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»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Библиотечная система бронирования»

Студент ИУ7-23М _____ Н. Р. Трошкин
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка распределенной системы бронирования книг в библиотеках. Для достижения поставленной цели необходимо решить следующие задачи:

— сформулировать основные требования к системе;

— описать требования к подсистемам;

— описать архитектуру системы, а так же сценарии её функционирования;

— выбрать и обосновать инструменты для разработки программного обеспечения;

— реализовать программный продукт.

1 Аналитический раздел

1.1 Постановка задачи

Разрабатываемая система должна предоставлять пользователям информацию о наличии книг в библиотеках, возможность найти интересующую книгу и взять ее в библиотеке. Если у пользователя на руках есть уже N книг, то он не может взять новую, пока не сдал старые. Если пользователь возвращает книги в хорошем состоянии и сдает их в срок, то максимальное возможное количество книг у него на руках увеличивается.

1.2 Описание системы

Разрабатываемое программное обеспечение должно представлять собой распределённую систему для поиска и бронирования книг в библиотеках.

Авторизованные пользователи могут просмотреть список библиотек в городе, список книг в выбранной библиотеке, информацию по всем книгам, которые были взяты в прокат данным пользователем, свой рейтинг, а также забронировать и вернуть книгу в библиотеке. Неавторизованные пользователи могут только зарегистрироваться или войти в систему, если учетная запись уже существует. Для регистрации необходимо указать следующую информацию о себе: фамилия, имя, отчество, дата рождения, номер телефона, электронная почта.

1.3 Общие требования к системе

Требования к системе следующие:

1. Разрабатываемое ПО на рекомендованной аппаратной конфигурации должно поддерживать функционирование системы в режиме 24/7/365 (24 часа, 7 дней в неделю, 365 дней в году) со среднегодовым временем доступности не менее 99.9%. Допустимое время, в течении которого система недоступна, за год должна составлять $24 \cdot 365 \cdot 0.001 = 8.76$ ч.

2. Время восстановления системы после сбоя не должно превышать 15 минут.

3. Каждый узел должен восстанавливаться после сбоя автоматически.

4. Система должна поддерживать возможность «горячего» переконфигурирования. Необходимо предусмотреть поддержку добавления нового узла во время работы системы без рестарта.

5. Обеспечить безопасность работоспособности за счёт отказоустойчивости узлов.

1.4 Требования к функциональным характеристикам

1. По результатам работы модуля сбора статистики среднее время отклика системы на запросы пользователя на получение информации не должна превышать 3 секунд.

2. По результатам работы модуля сбора статистики медиана времени отклика системы на запросы, добавляющие или изменяющие информацию на портале не должна превышать 5 секунд.

3. Медиана времени отклика системы на действия пользователя должна быть менее 0.8 секунд при условии работы на рекомендованной аппаратной конфигурации, задержках между взаимодействующими сервисами менее 0.2 секунды и одновременном числе работающих пользователей менее 100 на каждый сервер, обслуживающий внешний интерфейс.

1.5 Функциональные требования к системе с точки зрения пользователя

Система должна обеспечивать реализацию следующих функций.

1. Система должна обеспечивать регистрацию и авторизацию пользователей с валидацией вводимых данных как через интерфейс приложения, так и через социальные сети.

2. Система должна обеспечивать аутентификацию пользователей.

3. В системе должно быть обеспечено разделение всех пользователей на три роли:

   - Пользователь (неавторизированный пользователь);
   - Клиент (авторизированный пользователь);

- Администратор.

4. Предоставление возможностей **Пользователю, Клиенту, Администратору** представленных в таблице 1.

Таблица 1: Функции пользователей

Пользователь	1. регистрация в системе; 2. авторизация в системе;
Клиент	1. получение и изменение информации текущего аккаунта; 2. просмотр списка доступных библиотек; 3. просмотр списка книг, доступных в конкретной библиотеке; 4. просмотр всех своих бронирований; 5. просмотр информации о своем рейтинге; 6. бронирование конкретной книги в конкретной библиотеке; 7. возврат забронированной данным клиентом книги в библиотеку;
Администратор	1. просмотр списка всех клиентов, зарегистрированных в системе; 2. создание новых учетных записей; 3. получение отчетов о выполненных действиях от сервиса статистики; 4. просмотр и редактирование списка доступных библиотек; 5. просмотр и редактирование списка доступных книг в конкретной библиотеке; 6. бронирование книги в библиотеке на имя любого клиента; 7. возврат книги в библиотеку от имени любого клиента; 8. изменение рейтинга любого клиента; 9. просмотр списка бронирований любого клиента;

1.6 Входные данные

Входные параметры системы представлены в таблице 2.

Таблица 2: Входные данные

Сущность	**Входные данные**
Клиент / Администратор	1. *фамилия, имя* и *отчество* не более 256 символов каждое поле; 2. *дата рождения* в формате дд.мм.гггг; 3. *логин* не более 80 символов; 4. *пароль* не менее 8 символов и не более 128, как минимум одна заглавная и одна строчная буква, только латинские буквы, без пробелов, как минимум одна цифра; 5. *номер телефона*; 6. *электронная почта*; 7. *роль пользователя* — клиент или администратор;
Рейтинг	1. *идентификатор* пользователя; 2. *логин* пользователя не более 80 символов; 3. *рейтинг* — целое число от 0 до 100;
Библиотека	1. *идентификатор* библиотеки; 2. *UUID* библиотеки; 3. *название* библиотеки не более 80 символов; 4. *город*, в котором находится библиотека, не более 255 символов; 5. *адрес* библиотеки не более 255 символов;
Книга	1. *идентификатор* книги; 2. *UUID* книги; 3. *название* книги не более 255 символов; 4. *автор* книги не более 255 символов; 5. *жанр* книги не более 255 символов; 6. *состояние* книги, одно из трех: EXCELLENT, GOOD, BAD;

Продолжение на следующей странице

Сущность	**Входные данные**
Связь книги и библиотеки	1. *идентификатор* книги; 2. *идентификатор* библиотеки; 3. *количество* экземляров книги в библиотеке;
Бронирование	1. *идентификатор* бронирования; 2. *UUID* бронирования; 3. *логин* пользователя, совершившего бронирование, не более 80 символов; 4. *UUID* забронированной книги; 5. *UUID* библиотеки, в которой совершено бронирование; 6. *статус* бронирования, одно из трех: RENTED, RETURNED, EXPIRED; 7. *дата и время* совершения бронирования; 8. *дата и время*, когда истекает бронирование.

1.7 Выходные параметры

Выходными параметрами системы являются web-страницы. В зависимости от запроса и текущей роли пользователя они содержат следующую информацию (таблица 3)

Таблица 3: Выходные параметры

Клиент	1. список доступных библиотек в городе, в списке указывается: • *название* библиотеки; • *полный адрес* библиотеки; • *город*;
	2. список книг в выбранной библиотеке, в котором указывается: • *название* книги; • *автор* книги; • *жанр* книги; • *состояние* книги; • *количество экземляров* книги в выбранной библиотеке.

	3. список взятых пользователем в прокат книг, в котором указывается: • *статус* бронирования; • *дата начала* бронирования; • *дата окончания* бронирования; • *название* книги; • *автор* книги; • *жанр* книги; • *название* библиотеки; • *адрес* библиотеки; • *город* библиотеки;
	4. информация о рейтинге пользователя: • *рейтинг* пользователя.
Администратор	Страницы, аналогичные страницам, доступным клиентам, а также: 5. список клиентов, зарегистрированных в системе: • *имя* клиента; • *логин* клиента; • *дата рождения* клиента; • *почта* клиента; • *телефон* клиента.
	6. отчет о пришедших данных из сервиса статистики.

1.8 Топология Системы

На рисунке 1 изображён один из возможных вариантов топологии разрабатываемой распределенной Системы.

Система будет состоять из фронтенда и 6 подсистем:

— сервис-координатор;

— сервис авторизации;

— сервис объектов библиотек;

— сервис бронирования;

— сервис рейтингов;

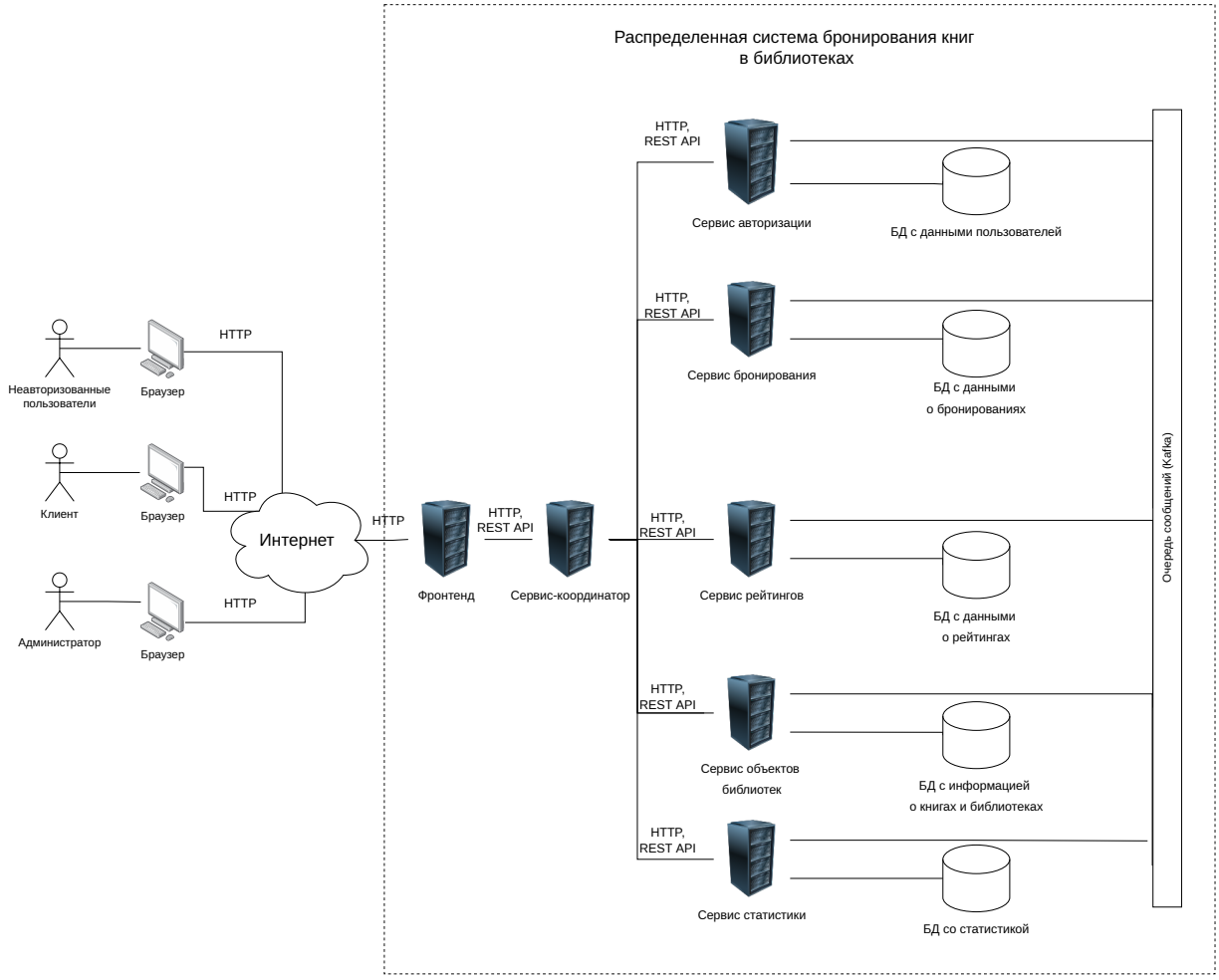

Рисунок 1: Топология системы

— сервис статистики.

Фронтенд принимает запросы от пользователей по протоколу HTTP и анализирует их. На основе проведенного анализа выполняет запросы к микросервисам бекенда, агрегирует ответы и отсылает их пользователю.

Сервис-координатор – единая точка входа и межсервисной коммуникации.

Сервис авторизации отвечает за:

— возможность регистрации нового клиента;

— аутентификацию пользователя (клиента и администратора);

— авторизацию пользователя;

— изменение персональных данных пользователя;

— выход из сессии.

Сервис объектов библиотек отвечает за хранение информации обо всех библиотеках и книгах в них.

Сервис бронирования реализует следующие функции:

— хранение и передача информации о книгах, взятых в прокат пользователем;

— бронирование новой книги в библиотеке;

— возврат книги в библиотеку.

Сервис рейтингов реализует хранение рейтингов пользователей системы.

Сервис статистики получает через Kafka информацию о выполненных действиях и по этим данным строит отчет для администратора.

1.9 Требования к программной реализации

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

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

3. Необходимо реализовать один web-интерфейс для фронтенда в виде SPA или мобильного клиента с обязательным использованием CSS. Интерфейс должен быть доступен через тонкий клиент (браузер).

4. Взаимодействие между сервисами осуществляется посредством HTTP-запросов и/или очереди сообщений по технологии REST [1].

5. Требуется выделить Gateway Service как единую точку входа и межсервисной коммуникации. В системе не должно осуществляться горизонтальных запросов.

6. На Gateway Service для всех операций чтения реализовать паттерн Circuit Breaker [2]. Накапливать статистику в памяти, и если система не

ответила N раз, то в N + 1 раз вместо запроса сразу отдавать fallback. Через небольшой timeout выполнить запрос к реальной системе, чтобы проверить ее состояние.

7. При недоступности подсистем должна осуществляться деградация функциональности (если подсистема некритичная) или выдача пользователю сообщения об ошибке и полный откат операции (либо постановка запроса в очередь для повторного выполнения и возврат пользователю сообщения об успехе).

8. Все методы /api/** (кроме /api/v1/authorize и /api/v1/callback) на всех сервисах должны быть закрыты token-based авторизацией.

9. Если авторизация некорректная (отсутствие токена, ошибка валидации JWT токена [3], закончилось время жизни токена (поле exp в payload)), то отдавать 401 ошибку.

10. Развертывание приложения выполнить с использованием Managed Kubernetes Cluster [4] и настроить Ingress Controller [5] (для публикации сервисов наружу можно использовать только Ingress). Для деплоя использовать helm charts [6], они должны быть универсальными для всех сервисов и отличаться лишь набором параметров запуска. Сервисы должны быть завернуты в docker.

1.10 Функциональные требования к подсистемам

Подсистемы: фронтенд, бекенд-координатор, бекенд авторизации, бекенд объектов библиотеки, бекенд бронирования, бекенд рейтингов, бекенд статистики.

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.

Сервис-координатор – серверное приложение, через которое проходит весь поток запросов и ответов, должен соответствовать следующим требованиям разработки:

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

— для всех операций чтения реализовать паттерн Circuit Breaker — накапливать статистику в памяти, и если система не ответила N раз, то в N + 1 раз вместо запроса сразу отдавать fallback, а через небольшой timeout выполнить запрос к реальной системе, чтобы проверить ее состояние.

— выполнять валидацию JWT токена с помощью JWKs;

— реализовывать запросы, связанные с получением информации о книгах, библиотеках, бронированиях, пользователях и рейтинге пользователей, а также запросы статистики от администратора;

— реализовывать бизнес-логику бронирования книги в библиотеке, в которую включается:

  — получение информации от пользователя о выбранной библиотеке, книге и дате окончания бронирования;

  — проверка количества книг у пользователя на руках (в состоянии RENTED);

  — проверка рейтинга пользователя (рейтинг определяет максимальное возможное количество книг у пользователя на руках);

  — если у пользователя меньше книг на руках, чем его рейтинг, запрос в сервис бронирований о создании нового бронирования;

- запрос на изменение числа доступных экземпляров взятой книги в библиотеке, где произошло бронирование;
- возврат пользователю информации об успешном или неуспешном бронировании.

— реализовывать бизнес-логику возврата книги в библиотеку, в которую включается:

- получение информации от пользователя о бронировании, которое он собирается закрыть, состоянии книги и фактической дате возврата;
- запрос на изменение статуса бронирования (возвращено или просрочено);
- запрос на изменение числа доступных экземпляров возвращенной книги в библиотеке, куда произошел возврат;
- если у книги ухудшилось состояние или возврат просрочен, запрос на уменьшение рейтинга пользователя на 10 за каждое условие;
- в противном случае запрос на увеличение рейтинга пользователя на 1.

Сервис авторизации должен реализовывать следующие функциональные возможности:

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

— возможность регистрации нового клиента и обновление данных уже существующего;

— выполнять функцию Identity Provider с использованием протокола OpenID Connect [7] (scope: openid, profile, email);

— для получения токена на Identity Provider должен быть реализован Authorization Flow [8] (UI рассматривается как 3rd party application и авторизация пользователя так же выполняется через Authorization Flow).

Сервис объектов библиотеки реализует следующие функции:

— получение и отправка данных в формате JSON по протоколу HTTP;

— получение таких данных, как:

  — список доступных библиотек в заданном городе;

  — список книг в конкретной библиотеке;

— валидация JWT токена с помощью JWKs.

Сервис рейтингов реализует функции:

— получение и отправка данных в формате JSON по протоколу HTTP;

— предоставления информации об рейтинге по идентификатору пользователя;

— валидация JWT токена с помощью JWKs.

Сервис бронирования должен реализовывать представленные такие функциональные возможности, как:

— получение и отправка ответов на запросы в формате JSON по протоколу HTTP;

— валидация JWT токена с помощью JWKs;

— предоставление информации о списке книг, взятых в прокат пользователем;

— создание и обновление записей о бронировании в базе данных при взятии книги из библиотеки или возврате в библиотеку.

Сервис статистики получает через Kafka [9] информацию о выполненных действиях и по этим данным строит отчет для администратора и валидирует его JWT-токен.

2 Конструкторский раздел

2.1 Концептуальный дизайн

Концептуальный дизайн позволяет рассмотреть создаваемую систему с точки зрения пользователей. На рисунке 3 отображена контекстная диаграмма верхнего уровня, которая обеспечивает наиболее общее описание работы системы. Данный вид диаграммы позволяет формализовать описание запросов пользователя и ответов системы на них, отобразив её в виде «чёрного ящика».

Для уточнения деталей по операции бронирования книги, отображённой на диаграмме верхнего уровня, используется дочерняя диаграмма, которая изображена на рисунке 4. Она определяет последовательность выполнения операций в системе при обработке запроса клиента.

2.2 UML-диаграммы

В системе выделены три роли: Пользователь, Клиент, Администратор. На рисунках 2, 5-6 представлены диаграммы прецедентов для каждой из ролей.

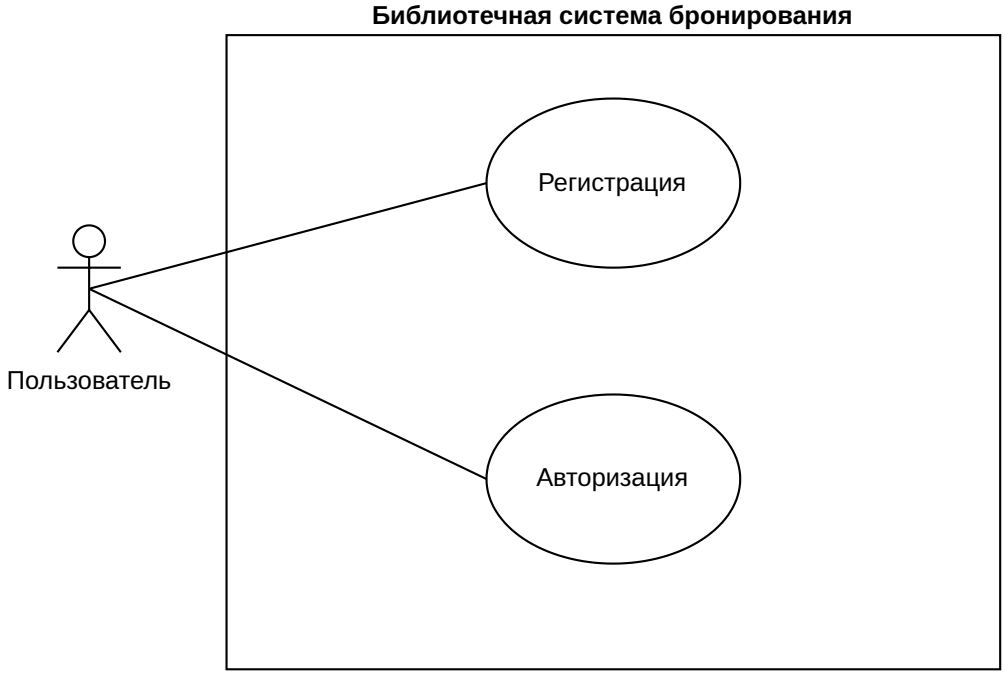

Рисунок 2: Диаграмма прецедентов с точки зрения пользователя

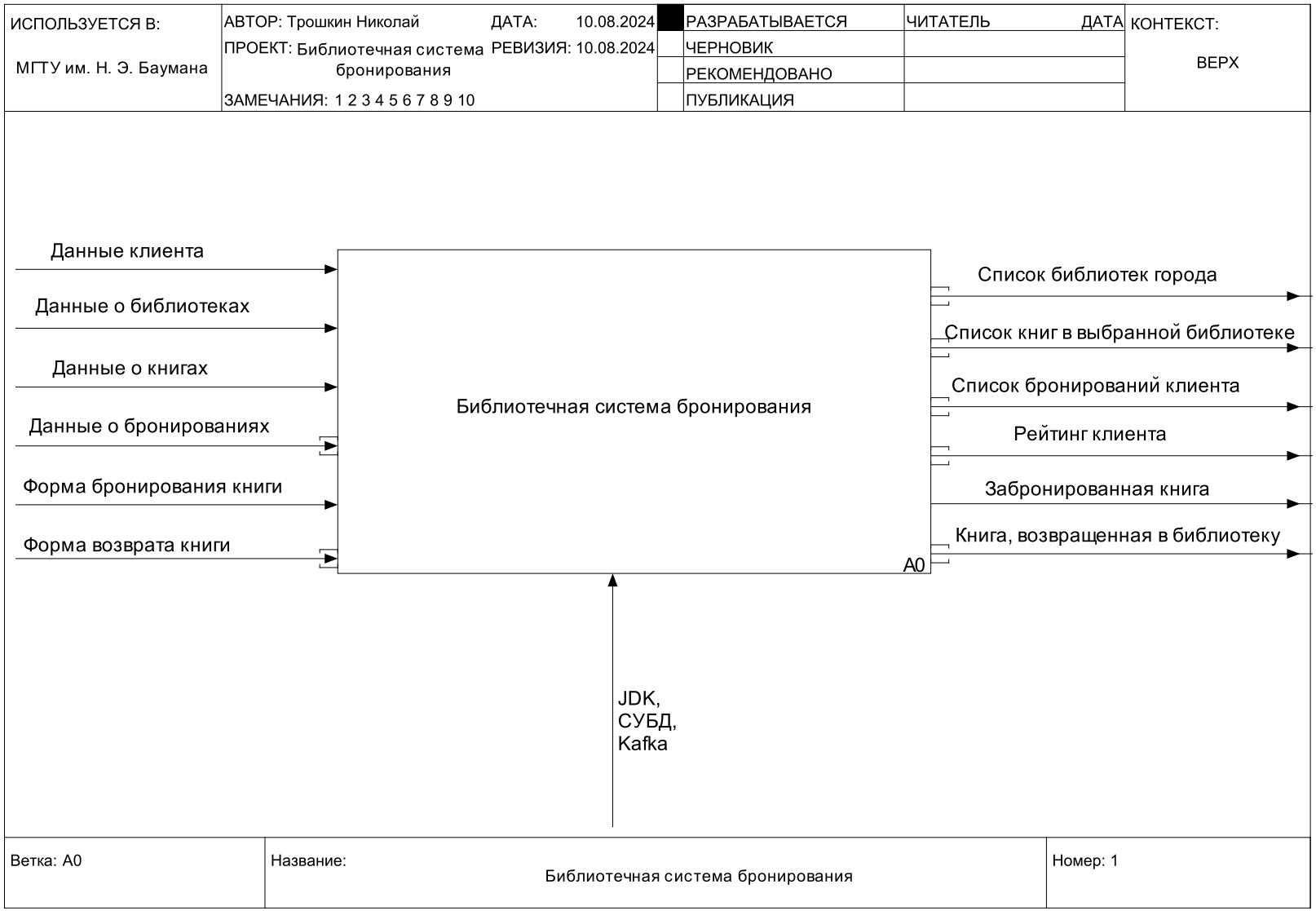

Рисунок 3: Концептуальная модуль системы в нотации IDEF0.

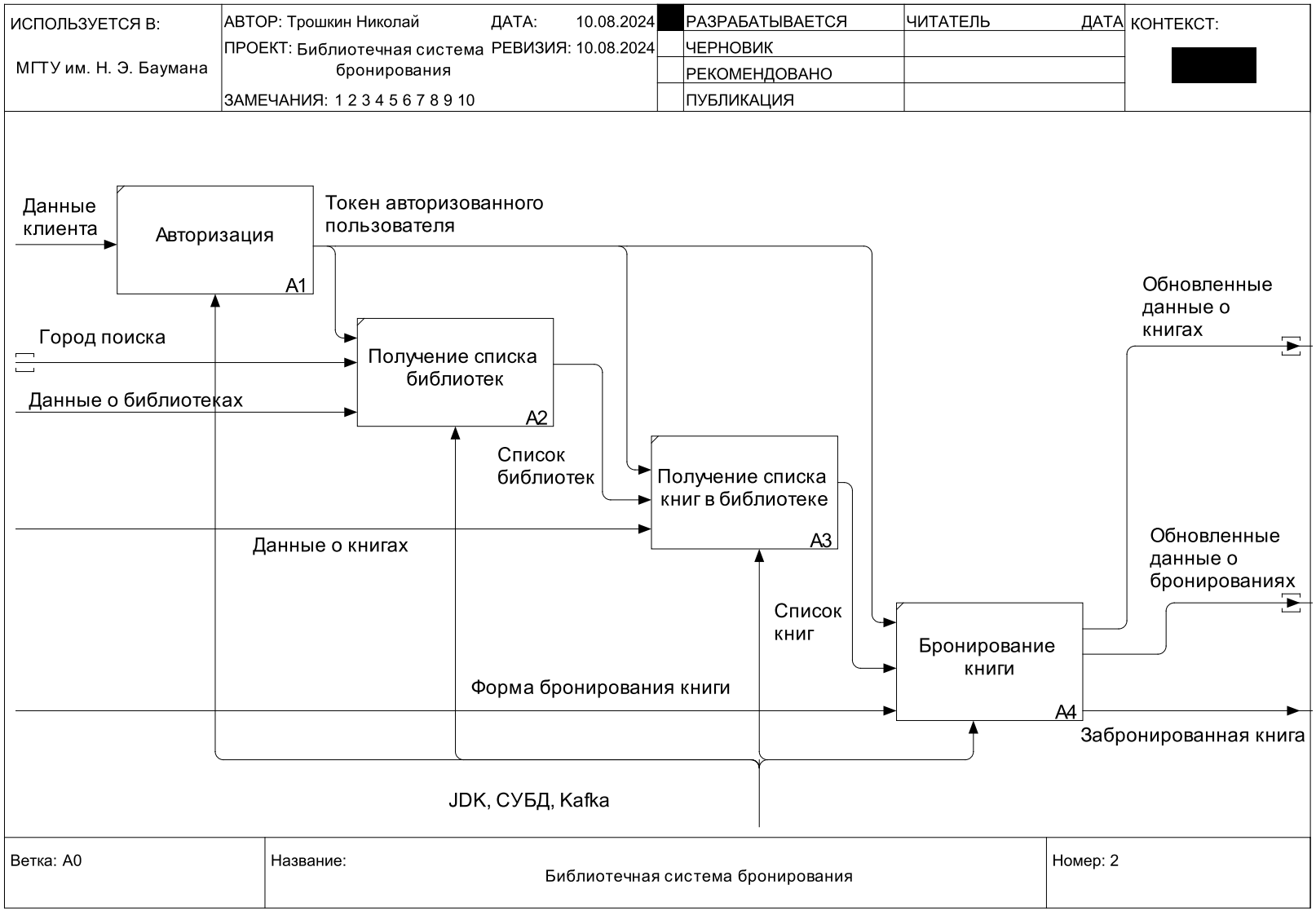

Рисунок 4: Детализированная концептуальная модель системы в нотации IDEF0.

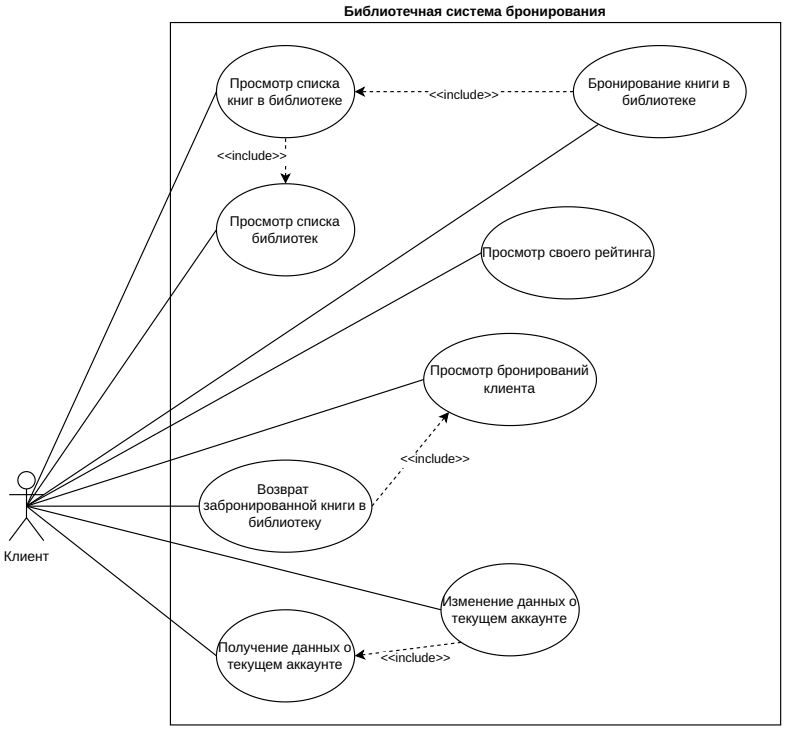

Рисунок 5: Диаграмма прецедентов с точки зрения клиента.

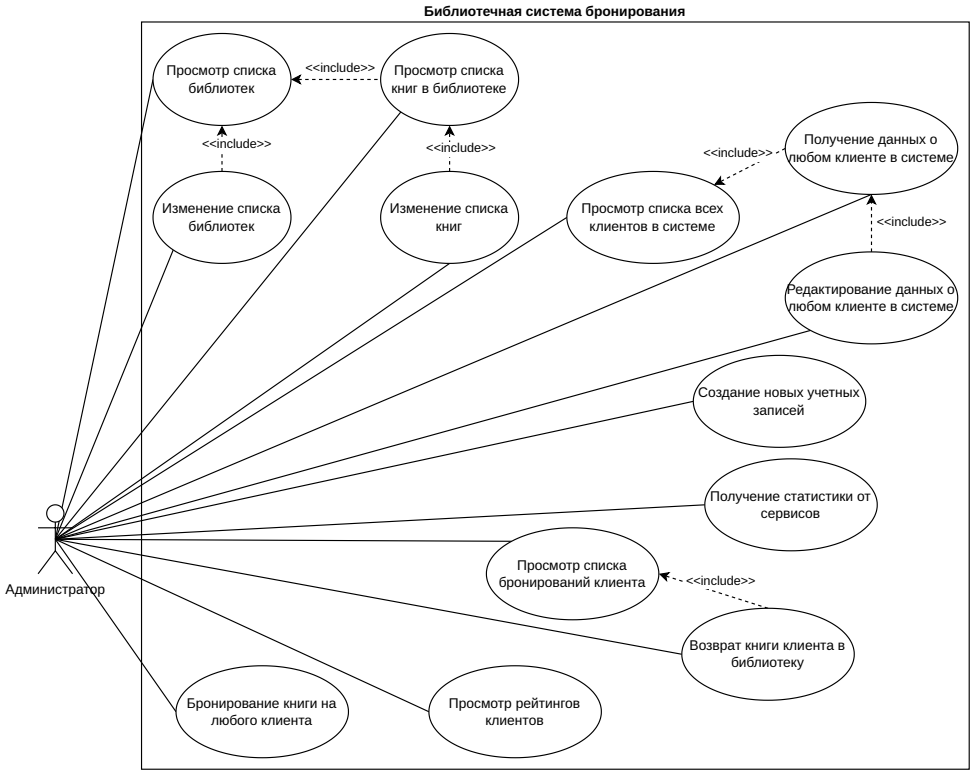

Рисунок 6: Диаграмма прецедентов с точки зрения администратора.

На рисунке 7 представлена диаграмма деятельности в режиме «Бронирование книги» для клиента.

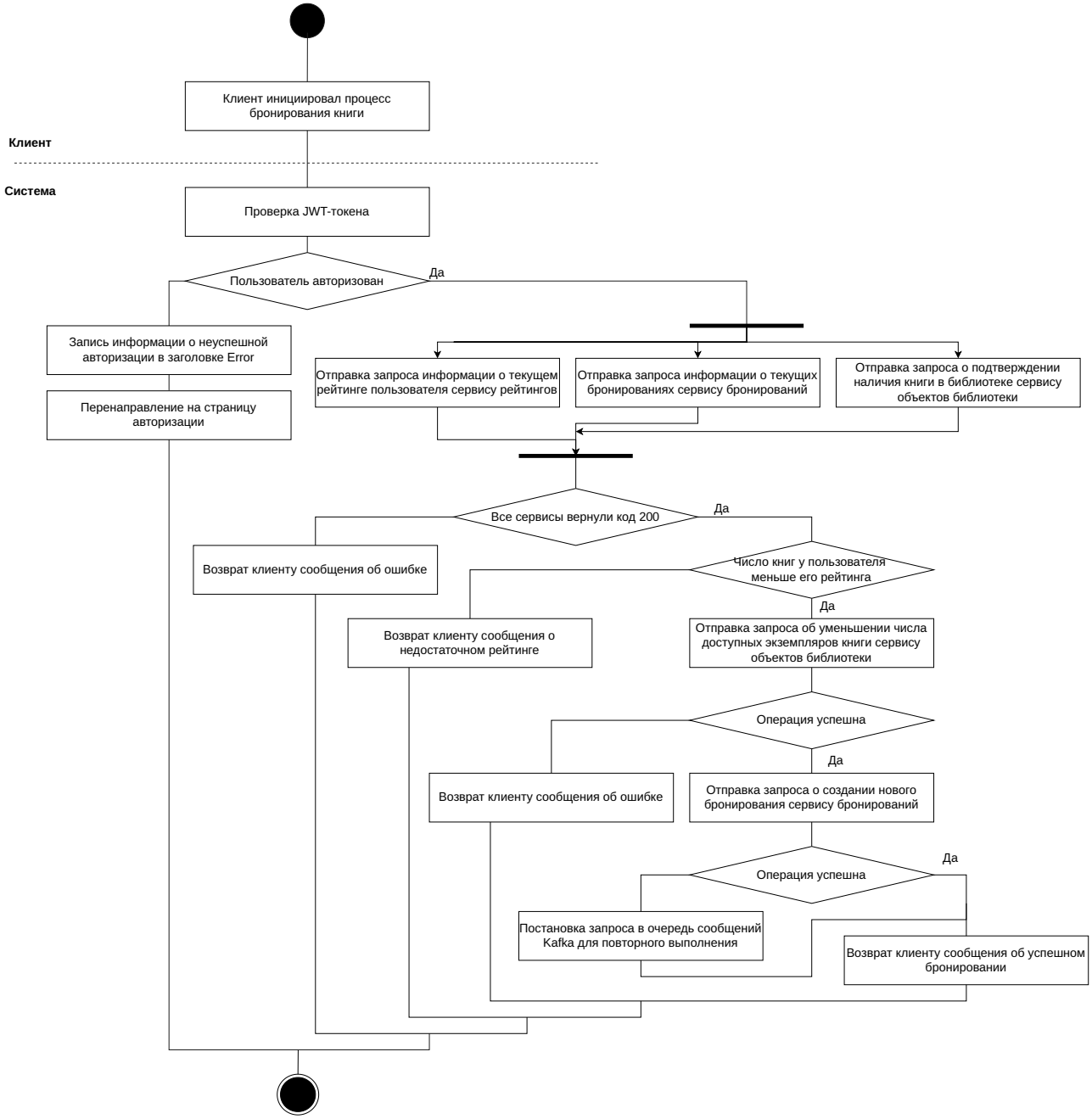

Рисунок 7: Диаграмма деятельности в режиме «Бронирование книги» для клиента.

На рисунке 8 представлена наиболее важная для системы динамическая модель, представленная в виде диаграммы последовательности действий.

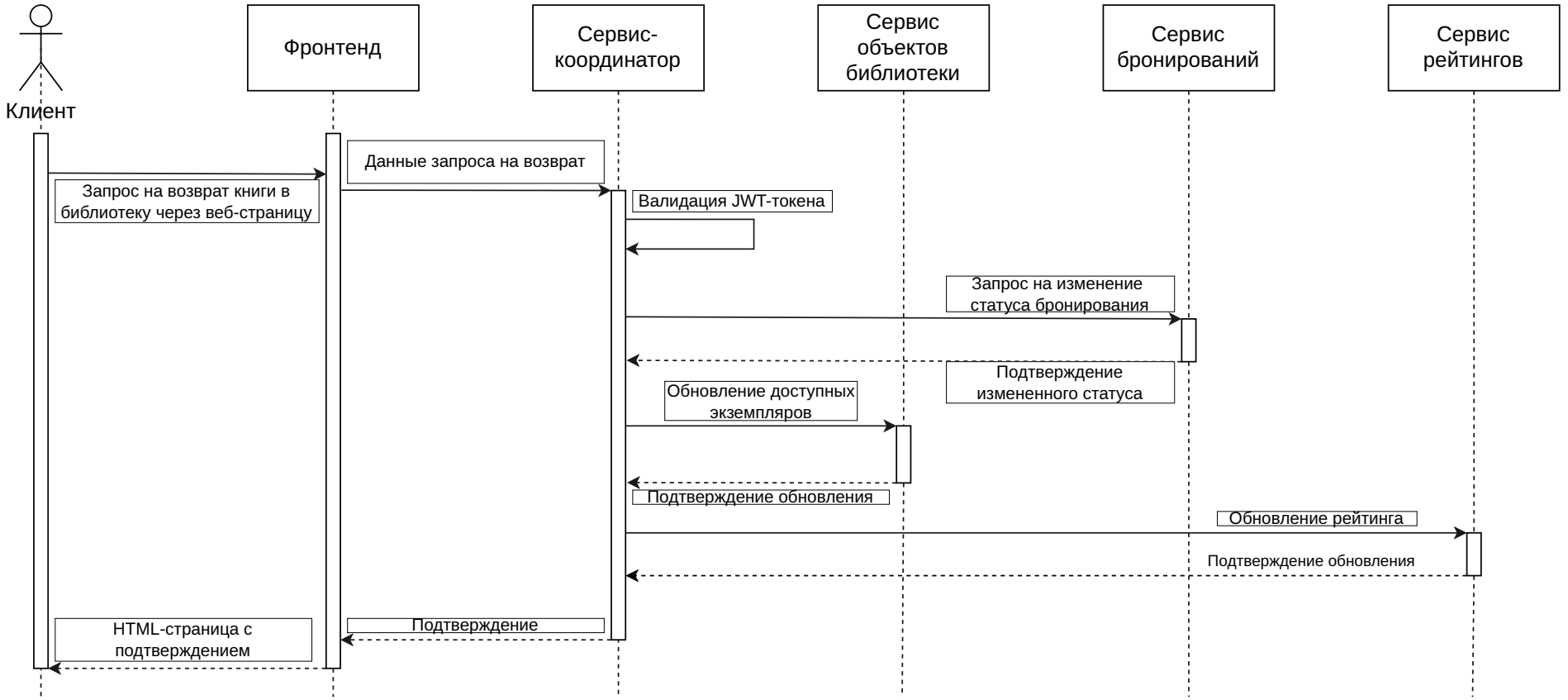

Рисунок 8: Диаграмма последовательности действий при возврате книги клиентом.

Диаграмма потоков данных, представленная на рисунке 9, позволяет описать распределение сохраняемой информации в хранилищах данных. Каждое хранилище данных будет реализовано в виде базы данных.

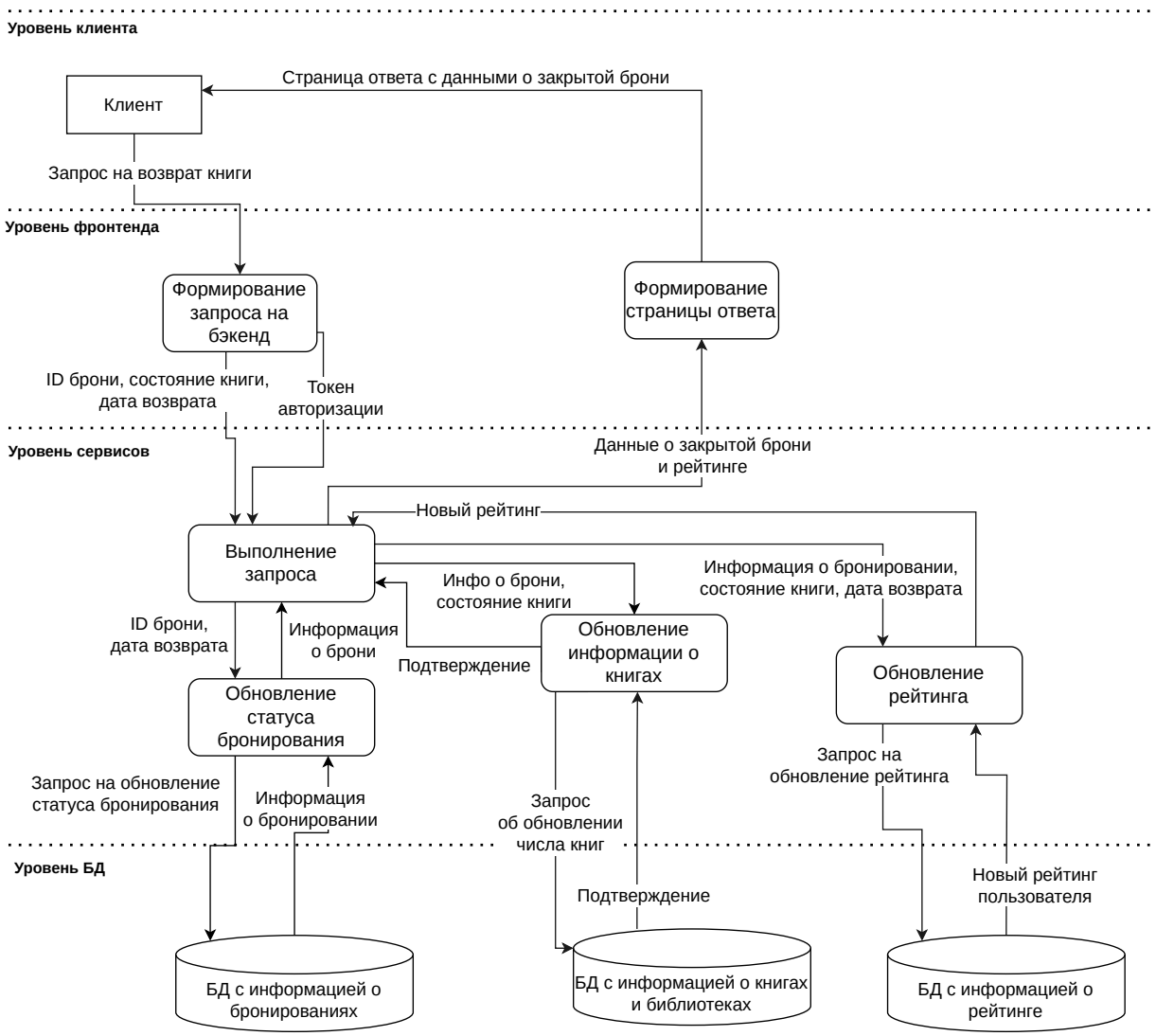

Рисунок 9: Диаграмма потоков данных при возврате книги.

2.3 Спецификация классов

На рисунке 10 представлена диаграмма классов, принадлежащих основным компонентам микросервиса объектов библиотек.

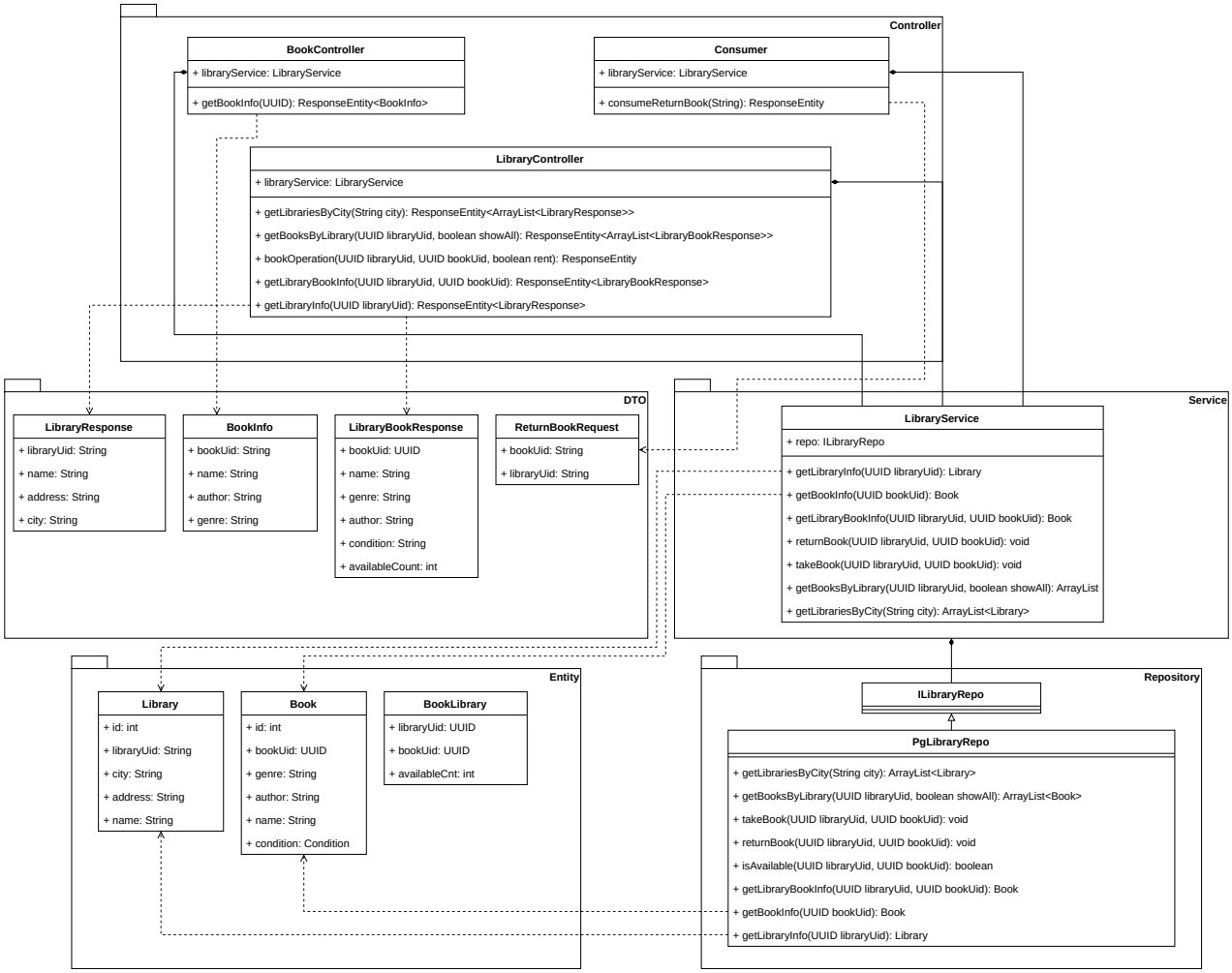

Рисунок 10: Диаграмма классов

Классы Library, Book, BookLibrary представляют легковесные объекты бизнес-логики, ассоциированные с соответствующими сущностями базы данных. Атрибуты указанных классов представлены в таблицах 4-6.

Таблица 4: Атрибуты класса Library

Атрибуты		
Имя	**Тип**	**Описание**
id	private: int	идентификатор библиотеки в БД
libraryUid	private: String	UUID библиотеки
address	private: String	адрес
city	private: String	город
name	private: String	название библиотеки

Таблица 5: Атрибуты класса Book

Атрибуты		
Имя	**Тип**	**Описание**
id	private: int	идентификатор книги в БД
bookUid	private: UUID	UUID книги
name	private: String	название книги
author	private: String	автор
genre	private: String	жанр книги
condition	private: Condition	состояние книги (enum: плохое, хорошее, отличное)

Таблица 6: Атрибуты класса BookLibrary

Атрибуты		
Имя	**Тип**	**Описание**
libraryUid	private: UUID	UUID библиотеки
bookUid	private: UUID	UUID книги
availableCnt	private: int	количество доступных экземпляров данной книги в данной библиотеке

Классы LibraryResponse, BookInfo, LibraryBookResponse, ReturnBookRequest представляют объекты для передачи данных

между сервисами. Атрибутивный состав LibraryResponse, BookInfo, LibraryBookResponse аналогичен составу ассоциированных с ними сущностей базы данных.

Класс PgLibraryRepo отвечает за взаимодействие микросервиса с базой данных. Его методы приведены в таблице 7.

Таблица 7: Атрибуты класса PgLibraryRepo

Методы	
Название	**Описание**
getLibrariesByCity(String): Library[]	param: [String - in] - город *получение информации о библиотеках в городе*
getBooksByLibrary(UUID, boolean): Book[]	param: libraryUid [UUID - in] - идентификатор библиотеки param: showAll [boolean - in] - флаг, если установлен, получить даже книги, которых нет в наличии *получение информации о фитнес-клубе по его идентификатору*
takeBook(UUID, UUID): void	param: libraryUid [UUID - in] - идентификатор библиотеки param: bookUid [UUID - in] - идентификатор книги *уменьшение числа доступных экземпляров взятой книги в выбранной библиотеке*

Продолжение на следующей странице

Название	**Описание**
returnBook(UUID, UUID): void	param: libraryUid [UUID - in] - идентификатор библиотеки param: bookUid [UUID - in] - идентификатор книги *увеличение числа доступных экземпляров возвращенной книги в библиотеке*
isAvailable(UUID, UUID): boolean	param: libraryUid [UUID - in] - идентификатор библиотеки param: bookUid [UUID - in] - идентификатор книги *проверка наличия книги в библиотеке*
getLibraryBookInfo(UUID, UUID): Book	param: libraryUid [UUID - in] - идентификатор библиотеки param: bookUid [UUID - in] - идентификатор книги *получение полной информации о книге в библиотеке*
getBookInfo(UUID): Book	param: bookUid [UUID - in] - идентификатор книги *получение полной информации о книге*
getLibraryInfo(UUID): Library	param: libraryUid [UUID - in] - идентификатор библиотеки *получение полной информации о библиотеке*

Класс LibraryService реализует бизнес-логику микросервиса, преобразование данных и передачу их на последующий слой, непосредственно связан-

ный с базой данных, поэтому предоставляемые им методы схожи с методами класса PgLibraryRepo.

Классы LibraryController, BookController выполняют функцию преобразования транспортных сущностей в сущности базы данных и передачу запроса на уровень сервисов, а также формирование HTTP-ответа. Класс Consumer отвечает за обработку очереди сообщений в Kafka, куда попадают не обработанные сразу запросы о возврате книги.

3 Технологический раздел

3.1 Средства разработки

3.1.1 Выбор СУБД

В соответствии с техническим заданием разработка бекенда предусматривает следующие требования.

— **Безопасность хранения данных**. Несанкционированный доступ к данным клиентам должен быть невозможен.

— **Транзакционность**. Должен соблюдаться принцип «ACID» (Atomicity — Атомарность, Consistency — Согласованность, Isolation — Изолированность, Durability — Надежность). Атомарность гарантирует, что транзакция не может быть зафиксирована частично. Согласованность — что успешное завершение транзакции оставит систему в согласованном состоянии. Изолированность — что параллельно выполняемые транзакции не будут влиять друг на друга. Надежность — что успешно завершенная транзакция будет зафиксирована, а в случае сбоя, после восстановления системы, результаты транзакции не будут утеряны.

— **Масштабируемость**. Выбранная СУБД должна поддерживать репликацию, шардирование.

PostgreSQL [10] — реляционная система управления базами данных. Она является некоммерческим ПО с открытым исходным кодом. Для работы с этой СУБД существуют библиотеки для таких распространенных языков программирования, как Python, Ruby, Perl, PHP, C, C++, Java, C#, Go. Она работает под управлением многих операционных систем: Linux, MacOS, Windows, Solaris и др. По сравнению с MySQL система PostgreSQL лучше работает с репликацией, так как в ней существует журнал (средство восстановления системы в случае сбоя) физической модификации страниц. PostgreSQL осуществляет асинхронную репликацию типа «ведущий — ведомый».

Выбор СУБД PostgreSQL для хранения данных разрабатываемой системы обеспечит надежность, безопасность и масштабируемость.

3.1.2 Средства разработки Identity Provider

Для реализации функций Identity Provider используется Keycloak [11]. Это открытая платформа для управления идентификацией и доступом, которая позволяет разработчикам и администраторам легко управлять пользователями, ролями и доступом к различным приложениям. На Keycloak переложены аутентификация, хранение данных о пользователях и управление сессиями. Получение токенов, создание пользователей администратором и выход из учетной записи выполняются сервисом авторизации путем отправки соответствующих HTTP-запросов в Keycloak.

3.1.3 Средства развертывания приложения

Первоначально для развертывания приложения была выбрана платформа VK Cloud [12]. Это облачная платформа, предоставляемая компанией VK, которая предлагает различные облачные сервисы и решения для разработчиков. Платформа ориентирована на предоставление гибких и масштабируемых инфраструктурных ресурсов, таких как вычислительные мощности, хранилища данных, базы данных, сетевые решения и инструменты для развертывания приложений, в частности, управляемые кластеры Kubernetes. Однако на каждого пользователя выделяется квота в 4 машины и 16 ГБ суммарной оперативной памяти. На мастер-узел кластера требуется минимум 6 ГБ RAM. На рабочих узлах требовалось развернуть 15 подов — 6 сервисов, 4 базы данных, 3 пода для работы Kafka, клиентская часть приложения и Keycloak. Были проверены следующие конфигурации:

1. 2 рабочих узла с 4 ГБ оперативной памяти в каждом. Конфигурация не подошла, так как на ней можно было развернуть максимум 8 подов, этого не хватает даже, чтобы развернуть только бэкенд (все остальное можно было поднять на свободной машине в контейнере Docker).

2. 3 рабочих узла с 2 ГБ оперативной памяти в каждом. Такая конфигурация также не подошла, так как узлы не выдерживали нагрузки и их требовалось постоянно перезагружать.

3. Остальные возможные конфигурации выходили за рамки квот.

Таким образом, пришлось отказаться от разворачивания приложения в облаке и для деплоя был выбран minikube [13], который позволяет развернуть кластер Kubernetes локально.

3.2 Интерфейс программы

На рисунке 11 приведен пример web-страницы интерфейса разработанного приложения.

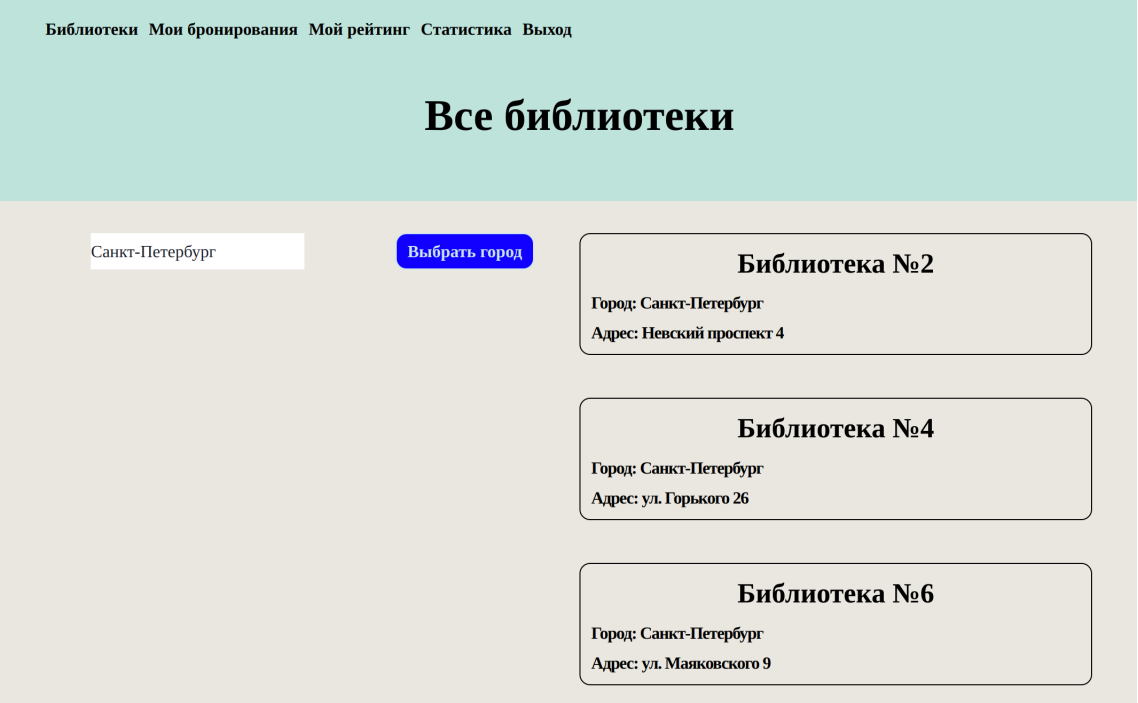

Рисунок 11: Интерфейс приложения

В приложении реализована деградация функциональности. Например, в тестовых данных были бронирования, для которых указана несуществующая библиотека. При запросе дополнительной информации о библиотеке в сервисе библиотек срабатывало исключение и возвращалась 500 ошибка. При этом страница отобразилась корректно, остались лишь пропуски на месте несуществующей библиотеки, как показано на рисунке 12.

Также была реализована ролевая модель. Роль пользователей считывается сервисами из JWT-токена и выполняется проверка соответствия роли пользователя той роли, которой разрешен запрос. Например, сервис статистики доступен только администратору. Клиент при запросе статистики увидит сообщение, приведенное на рисунке 13.

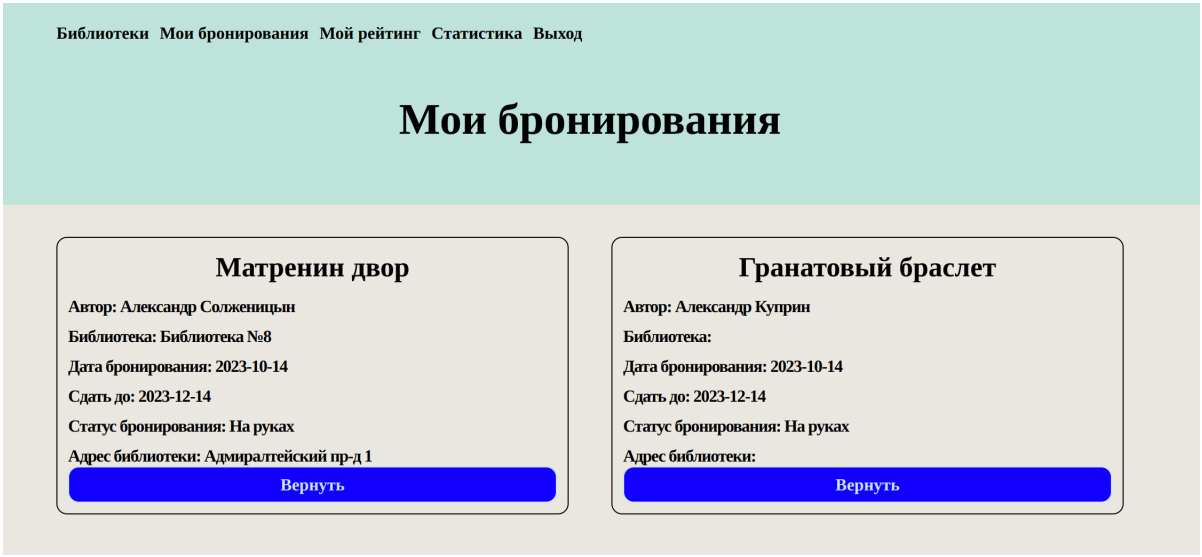

Рисунок 12: Деградация функциональности при ошибке в сервисе библиотек

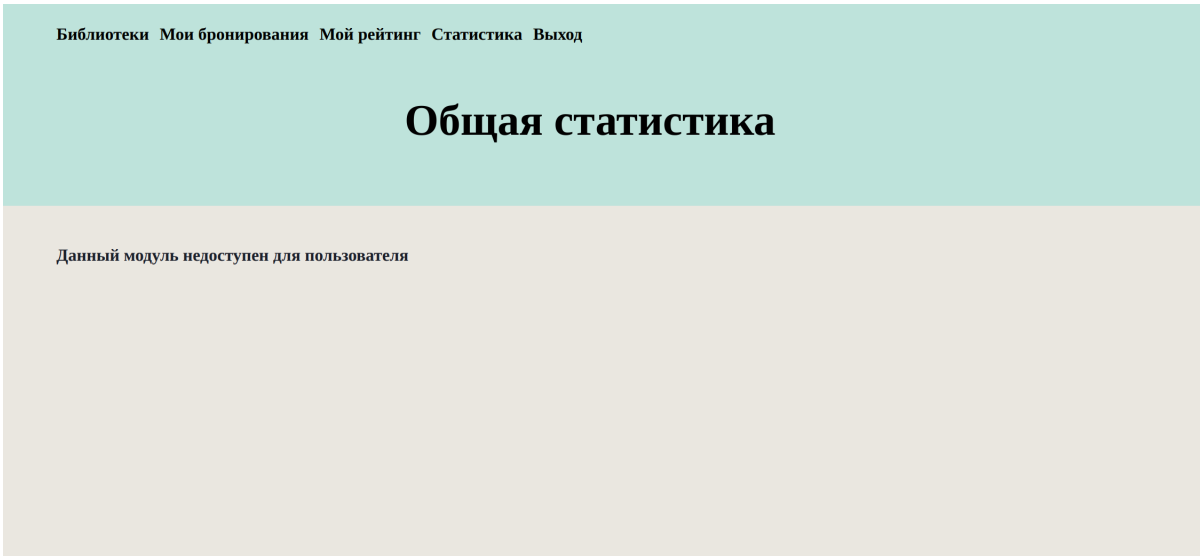

Рисунок 13: Сообщение о недостатке прав для обычного пользователя

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы:

— сформулированы основные требования к системе и ее подсистемам;

— описана архитектура системы в виде диаграмм в нотациях UML и IDEF0, а также указаны некоторые сценарии её функционирования;

— разработано, развернуто и протестировано программное обеспечение, выполняющее основные заявленные требования и функциональность.

Таким образом, была разработана распределенная система для бронирования книг в библиотеках.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое REST API [Электронный ресурс]. Режим доступа: https://ru.hexlet.io/blog/posts/chto-takoe-rest-api (дата обращения: 27.04.2024).
2. Circuit Breaker pattern — Azure Architecture Center | Microsoft Learn [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker (дата обращения: 17.08.2024).
3. JSON Web Token Introduction — jwt.io [Электронный ресурс]. Режим доступа: https://jwt.io/introduction (дата обращения: 17.08.2024).
4. Kubernetes Documentation | Kubernetes [Электронный ресурс]. Режим доступа: https://kubernetes.io/docs/home/ (дата обращения: 17.08.2024).
5. Ingress Controllers | Kubernetes [Электронный ресурс]. Режим доступа: https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/ (дата обращения: 17.08.2024).
6. Helm | Docs [Электронный ресурс]. Режим доступа: https://helm.sh/docs/ (дата обращения: 17.08.2024).
7. Final: OpenID Connect Core 1.0 incorporating errata set 2 [Электронный ресурс]. Режим доступа: https://openid.net/specs/openid-connect-core-1_0.html (дата обращения: 21.08.2024).
8. RFC 6749 — The OAuth 2.0 Authorization Framework [Электронный ресурс]. Режим доступа: https://datatracker.ietf.org/doc/html/rfc6749#section-1.3.1 (дата обращения: 21.08.2024).
9. Apache Kafka [Электронный ресурс]. Режим доступа: https://kafka.apache.org/documentation/ (дата обращения: 17.08.2024).
10. PostgreSQL: Documentation [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/ (дата обращения: 20.06.2024).

11. Documentation — Keycloak [Электронный ресурс]. Режим доступа: https://www.keycloak.org/documentation (дата обращения: 20.08.2024).

12. VK Cloud | Облачная ИТ-платформа бизнес-класса от VK [Электронный ресурс]. Режим доступа: https://cloud.vk.com/ (дата обращения: 26.08.2024).

13. Welcome! | minikube [Электронный ресурс]. Режим доступа: https://minikube.sigs.k8s.io/docs/ (дата обращения: 26.08.2024).