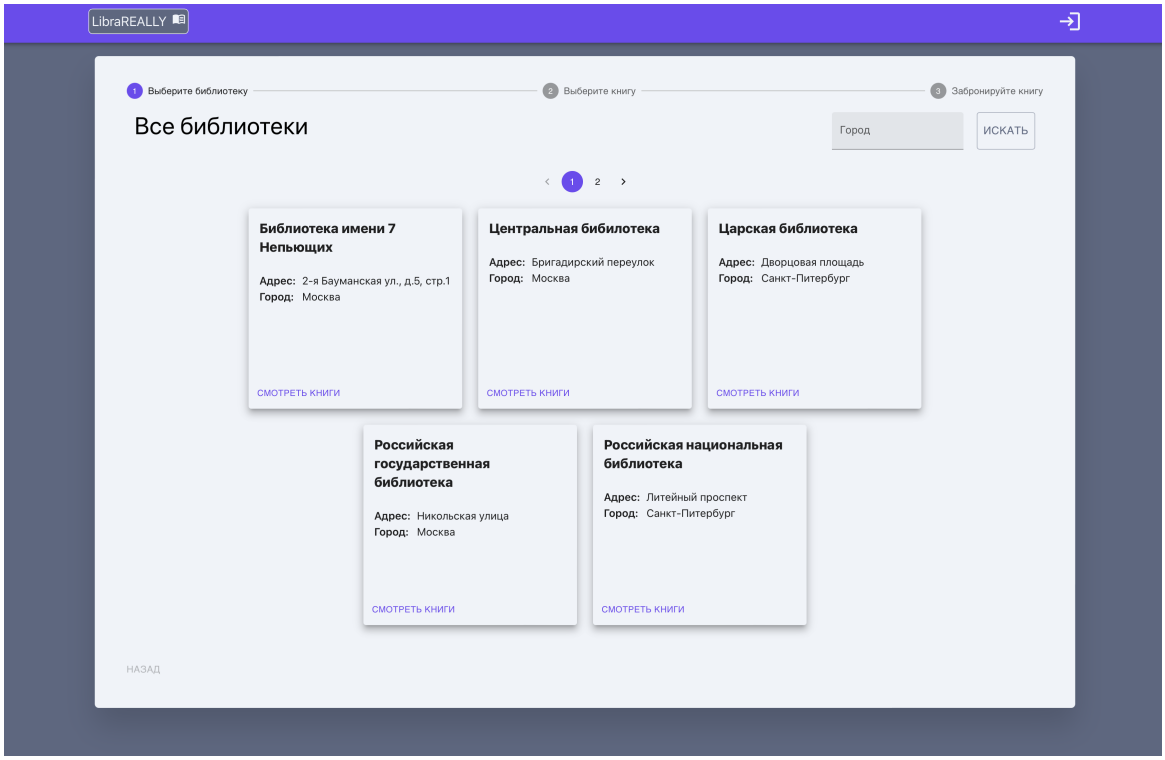

Рисунок 2.6 – Главная страница

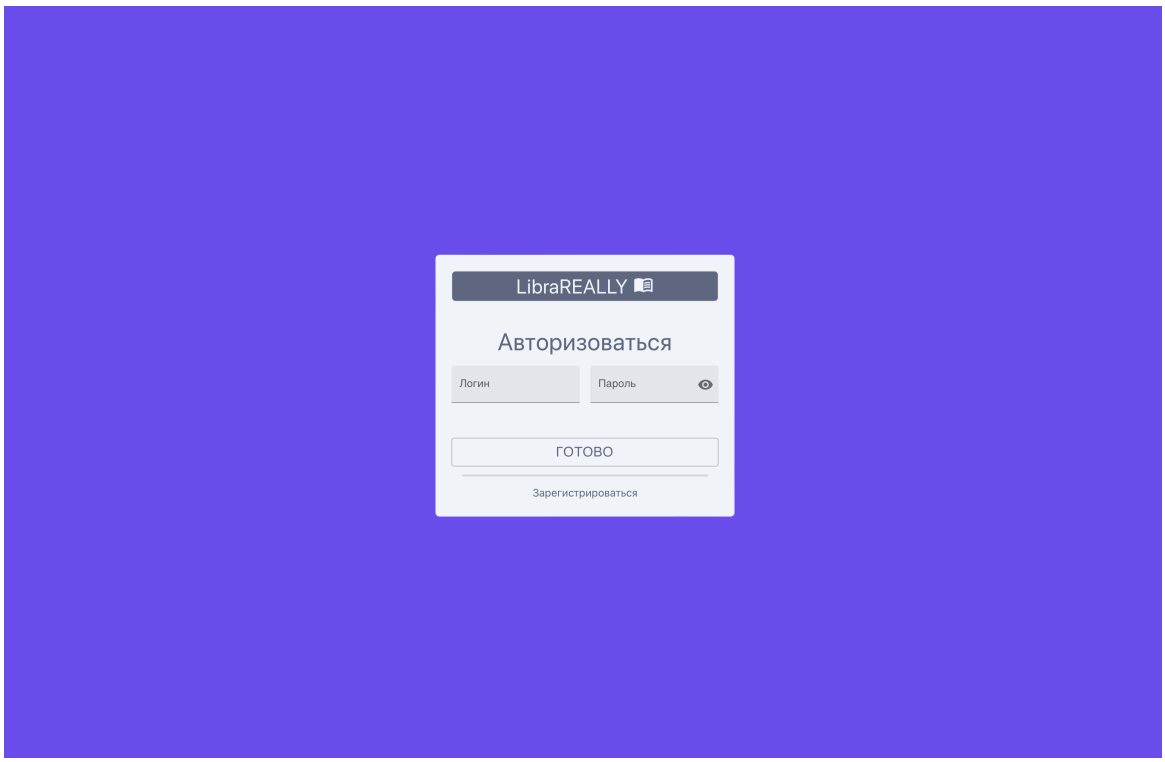

Рисунок 2.7 – Страница авторизации

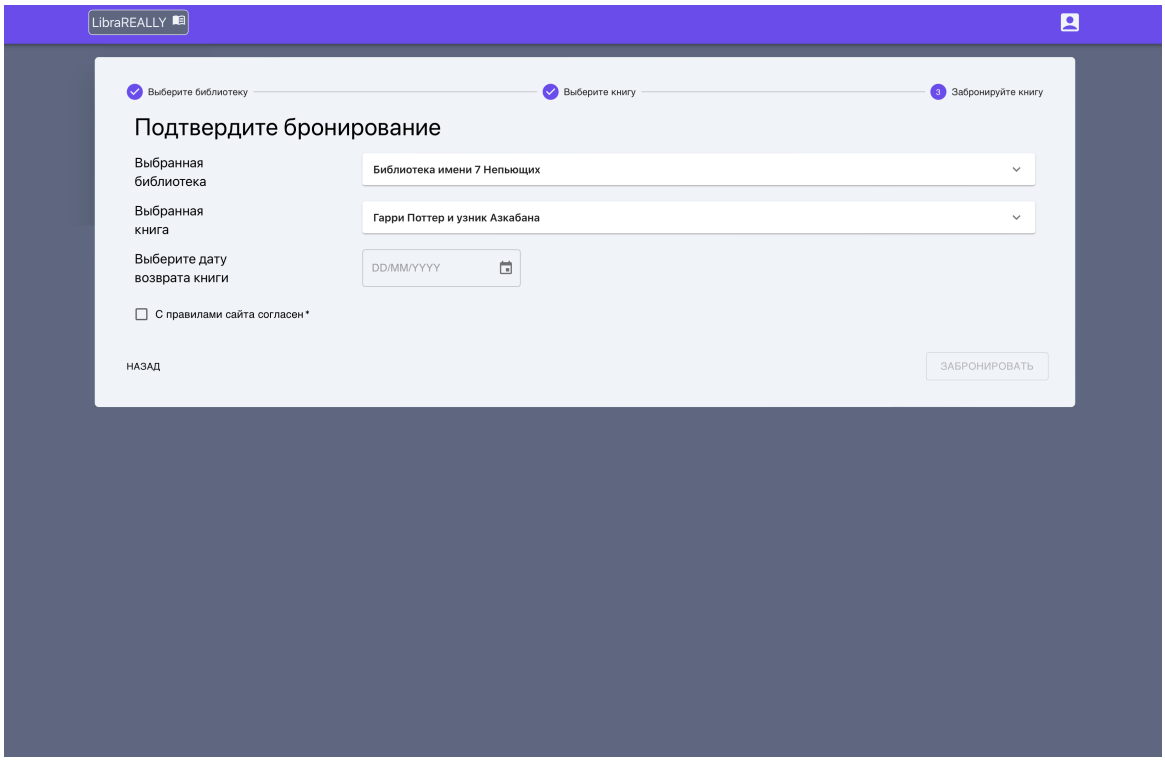

Рисунок 2.8 – Страница бронирования книги

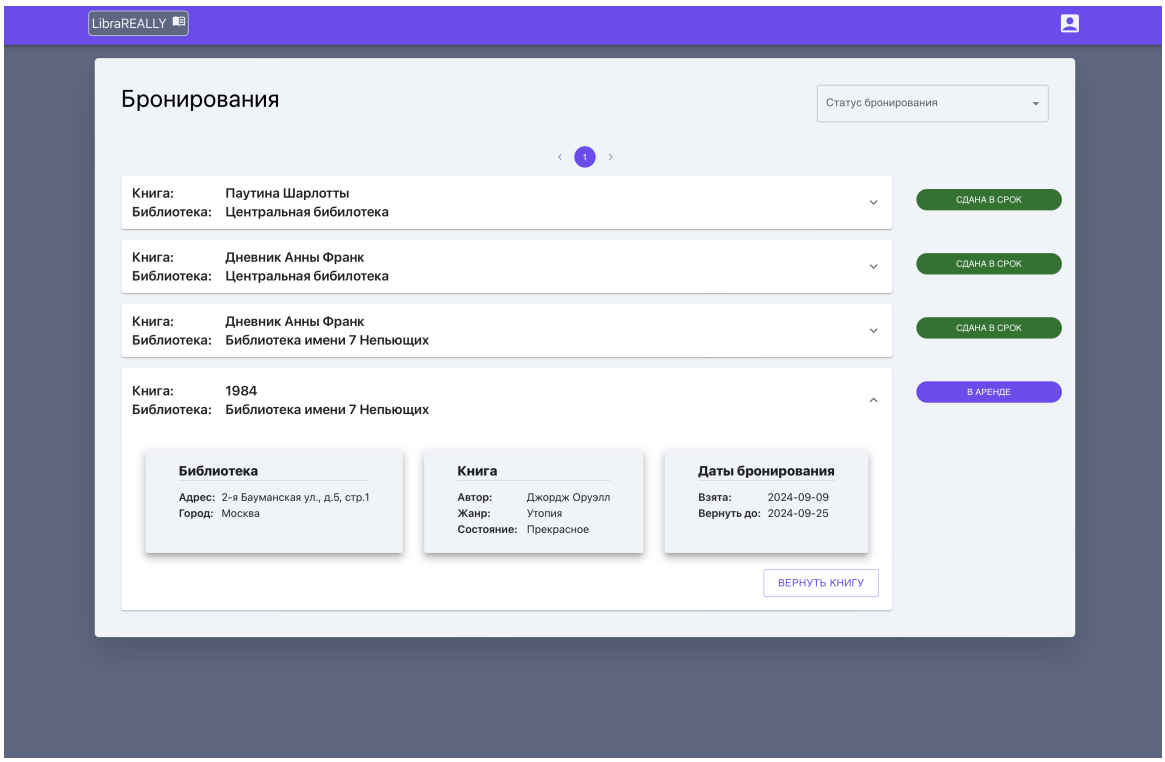

Рисунок 2.9 – Страница со всеми забронированными книгами

3 Технологическая часть

3.1 Выбор операционной системы

Согласно требованиям технического задания, разрабатываемый портал должен обладать высокой доступностью, работать на типичных архитектурах ЭВМ (Intel x86, Intel x64), а так же быть экономически недорогим для сопровождения. Таким образом, требования к ОС следующие.

— **Распространенность**. На рынке труда должно быть много специалистов, способных администрировать распределенную систему, работающую под управлением выбранной операционной системы.

— **Надежность**. Операционная система должна широко использоваться в стабильных проектах, таких как Mail.Ru, Vk.com, Google.com. Эти компании обеспечивают высокую работоспособность своих сервисов, и на их опыт можно положиться.

— **Наличие требуемого программного обеспечения**. Выбор операционной системы не должен ограничивать разработчиков в выборе программного обеспечения, библиотек.

— **Цена**.

Под данные требования лучше всего подходит ОС Linux, дистрибутив Ubuntu. **Ubuntu** [7] — это дистрибутив, использующий ядро Linux. Как и все дистрибутивы Linux, Ubuntu является ОС с открытым исходным кодом, бесплатным для использования. Поставляется как в клиентской (с графическим интерфейсом), так и в серверной (без графического интерфейса) версиями. Ubuntu поставляется с современными версиями ПО. Преимуществом Ubuntu являются низкие требования к квалификации системных администраторов. Однако Ubuntu менее стабильна в работе.

3.2 Выбор СУБД

В качестве СУБД была выбрана **PostgreSQL** [8], так как она наилучшим образом подходит под требования разрабатываемой системы:

— Масштабируемость: PostgreSQL поддерживает горизонтальное масштабирование, что позволяет распределить данные и запросы между несколькими узлами базы данных. Это особенно полезно в географически распределенных системах, где данные и пользователи могут быть разбросаны по разным регионам.

— Географическая репликация: PostgreSQL предоставляет возможность настройки репликации данных между различными узлами базы данных, расположенными в разных географических зонах. Это позволяет обеспечить отказоустойчивость и более быстрый доступ к данным для пользователей из разных частей нашей страны.

— Гибкость и функциональность: PostgreSQL обладает широким набором функций и возможностей, что делает его подходящим для различных типов приложений и использования в распределенной среде. Он поддерживает сложные запросы, транзакции, хранимые процедуры и многое другое.

— Надежность и отказоустойчивость: PostgreSQL известен своей надежностью и стабильностью работы. В распределенной географической системе это особенно важно, поскольку он способен обеспечить сохранность данных и доступность даже при сбоях в отдельных узлах.

3.3 Выбор языка разработки и фреймворков компонент портала

Для разработки бэкенда существует множество языков программирования, каждый из которых имеет свои сильные и слабые стороны. Рассмотрим наиболее популярные языки: Python, Java, JavaScript (Node.js), Go и Ruby:

1. Python.
   (a) Плюсы.
      i. Простота и читаемость кода: Python славится своим чистым синтаксисом, что делает его одним из самых легких языков для изучения и поддержки. Это особенно важно для командной работы и проектов, которые должны быть легко расширяемыми.

ii. Богатая экосистема: Существуют мощные фреймворки, такие как Django и Flask, которые значительно ускоряют разработку веб-приложений. Python также имеет огромное количество библиотек для работы с базами данных, кешированием, очередями и другими бэкенд-задачами.

iii. Гибкость: Python применим не только для веб-разработки, но и для множества других задач, таких как обработка данных, машинное обучение, автоматизация и многое другое. Это делает его универсальным выбором.

iv. Сообщество и поддержка: Огромное сообщество разработчиков и обилие документации обеспечивают быстрое решение проблем и поддержку развития проектов.

(b) Минусы.

i. Скорость выполнения: Python является интерпретируемым языком, что делает его медленнее по сравнению с компилируемыми языками, такими как Java или Go. Однако эта проблема часто нивелируется мощностью серверов и оптимизацией кода.

2. Java.

(a) Плюсы.

i. Высокая производительность: Java компилируется в байт-код, который затем исполняется виртуальной машиной (JVM), что обеспечивает высокую производительность.

ii. Масштабируемость: Java часто используется в крупных корпоративных приложениях, благодаря своей способности легко масштабироваться.

iii. Безопасность и стабильность: Java известна своей стабильностью, что делает её подходящей для критически важных систем, требующих высокой надёжности.

(b) Минусы.

i. Сложность кода: Java имеет более громоздкий синтаксис по сравнению с Python, что делает разработку медленнее, а код — труднее читаемым.

ii. Более медленная разработка: Разработка на Java требует больше времени из-за необходимости писать больше кода для реализации аналогичных функций.

3. JavaScript (Node.js).

   (a) Плюсы.

      i. Единый язык для фронтенда и бэкенда: Если ваш проект использует JavaScript на фронтенде, то использование Node.js позволяет унифицировать стек технологий.

      ii. Асинхронная обработка: Node.js работает на основе событийной модели, что позволяет эффективно управлять большим количеством запросов одновременно.

      iii. Широкое сообщество: JavaScript — один из самых популярных языков, что гарантирует обилие ресурсов, библиотек и инструментов.

   (b) Минусы.

      i. Однопоточная модель: Несмотря на асинхронную природу Node.js, она может быть ограничена при выполнении тяжёлых задач, которые требуют много вычислительных ресурсов.

      ii. Меньшая стабильность: JavaScript часто обновляется, а экосистема модулей может быть нестабильной по сравнению с более зрелыми языками.

4. Go (Golang).

   (a) Плюсы.

      i. Высокая производительность: Go компилируется в машинный код, что делает его крайне быстрым.

      ii. Простота в управлении многозадачностью: Go предлагает встроенную поддержку параллелизма через горутины, что облегчает создание масштабируемых приложений.

      iii. Меньше накладных расходов: Go подходит для создания микросервисов и работы с высоконагруженными системами.

(b) Минусы.

i. Ограниченная экосистема: Хотя Go активно развивается, экосистема библиотек и фреймворков не так богата, как у Python или Java.

ii. Меньшая гибкость: Go ориентирован на высокую производительность, но его синтаксис менее гибок для других задач, таких как анализ данных или машинное обучение.

5. Ruby.

(a) Плюсы.

i. Простота и элегантность: Ruby, как и Python, предлагает удобный и понятный синтаксис, что ускоряет разработку.

ii. Фреймворк Ruby on Rails: Один из самых популярных фреймворков для веб-разработки, который значительно ускоряет создание веб-приложений.

(b) Минусы.

i. Производительность: Ruby медленнее, чем многие другие языки (например, Go или Java), что делает его менее подходящим для высоконагруженных приложений.

ii. Меньшая популярность: Несмотря на сильные стороны Ruby on Rails, язык постепенно теряет популярность, уступая Python и JavaScript.

Таким образом, **Python** [9] —- это оптимальный выбор для бэкенд-разработки по ряду причин:

1. Простота и скорость разработки: Python позволяет разработчикам писать меньше кода и быстрее реализовывать функциональность, благодаря удобному синтаксису и мощным фреймворкам (Django, Flask, FastAPI). Это особенно важно для стартапов и небольших проектов, где критично быстрое создание прототипов и внедрение изменений.

2. Широкая экосистема: В Python существует множество готовых библиотек для работы с базами данных, аутентификацией, кэшированием,

очередями, обработкой данных и другими задачами бэкенда. Это ускоряет разработку и снижает количество ручной работы.

3. Поддержка современных технологий: Python активно используется для задач, связанных с машинным обучением, обработкой данных и научными вычислениями. Это делает его выбором номер один для компаний, работающих с большими данными или развивающих искусственный интеллект.

4. Сообщество и поддержка: Python имеет одно из крупнейших сообществ разработчиков, что упрощает решение проблем, обновление знаний и получение поддержки.

5. Универсальность: Python может использоваться не только для разработки веб-приложений, но и для автоматизации, обработки данных и других областей, что делает его многофункциональным инструментом.

Также выберем фреймворк для бекенд разработки. FastAPI, Django и Flask — популярные фреймворки для создания веб-приложений на Python. Каждый из них имеет свои особенности, подходы и сферы применения. Рассмотрим ключевые различия.

1. FastAPI.

   (a) Плюсы.

      i. Асинхронная природа обеспечивает высокую производительность для обработки большого числа запросов.

      ii. Простота и понятность благодаря аннотациям типов и автоматической валидации данных.

      iii. Интеграция с современными стандартами API и поддержка OpenAPI.

      iv. Подходит для микросервисной архитектуры.

   (b) Минусы.

      i. Не такой полный фреймворк, как Django: нет встроенной поддержки ORM или административной панели "из коробки"(но это компенсируется сторонними библиотеками).

2. Django

   (a) Плюсы.

      i. Полноценный фреймворк для создания полнофункциональных приложений.

      ii. Быстрый старт для создания сложных веб-приложений благодаря встроенным инструментам.

      iii. Отличная документация и большое сообщество.

      iv. Интегрированная административная панель для удобного управления данными.

   (b) Минусы.

      i. Меньшая гибкость по сравнению с более лёгкими фреймворками, такими как Flask или FastAPI.

      ii. Не поддерживает асинхронность "из коробки"(сравнительно новые версии поддерживают её частично).

      iii. Более тяжеловесен для небольших проектов или микросервисов, где не нужны все встроенные возможности.

3. Flask

   (a) Плюсы.

      i. Минимализм и простота. Легко настраивается для нужд любого проекта.

      ii. Высокая гибкость и возможность выбора инструментов.

      iii. Хорошо подходит для микросервисной архитектуры.

   (b) Минусы.

      i. Отсутствие встроенных решений: многие базовые вещи, такие как ORM, маршрутизация или системы шаблонов, необходимо подключать через сторонние библиотеки.

      ii. Сложнее масштабировать и поддерживать крупные проекты, так как проектная структура зависит от разработчиков.

Таким образом, для разработки был выбран фреймфорк **FastAPI** [10] по следующим причинам.

1. Производительность: FastAPI предлагает одну из самых высоких производительностей среди Python-фреймворков. Это особенно важно для приложений, требующих быстрого отклика при большом количестве запросов, таких как API и микросервисы.

2. Поддержка асинхронности: В отличие от Django, который частично поддерживает асинхронные операции, FastAPI изначально построен с учётом асинхронного программирования. Это делает его идеальным для приложений, где необходимо эффективно обрабатывать большое количество параллельных запросов (например, в реальном времени или при работе с внешними API).

3. Простота и автоматическая валидация: FastAPI использует аннотации типов Python для автоматической валидации данных и генерации документации, что значительно упрощает работу с API. Это улучшает качество кода и сокращает количество ошибок.

4. Генерация документации "из коробки": FastAPI автоматически создаёт документацию API с использованием стандартов OpenAPI и Swagger. Это экономит время разработчиков и упрощает работу с клиентами и другими командами.

5. Гибкость и современность: FastAPI предлагает более гибкий и лёгкий подход к разработке по сравнению с Django, сохраняя простоту и читабельность кода, как в Flask. Он идеально подходит для создания быстрых, масштабируемых и лёгких приложений, особенно в микросервисной архитектуре.

3.4 Выбор фреймворка фронтенд разработки

Фреймворки для разработки фронтенда, такие как React, Angular и Vue, предоставляют различные подходы к созданию современных веб-приложений. Рассмотрим основные отличия этих фреймворков.

1. React (Библиотека для построения пользовательских интерфейсов (UI), с акцентом на компонентный подход)

   (a) Плюсы.

i. Широкая поддержка и большое сообщество разработчиков.

ii. Гибкость в выборе технологий для архитектуры приложения.

iii. Легко интегрируется в существующие проекты.

iv. Широко используется для мобильной разработки с помощью React Native.

(b) Минусы.

i. Требует настройки экосистемы (например, выбор между Redux, MobX или другими для управления состоянием).

ii. Не является полным фреймворком, что может потребовать больше усилий для конфигурации.

2. Angular (Полноценный фреймворк, предлагающий строгую структуру для разработки приложений с двусторонней привязкой данных)

(a) Плюсы.

i. Полноценный фреймворк с решениями "из коробки"для большинства задач.

ii. Поддержка TypeScript, что улучшает качество кода и упрощает работу в крупных проектах.

iii. Отличная документация и регулярные обновления от Google.

(b) Минусы.

i. Большая кривизна обучения по сравнению с React или Vue.

ii. Больше шаблонного кода и сложностей для выполнения простых задач.

iii. Более "тяжеловесный"по сравнению с другими решениями, что может сказываться на производительности.

3. Vue.js (Прогрессивный фреймворк, который можно постепенно интегрировать в существующий проект)

(a) Плюсы.

i. Простота использования и лёгкая кривая обучения.

ii. Высокая производительность благодаря лёгкости фреймворка.

iii. Прогрессивность: можно добавлять Vue к уже существующим проектам по мере необходимости.

(b) Минусы.

i. Меньшая популярность по сравнению с React и Angular, что приводит к менее богатой экосистеме и меньшему количеству библиотек.

ii. Меньшее количество крупных компаний используют Vue, что может сказаться на количестве вакансий и поддержке.

Таким образом, был выбран фреймворк **React** [11] по следующим причинам.

1. Популярность и поддержка сообщества: React является одним из самых популярных инструментов для фронтенд-разработки. Благодаря этому для React доступно множество библиотек, инструментов и ресурсов. Это значительно упрощает разработку, особенно при необходимости интеграции с другими технологиями.

2. Гибкость: В отличие от Angular, который является "жёстким"фреймворком с предустановленными решениями, React предлагает больше свободы. Разработчики могут выбирать любые библиотеки для маршрутизации, управления состоянием и работы с сервером, адаптируя проект под конкретные требования.

3. Производительность через Virtual DOM: React использует Virtual DOM для минимизации реальных изменений в DOM, что делает его быстрым даже для сложных пользовательских интерфейсов. Это особенно важно для крупных приложений с динамически изменяющимися данными.

4. Поддержка мобильной разработки: React Native предоставляет возможность использовать знания и компоненты React для разработки мобильных приложений под iOS и Android. Это позволяет создавать кроссплатформенные приложения с минимальными усилиями.

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была проведена разработка веб-приложения для аренды книг в библиотеках города, включающая как клиентскую, так и серверную части. На основании проведённого анализа современных технологий были выбраны оптимальные инструменты для создания высокопроизводительного и масштабируемого решения. В качестве бэкенд-фреймворка был использован FastAPI языка программирования Python, обеспечивающий высокую производительность, поддержку асинхронного программирования и простоту интеграции с современными инструментами. Для фронтенда был выбран React, отличающийся гибкостью, большим сообществом и поддержкой современных подходов к созданию интерактивных пользовательских интерфейсов.

Разработанное приложение предоставляет удобный интерфейс для поиска и аренды книг, а также обеспечивает автоматизированный сбор и обработку данных о пользователях и аренде. В процессе реализации были решены задачи по организации архитектуры приложения, обеспечению безопасности данных.

Данная работа продемонстрировала возможность применения современных технологий для создания удобных и функциональных веб-приложений, способных решать задачи цифровизации библиотек и улучшать доступ к знаниям для широкого круга пользователей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Литрес [Электронный ресурс]. — Режим доступа URL: https://www.litres.ru (Дата обращения: 09.04.2024).
2. Электронная библиотека [Электронный ресурс]. — Режим доступа URL: https://academia-moscow.ru/elibrary/ (Дата обращения: 09.04.2024).
3. OverDrive [Электронный ресурс]. — Режим доступа URL: https://www.overdrive.com (Дата обращения: 09.04.2024).
4. Identity Provider [Электронный ресурс]. — Режим доступа URL: https://www.okta.com/identity-101/why-your-company-needs-an-identity-provider/ (Дата обращения: 09.04.2024).
5. Kafka [Электронный ресурс]. — Режим доступа URL: https://kafka.apache.org (Дата обращения: 09.04.2024).
6. Zookeeper [Электронный ресурс]. — Режим доступа URL: https://zookeeper.apache.org (Дата обращения: 09.04.2024).
7. Ubuntu [Электронный ресурс]. — Режим доступа URL: https://ubuntu.com (Дата обращения: 09.04.2024).
8. PostgreSQL [Электронный ресурс]. — Режим доступа URL: https://www.postgresql.org (Дата обращения: 09.04.2024).
9. Python [Электронный ресурс]. — Режим доступа URL: https://www.python.org (Дата обращения: 09.04.2024).
10. FastAPI [Электронный ресурс]. — Режим доступа URL: https://fastapi.tiangolo.com (Дата обращения: 09.04.2024).
11. React [Электронный ресурс]. — Режим доступа URL: https://react.dev (Дата обращения: 09.04.2024).