

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  
Московский авиационный институт  
(национальный исследовательский университет)

Институт № 8  
Информационных технологий и прикладной математики

Кафедра 813 «Компьютерная математика»

КУРСОВАЯ РАБОТА  
по дисциплине «Базы данных»

на тему: Разработка клиент-серверного БД-ориентированного ПО  
«Книжный магазин»

**Выполнила:** студентка группы М8О-310Б-19

---

Байтякова Анастасия Викторовна

---

(Фамилия, имя, отчество)

---

(подпись)

**Принял:** доцент кафедры 813

---

Чумакова Екатерина Витальевна

---

(Фамилия, имя, отчество)

---

(подпись)

**Оценка:**

**Дата:**

Москва, 2021

# Содержание

<b>I</b>	<b>Теоретическая часть</b>	<b>4</b>
<b>1</b>	<b>Клиент-серверное ПО «Книжный магазин»</b>	<b>4</b>
1.1	Техническое задание . . . . .	4
1.2	Структура ПО . . . . .	6
<b>2</b>	<b>Анализ предметной области</b>	<b>6</b>
2.1	Информационная модель проектируемой системы . . . . .	7
<b>3</b>	<b>Функциональные задачи клиента</b>	<b>7</b>
<b>II</b>	<b>Практическая часть</b>	<b>7</b>
<b>4</b>	<b>Реляционная модель данных системы</b>	<b>8</b>
4.1	Общая схема таблиц в базе данных . . . . .	8
4.2	Таблицы . . . . .	9
4.2.1	Таблица ages . . . . .	9
4.2.2	Таблица author . . . . .	9
4.2.3	Таблица publisher . . . . .	9
4.2.4	Таблица genre . . . . .	9
4.2.5	Таблица book . . . . .	9
4.2.6	Таблица roles . . . . .	9
4.2.7	Таблица customer . . . . .	10
4.2.8	Таблица city . . . . .	10
4.2.9	Таблица promo_code . . . . .	10
4.2.10	Таблица way_of_delivery . . . . .	10
4.2.11	Таблица buy . . . . .	10
4.2.12	Таблица buy_book . . . . .	10
4.3	Представления . . . . .	11
4.3.1	Представление book_info . . . . .	11
4.4	Хранимые процедуры и функции . . . . .	11
4.4.1	Функция aft_update . . . . .	11

<b>5</b>	<b>Программная реализация системы</b>	<b>11</b>
5.1	Протокол общения пользовательского клиента и сервера <b>S</b>	11
5.2	Протокол общения клиента-админа системы и сервера <b>S</b>	13
5.3	Протокол общения сервера <b>S</b> и базы данных <b>DB</b>	13
5.4	Структура приложения	15
5.5	Основные алгоритмы	16
5.6	Руководство пользователя	17
<b>6</b>	<b>Список использованных источников</b>	<b>25</b>
<b>7</b>	<b>Приложение</b>	<b>26</b>
7.1	Скрипты создания таблиц	26
7.1.1	Создание таблицы ages	26
7.1.2	Создание таблицы author	26
7.1.3	Создание таблицы publisher	26
7.1.4	Создание таблицы genre	26
7.1.5	Создание таблицы book	27
7.1.6	Создание таблицы roles	27
7.1.7	Создание таблицы customer	27
7.1.8	Создание таблицы city	28
7.1.9	Создание таблицы promo_code	28
7.1.10	Создание таблицы ways_of_delivery	28
7.1.11	Создание таблицы buy	28
7.1.12	Создание таблицы buy_book	29
7.2	Скрипты создания представлений	29
7.2.1	Создание представления book_info	29
7.3	Скрипты создания хранимых процедур и функций	29
7.3.1	aft_update	29
7.4	ПО «Книжный магазин»	30
7.4.1	Код сервера	30
7.4.2	Код пользовательского клиента	43
7.4.3	Код клиента-администратора системы	50

# Часть I

## Теоретическая часть

### 1 Клиент-серверное ПО «Книжный магазин»

#### 1.1 Техническое задание

1. Необходимо выбрать предметную область для создания базы данных. Выбранная предметная область должна быть уникальной для всего потока, а не только в рамках учебной группы.
2. Необходимо описать таблицы и их назначение. Выполнить проектирование логической структуры базы данных. Описать схему базы данных.
3. Ограничения на техническую реализацию:
  - (a) Должна быть использована СУБД PostgreSQL версии 13 и выше.
  - (b) Все реальные таблицы должны иметь 3 нормальную форму или выше.
  - (c) База данных должна иметь минимум 5 содержательных таблиц. В них не входят таблицы справочники (вида ключ – значение), наследуемые таблицы ([1] п. 5.10) считаются одной таблицей, так же как все секции секционированной таблицы ([1] п. 5.11)
  - (d) В каждой таблице должен быть первичный ключ, который рекомендуется делать типом Serial или родственными ему.
  - (e) Таблицы должны быть связаны внешними ключами.
  - (f) Должен присутствовать обоснованный пример использования наследования и секционирования таблиц.
  - (g) При разумных ограничениях на значения полей необходимо в структуры таблиц прописывать ограничения на значения атрибутов.
  - (h) Должны быть содержательные запросы с применением вложенных запросов, агрегатных функций, Group By, Union и Join.
  - (i) Необходимо создать и применить свою агрегатную функцию ([1] п. 38.12).

- (j) База данных должна иметь представления ([1] п. 41.2), материализованные представления ([1] п. 41.3), триггеры ([1] п. 39) и хранимые процедуры ([1] п. 43), причём все эти объекты должны быть осмысленны, а их использование оправдано.
  - (k) Должно быть реализовано каскадное удаление или обновление данных при срабатывании триггера. Использование должно быть обосновано.
  - (l) Хотя бы в одной хранимой процедуре должен быть использован язык программирования PL/pgSQL с использованием условий и циклов.
  - (m) Клиент БД должен выполнять запросы в БД через курсоры ([1] п. 43.7).
  - (n) Таблицы должны использовать индексы ([1] п. 11).
  - (o) Доступ к таблицам должен быть разграничен на уровне прав ([1] п. 5.7) и политик доступа к строкам ([1] п. 5.8).
  - (p) Вся структура БД должна храниться в SQL файлах.
  - (q) Данные для тестовой работы с БД также должны быть представлены в SQL.
4. Необходимо разработать два клиентских приложения для доступа к базе данных. Данные приложения должны быть написаны на двух разных языках программирования и иметь разный интерфейс (например, классическое оконное приложение и web-приложение). Выбор языков программирования произволен: C/C++, python, perl, ruby, JavaScript, php, swift, C# 5.0, Java и др.
5. Необходимо организовать различные роли пользователей и права доступа к данным. Далее, необходимо реализовать возможность создания архивных копий и восстановления данных из клиентского приложения.
6. При разработке базы данных следует организовать логику обработки данных не на стороне клиента, а, например, на стороне сервера, базы данных, клиентские приложения служат только для представления данных и тривиальной обработки данных.
7. При показе вашего проекта необходимо уметь демонстрировать таблицы, представления, триггеры и хранимые процедуры базы данных, внешние ключи, ограничения целостности и др. В клиентских приложениях уметь демонстрировать

подключение к базе данных, основные режимы работы с данными (просмотр, редактирование, обновление...)

8. Необходимо реализовать корректную обработку различного рода ошибок, которые могут возникать при работе с базой данных.

## 1.2 Структура ПО

Клиент-серверное приложение должно иметь следующую структуру:

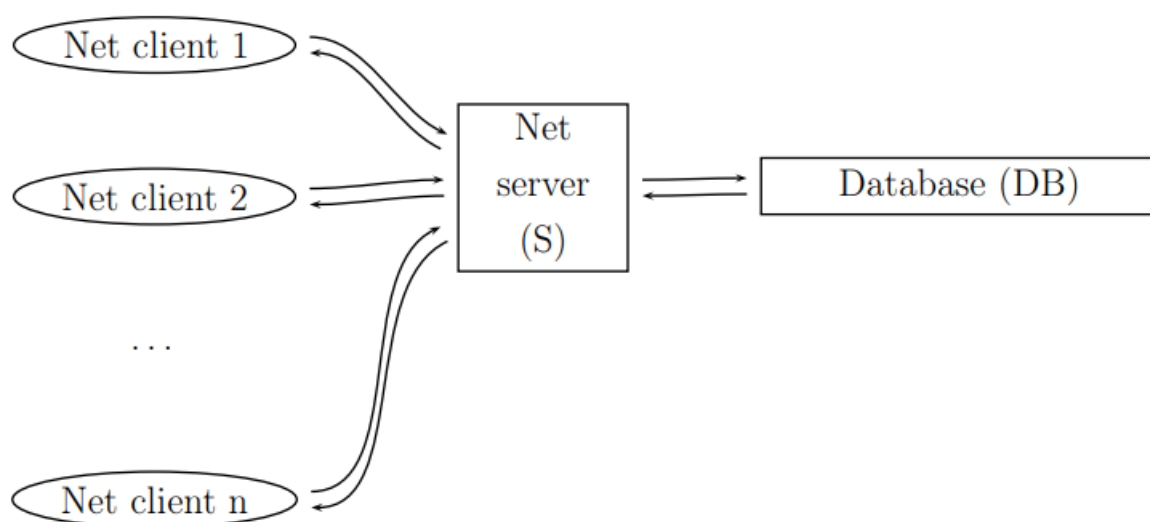


Рис. 1: Структура программного комплекса.

Сетевой сервер (S) должен принимать запросы от различных клиентов и использует базу данных (DB) для ответа на эти запросы.

Клиенты не должны иметь представление о внутренней структуре DB, структуре таблиц, хранимых процедурах и всех взаимосвязях между данными.

## 2 Анализ предметной области

Была смоделирована обобщенная модель книжных магазинов и выделены возможные функции действий клиентов, а также администраторов. Предполагаемый функционал магазина: подбор клиентом книг и их покупка, выбор способа и адреса доставки, также личный кабинет пользователя и возможность редактирования данных. Возможен поиск книг по нескольким параметрам. Создание отдельной страницы для

администратора с возможностью добавления новых книг и редактирования их количества, а также возможность редактирования ролей пользователей.

## **2.1 Информационная модель проектируемой системы**

В моделируемой системе будет несколько основных сущностей, такие как книга, клиент, заказ и покупка.

Сущность книги содержит атрибуты с основной информацией о книге (название, автор, жанр, цена и др.).

Сущность клиента содержит атрибуты ФИО, пароль и email.

Сущность покупки предоставляет информацию о городе и способе доставки, а также информацию о покупателе.

Сущность заказа объединяет в себе две сущности: покупки и книги, а также содержит дату покупки.

## **3 Функциональные задачи клиента**

Для обычного пользователя:

- Поиск книг по заданным параметрам
- Покупка книги с вводом информации о доставке
- Редактирование данных (пароля и email) в личном кабинете

Для пользователя с правами администратора:

- Все возможности обычного пользователя
- Добавление новых книг
- Изменение количества книг в продаже, удаление книг
- Редактирование ролей всех зарегистрированных пользователей

# Практическая часть

## 4 Реляционная модель данных системы

#### 4.1 Общая схема таблиц в базе данных

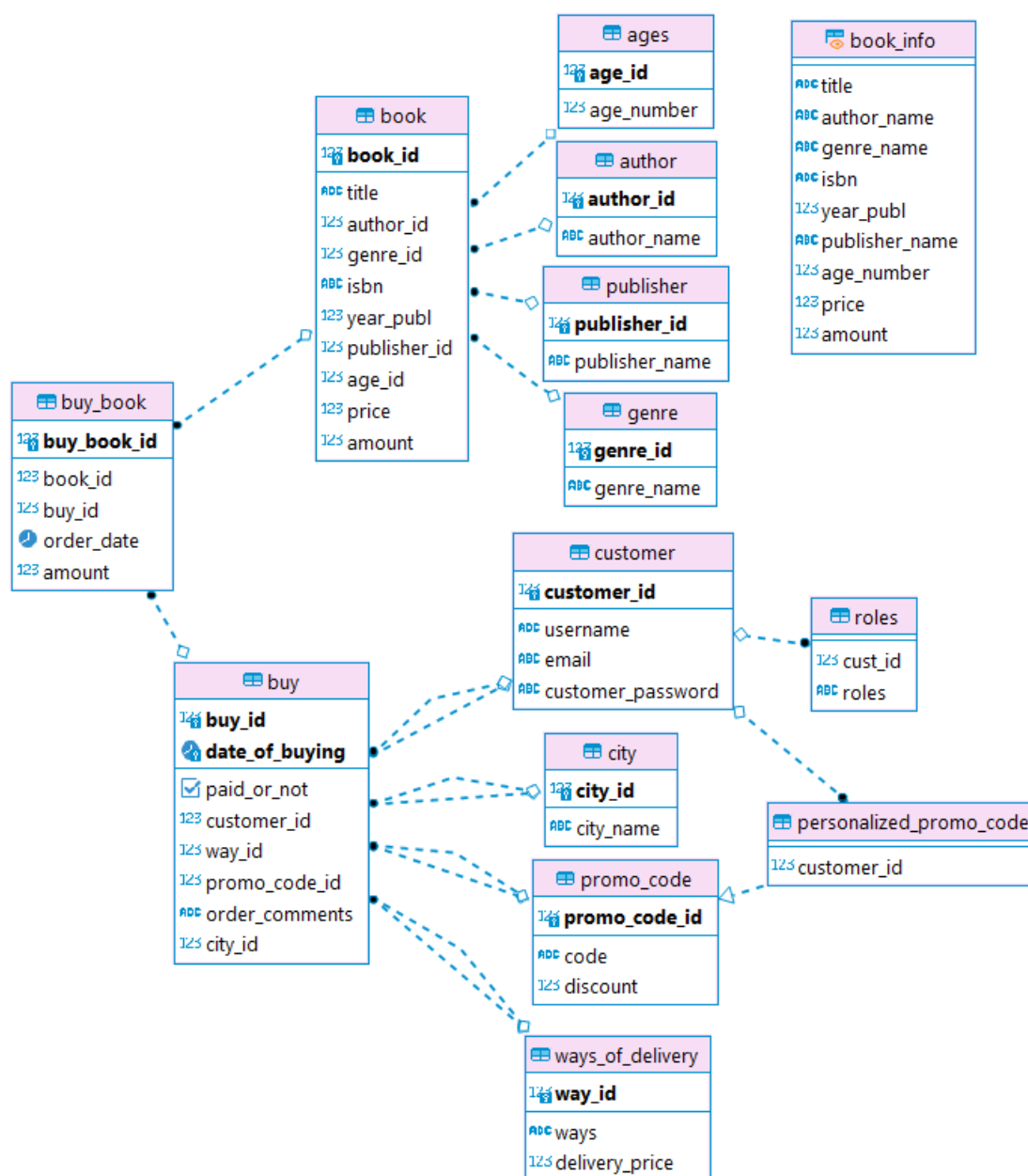


Рис. 2: Общая схема БД со связями между таблицами.



## 4.2 Таблицы

DDL код создания таблиц можно найти в разделе [7.1](#).

### 4.2.1 Таблица ages

Таблица для хранения возрастных ограничений. Содержит уникальный идентификатор - первичный ключ, возрастное ограничение.

Все последующие таблицы хранят уникальный идентификатор - первичный ключ.

### 4.2.2 Таблица author

Содержит ФИО автора книг.

### 4.2.3 Таблица publisher

Содержит название издательства.

### 4.2.4 Таблица genre

Содержит название жанра книг.

### 4.2.5 Таблица book

Таблица хранит полную информацию о книге и внешними ключами связана с таблицей авторов, жанров, издательства и возрастных ограничений.

Также в ней содержится информация о названии, isbn (международный стандартный номер книги), годе публикации, цене и количестве книг в наличии. Все поля таблицы не могут быть нулевыми.

Если количество книг становится равным нулю, срабатывает триггер на удаление книги. Данная функция вызывается при срабатывании триггера amount\_del.

### 4.2.6 Таблица roles

Таблица ролей связана с таблицей пользователей внешним ключом. Содержит в себе уникальный идентификатор пользователя и его роль. Возможные роли: USER, ADMIN.

#### **4.2.7 Таблица customer**

Содержит информацию о пользователе - его ФИО, email и пароль.

#### **4.2.8 Таблица city**

Название городов, в которые осуществляется доставка.

#### **4.2.9 Таблица promo\_code**

Таблица с промокодами, дающими скидки на покупку. Содержит в себе название промокода и размер скидки, которую он предоставляет.

Эту таблицу наследует таблица personalized\_promo\_code, которая содержит в себе уникальных идентификатор пользователя, владеющего этим промокодом.

#### **4.2.10 Таблица way\_of\_delivery**

Способы доставки книг и их стоимость.

#### **4.2.11 Таблица buy**

Таблица, содержащая в себе информацию о покупке. Имеет составной первичный ключ в котором содержатся уникальный идентификатор заказа и дата его оформления. Внешними ключами связана с таблицей пользователей, городов, промокодов и способов доставки. В данной таблице происходит секционирование по дате оформления заказа. Создаются секции по месяцу заказа.

Также содержит комментарии к заказу и статус оплаты.

#### **4.2.12 Таблица buy\_book**

Внешними ключами связана с таблицей книг и информации о покупке. Фактически является итоговой таблицей заказа, объединяющей в себе информацию о покупателе и книге, которую он приобретает. Также содержит в себе дату оформления заказа и итоговую сумму покупки.

## 4.3 Представления

DDL код создания представлений можно найти в разделе [7.2](#).

### 4.3.1 Представление book\_info

Содержит в себе все поля таблицы book с присоединенными к ней другими таблицами с помощью JOIN. В итоге можно получить полную информацию о книге.

## 4.4 Хранимые процедуры и функции

PL/pgSQL код описания функций можно найти в разделе [7.3](#).

### 4.4.1 Функция aft\_update

Данная функция вызывается при срабатывании триггера amount\_del (после обновления таблицы book) проверяется количество экземпляров в наличии для каждой книги. В случае равенства количества нулю книга удаляется иначе ничего не происходит.

## 5 Программная реализация системы

### 5.1 Протокол общения пользовательского клиента и сервера S

Общение клиента с сервером происходит посредством паттерна MVC.

Model-View-Controller (MVC, «Модель-Представление-Контроллер») — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Среда Spring Web Model-View-Controller (MVC) разработана на основе DispatcherServlet, который обрабатывает все HTTP-запросы и ответы. Последовательность событий, соответствующая входящему HTTP-запросу:

- После получения HTTP-запроса DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет, какой Контроллер должен быть вызван, после чего, отправляет запрос в нужный Контроллер.
- Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные Модели, основанные на определённой логике и возвращает в DispatcherServlet имя View.
- При помощи интерфейса ViewResolver DispatcherServlet определяет, какой Вид нужно использовать на основании полученного имени.
- После того как View создан, DispatcherServlet отправляет данные Модели в виде атрибутов во View, который в конечном итоге отображается в браузере.

Для упрощения понимания приведем пример конкретной ситуации общения клиента с сервером - приобретение книги. Когда клиент нажимает кнопку покупки книги, он переходит на страницу оформления заказа.

Контроллер для конкретного URL-адреса предоставляет методы GET и POST. С помощью метода GET в модель из базы данных заносятся все города и способы доставки, для дальнейшего отображения пользователю на странице оформления заказа. После заполнения пользователем всей информации и нажатия кнопки оплаты заказа происходит переход к методу POST. В нем все введенные пользователем данные заносятся в таблицу информации о покупке с применением созданных интерфейсов-репозиторий. Далее высчитывается сумма заказа, с применением возможной скидки и эти данные с текущей датой заказа заносятся в итоговую таблицу заказа. После этого пользователь перенаправляется на страницу о завершении покупки, содержащую номер заказа.

Все остальные действия, которые может выполнять пользователь (например редактирование профиля) происходят по подобному сценарию, описанному выше.

## 5.2 Протокол общения клиента-админа системы и сервера S

Общение клиента-админа с сервером практически не отличается от общения обычного пользователя. Но у админа есть собственная страница, к которой закрыт доступ обычным пользователям. Разграничение доступа происходит с помощью аннотации `@PreAuthorize("hasAuthority('ADMIN')")`. Таким образом указывается, что только пользователь с ролью администратора может получить доступ к данной странице. Если на такую страницу попытается зайти обычный пользователь вернется ошибка прав доступа 403. Администратор может редактировать роли пользователей, то есть назначать новых клиентов-админов, а также редактировать количество книг в наличии. Если количество книг будет установлено в 0, книга удалится.

```
@PreAuthorize("hasAuthority('ADMIN')")
@GetMapping("/{customer}")
public String userEditForm(@PathVariable Customer customer, Model model){
    model.addAttribute(attributeName: "user", customer);
    model.addAttribute(attributeName: "roles", Role.values());
    return "userEdit";
}
```

Рис. 3: Метод GET редактирования ролей пользователей.

В методе в модель добавляются пользователи и их роли, далее они отображаются на странице редактирования.

## 5.3 Протокол общения сервера S и базы данных DB

Каждая таблица базы данных представлена в виде класса. Аннотация `@Entity` позволяет представлять таблиц в виде Java-классов. Поля повторяют поля таблиц на стороне БД. Пример представления таблицы:

```

@Entity
@Table(name = "author")
@Data
public class Author{

    @Id
    @Column(name = "author_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long author_id;

    @Column(name = "author_name")
    private String author_name;

    public Author() {
    }

    public Author(String author_name) {
        this.author_name = author_name;
    }
}

```

Рис. 4: Таблица в виде Java-класса.

Построение запросов из сервера к базе реализовано через специальные интерфейсы-репозитории. Для каждой сущности создаются отдельные репозитории с методами без тела, а создает эти методы во время компиляции Spring Data.

Такие интерфейсы помечаются специальной аннотацией `@Repository`. Эта аннотация используется в классах Java, которые напрямую обращаются к базе данных. `@Repository` работает как маркер для любого класса, который выполняет роль хранилища или доступ к данным объекта. Эта аннотация имеет функцию автоматического перевода. Например, когда возникает исключение в `@Repository`, существует обработчик для этого исключения, и нет необходимости добавлять блок `try-catch`.

Все интерфейсы - репозитории наследуют интерфейс `JpaRepository`. `JpaRepository<сущность, тип_её_ID>` — включает все заготовленные запросы, чтобы обращаться к определённой сущности. Пример использования запросов, предоставленных `JpaRepository`:

```

@Repository
public interface PromoRepo extends JpaRepository<PromoCode, Long> {
    PromoCode findPromoCodeByCode(String code);
}

```

Рис. 5: Запрос из JpaRepository.

На вход запроса приходит строка, по которой запрос ищет в базе данных соответствующий промокод, если такой не найден возвращается null.

Также при помощи аннотации @Query можно писать собственные запросы на языке SQL. Подобный пример приведен ниже:

```
@Repository
public interface AuthorRepo extends JpaRepository<Author, Long> {
    @Query("select a from Author a where a.author_name=?1")
    Author findAuthorByAuthor_name(String name);
}
```

Рис. 6: Запрос при помощи аннотации @Query.

Написанный вручную запрос выполняет поиск автора по его ФИО.

## 5.4 Структура приложения

После создания базы данных происходит подключение к ней из сервера через localhost:5432. После запуска приложения происходит запуск сервера, и он начинает "слушать" порт 8075. Далее, введя в поисковую строку браузера localhost:8075, можно перейти на сайт книжного магазина. Если же сервер не был запущен или произошла ошибка, обращение к данному порту не даст никакой информации.

Неавторизованный пользователь увидит лишь начальную страницу без возможности приобретения книги. Новый пользователь может пройти регистрацию и если введенный email не был найден в базе данных, новый клиент будет зарегистрирован. Необходимо пройти авторизацию, и если все прошло успешно, произойдет переход на страницу с возможностью покупки книги. При всех дальнейших действиях происходит отправка HTTP-запросов на сервер. При помощи контроллеров для конкретных URL-адресов происходит обработка этих запросов и пользователь видит в браузере полученную информацию.

Всего используется 5 основных контроллеров.

1. RegistrationController - обработка всех действий связанных с регистрацией.
2. BuyBookController - покупка книг (вывод пользователю информации на странице оформления покупки, добавление заказа в базу данных и др.)

3. MainController - контроллер URL-адресов, по которым админ может добавлять книги и редактировать их количество.
4. UserController - изменение администратором ролей пользователей, а также редактирование пользователями личных данных
5. UserPageController - вывод пользователю списка книг, а также поиск по заданным параметрам

## 5.5 Основные алгоритмы

### 1. Регистрация и авторизация

Происходит с использованием Spring Security. При регистрации нового пользователя, производится проверка на существование такого пользователя в базе данных по email. Если метод addCustomer возвращает false, выводится сообщение, что такой пользователь уже существует. Иначе пользователь добавляется в базу данных, ему назначается роль USER и его пароль шифруется при помощи BCryptPasswordEncoder - криптографическая хеш-функция формирования ключа.

При авторизации посредством метода loadUserByUsername в базе данных ищется пользователь. Если найден - переход на главную страницу, иначе сообщение, что данный пользователь не найден.

### 2. Добавление новой книги

Входными параметрами в метод add являются все поля книги (название, автор, цена и т.д.). Каждый параметр проверяется на корректность (например не отрицательность цены и возрастного ограничения). Поля автор, издательство, возрастные ограничение и жанр сначала ищутся в своих таблицах при помощи репозиториев, если не найдены то они добавляются туда и возвращается объект каждой сущности. Далее книга создается, добавляется в базу данных и выводится на странице. В случае некорректности или незаполнения какого либо поля выводится сообщение об ошибке.

### 3. Редактирование количества книг



Входными параметрами в метод `updateAmount` является `id` выбранной для редактирования книги и новое количество экземпляров. Книга ищется в базе данных по `id` и ей устанавливается новое количество. В случае установки количества равным 0, книга сразу же удалится.

#### 4. Изменение данных в личном кабинете

На вход в метод `updateProfile` подается новый email и пароль (либо пустая строка если ничего не было введено). Происходит проверка на равенство старого email новому, и в случае их различия, новая почта пользователя сохраняется. Новый пароль сразу заносится зашифрованным. И обновленные данные о пользователе сохраняются в базу данных.

#### 5. Покупка книги

Входные параметры в метод `buy`: город доставки, комментарий к заказу, промокод, способ доставки, а также `id` покупаемой книги и имя пользователя. Книга ищется по `id` в базе данных, также ищутся город, способ доставки и промокод по строке. Создается объект информации о покупке.

Затем считается итоговая сумма заказа (цена книги + стоимость доставки). В случае правильности введенного промокода, применяется скидка. Итоговые данные передаются в базу данных. Пользователю выводится номер заказа и его финальная стоимость.

Реализация алгоритмов представлена в разделе приложения [7.4](#).

## 5.6 Руководство пользователя

После запуска приложения новому пользователю необходимо пройти регистрацию, после которой он сможет авторизоваться, если же пользователь был ранее зарегистрирован, то необходимо просто ввести данные для авторизации. В случае неудачной регистрации пользователь получит сообщение о том, что такой email уже существует. Нажав на кнопку с надписью "Книги" не авторизованный пользователь может увидеть список всех книг в наличии, но не может их приобрести.


BOOKS FOR EVERYONE Книги

Выйти

## Регистрация

Имя:

Email:

Пароль:  

Зарегистрироваться

[Уже зарегистрирован\(а\)](#)


Рис. 7: Регистрация нового пользователя.

BOOKS FOR EVERYONE Книги

Выйти

## Вход в личный кабинет

Имя:

Пароль:  

Войти

Еще нет аккаунта? [Зарегистрироваться](#)

Рис. 8: Авторизация пользователя.

В случае ошибки при авторизации, будет выведено сообщение.

Bad credentials

## Вход в личный кабинет

Имя:

Пароль:

Войти

Еще нет аккаунта? [Зарегистрироваться](#)

Рис. 9: Ошибка авторизации.

При успешной авторизации пользователь попадает на страницу с возможностью покупки книг.

BOOKS FOR EVERYONE Книги Профиль Выйти

Найти  Найти  Найти

**краткая история**  
Лив Стрём  
**Жанр:** Общество  
**Возрастное ограничение:** 12+  
**Издательство:** Азбука-Аттикус  
**Год издания:** 2 017  
**ISBN:** 4563951956382  
**543₽**  
Купить

**Решение задачи по математическому анализ**  
Иванова Е. П.  
**Жанр:** Хоррор  
**Возрастное ограничение:** 18+  
**Издательство:** АСТ  
**Год издания:** 2 000  
**ISBN:** 6573951956356  
**1 500₽**  
Купить

**Человек, который принял жену за шляпу**  
Оливер Сакс  
**Жанр:** Психология  
**Возрастное ограничение:** 16+  
**Издательство:** Альпина Паблишер  
**Год издания:** 2 016  
**ISBN:** 9763170121152  
**240₽**  
Купить

**1984**  
Джордж Оруэлл  
**Жанр:** Художественная литература

**Чувства принца Чарльза**

**Миф о красоте**

Рис. 10: Страница пользователя.

Одно из возможных действий далее это покупка выбранной книги. Нажав на кнопку "Купить" пользователь переходит на страницу оформления заказа. И после ввода

всех данных может оплатить покупку и увидеть сообщение об успешном оформлении.

Город доставки:

Челябинск

улица

something

winter

Способ доставки:

Самовывоз 100

Оплатить

Рис. 11: Оформление заказа.

Был введен промокод, который есть в базе данных, соответственно пользователю будет предоставлена скидка.

**Заказ № 10 на сумму 323 ₽**

Применен промокод: winter

**Спасибо за Ваш заказ**

**Он будет доставлен в ближайшее время!**

**рири, ждем Вас снова!**

Вернуться на [Главную](#)

Рис. 12: Успешная покупка.

Также пользователь может изменить свой email и пароль нажав на кнопку "Профиль".

рири

Email:

Пароль:

Рис. 13: Редактирование данных в личном кабинете.

Теперь рассмотрим страницу администратора. На ней возможно отредактировать количество экземпляров у выбранной книги и добавить новую книгу в продажу.

BOOKS FOR EVERYONE [Книги](#) [Список пользователей](#) [Профиль](#) v

<p><b>краткая история</b> Лив Стрё</p> <p><b>Жанр:</b> Общество</p> <p><b>Возрастное ограничение:</b> 12+</p> <p><b>Издательство:</b> Азбука-Аттикус</p> <p><b>Год издания:</b> 2 017</p> <p><b>ISBN:</b> 4563951956382</p> <p><b>543Р</b></p> <p><b>Количество:</b> 2</p> <p><input type="button" value="Редактировать"/></p>	<p><b>Решение задачи по математическому анализ</b> Иванова Е. П.</p> <p><b>Жанр:</b> Хоррор</p> <p><b>Возрастное ограничение:</b> 18+</p> <p><b>Издательство:</b> АСТ</p> <p><b>Год издания:</b> 2 000</p> <p><b>ISBN:</b> 6573951956356</p> <p><b>1 500Р</b></p> <p><b>Количество:</b> 3</p> <p><input type="button" value="Редактировать"/></p>	<p><b>Человек, который принял жену за шляпу</b> Оливер Сакс</p> <p><b>Жанр:</b> Психология</p> <p><b>Возрастное ограничение:</b> 16+</p> <p><b>Издательство:</b> Альпина Паблишер</p> <p><b>Год издания:</b> 2 016</p> <p><b>ISBN:</b> 9763170121152</p> <p><b>240Р</b></p> <p><b>Количество:</b> 3</p> <p><input type="button" value="Редактировать"/></p>
<p><b>1984</b> Джордж Оруэлл</p> <p><b>Жанр:</b> Художественная литература</p> <p><b>Возрастное ограничение:</b> 16+</p>	<p><b>Чувства принца Чарльза</b> Лив Стрёмквист</p>	<p><b>Миф о красоте</b> Наоми Вульф</p>

Рис. 14: Страница администратора.

Рассмотрим редактирование экземпляров книги.

**Миф о красоте**  
Наоми Вульф  
Жанр: Психология  
Возрастное ограничение: 16+  
Издательство: Альпина Пабlishер  
Год издания: 2 018  
ISBN: 9785001390244  
**300₽**  
Количество: 5  
Редактировать

Рис. 15: До редактирования.

6  
Обновить количество

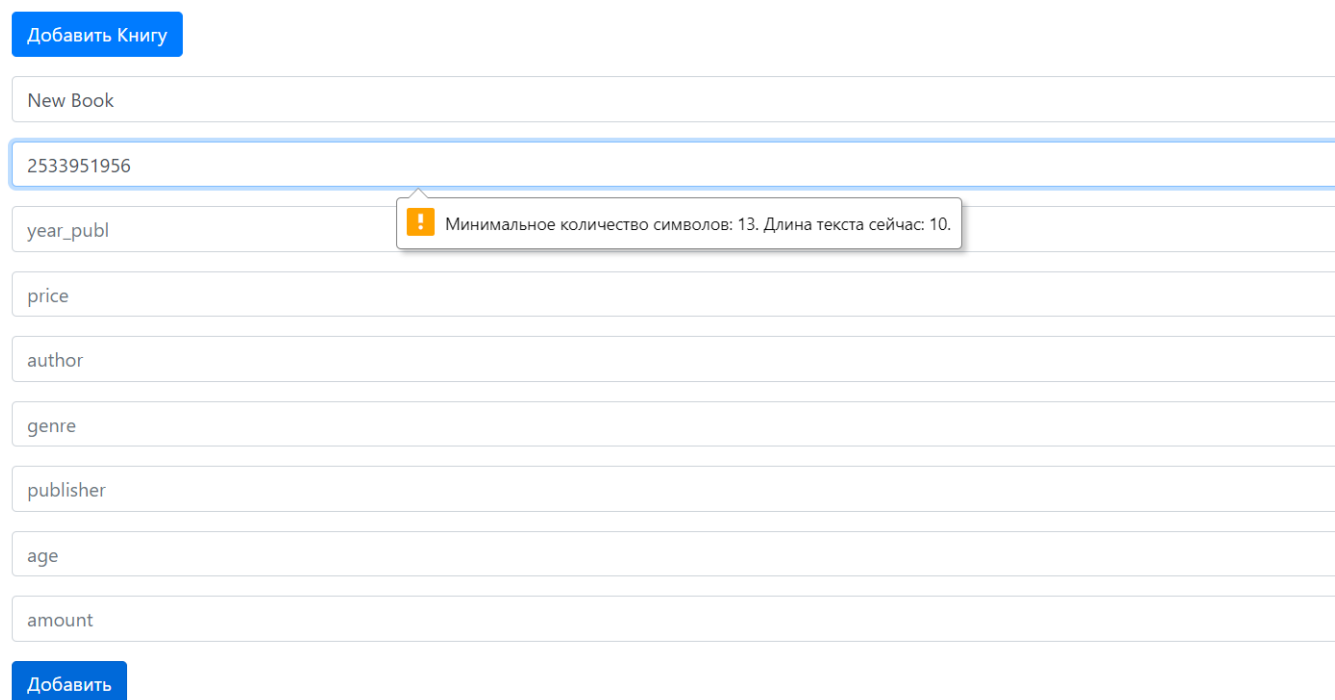
Рис. 16: Редактирование количества.

**Миф о красоте**  
Наоми Вульф  
Жанр: Психология  
Возрастное ограничение: 16+  
Издательство: Альпина Пабlishер  
Год издания: 2 018  
ISBN: 9785001390244  
**300₽**  
Количество: 6  
Редактировать

Рис. 17: После редактирования.

Добавление новой книги производится с валидацией каждого поля: isbn строго в

13 символов, в поля цена и количество экземпляров нельзя записать отрицательные числа, также все поля являются обязательными для заполнения.



Добавить Книгу

New Book

2533951956

year\_publ

price

author

genre

publisher

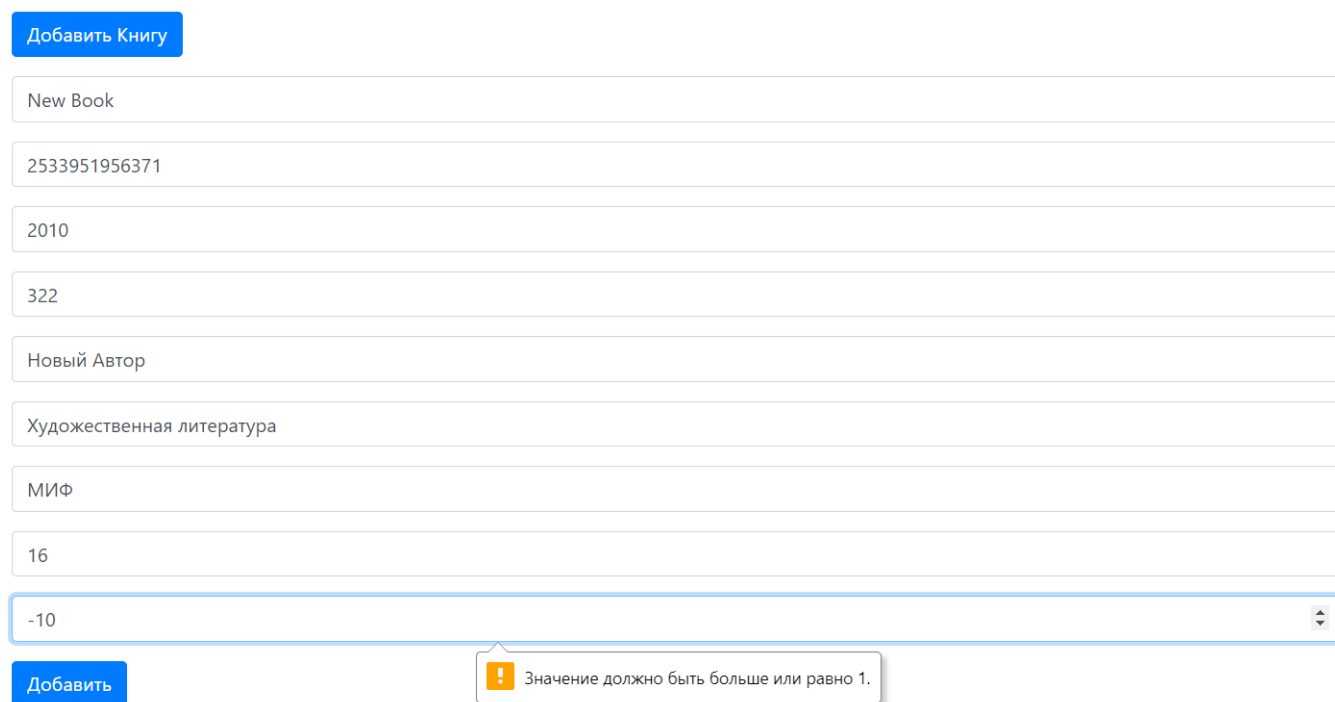
age

amount

Добавить

Минимальное количество символов: 13. Длина текста сейчас: 10.

Рис. 18: Неверное количество цифр в ISBN.



Добавить Книгу

New Book

2533951956371

2010

322

Новый Автор

Художественная литература

МИФ

16

-10

Добавить

Значение должно быть больше или равно 1.

Рис. 19: Отрицательное число в количестве экземпляров.

После того как все поля верно заполнены новая книга сразу добавляется на страницу.

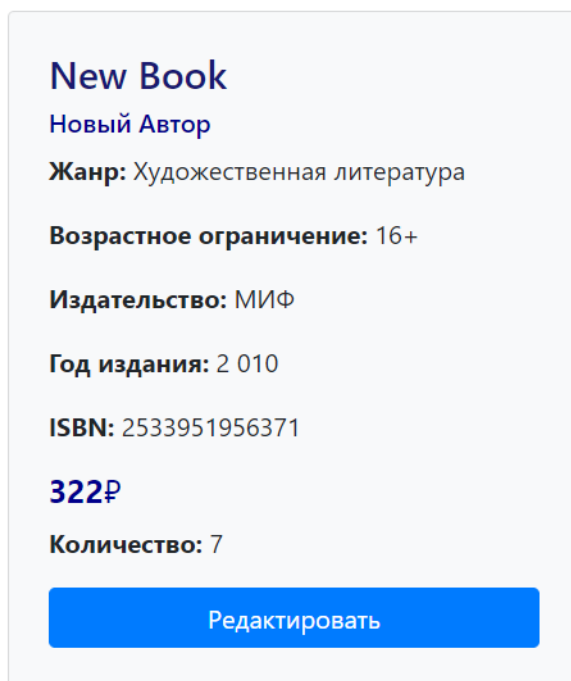


Рис. 20: Успешное добавление новой книги.



## 6 Список использованных источников

1. Документация PostgresPro 14. <http://postgrespro.ru/docs/14/>.
2. П. В. Лузанов, Е. В. Рогов, И. В. Лёвшин. Postgres. Первое знакомство. <http://postgrespro.ru/education/books/introbook>.
3. Дж. Р. Грофф, Пол Н. Вайнберг, Эндрю Оппель. SQL полное руководство. Третье издание. 2015
4. К. Дж. Дейт. Введение в системы баз данных. — 8-е изд. — М.: Вильямс, 2006. — С. 1328.
5. Д. С. Боуман, С. Л. Эмерсон, М. Дарновски. Практическое руководство по SQL. “Вильямс”, 2001, 4 изд.

## 7 Приложение

### 7.1 Скрипты создания таблиц

#### 7.1.1 Создание таблицы ages

```
CREATE TABLE public.ages (  
    age_id serial4 NOT NULL,  
    age_number int2 NOT NULL,  
    CONSTRAINT ages_age_number_key UNIQUE (age_number),  
    CONSTRAINT ages_pkey PRIMARY KEY (age_id)  
);
```

---

#### 7.1.2 Создание таблицы author

```
CREATE TABLE public.author (  
    author_id serial4 NOT NULL,  
    author_name text NOT NULL,  
    CONSTRAINT author_author_name_key UNIQUE (author_name),  
    CONSTRAINT author_pkey PRIMARY KEY (author_id)  
);
```

---

#### 7.1.3 Создание таблицы publisher

```
CREATE TABLE public.publisher (  
    publisher_id serial4 NOT NULL,  
    publisher_name text NOT NULL,  
    CONSTRAINT publisher_pkey PRIMARY KEY (publisher_id),  
    CONSTRAINT publisher_publisher_name_key UNIQUE (publisher_name)  
);
```

---

#### 7.1.4 Создание таблицы genre

```
CREATE TABLE public.genre (  
    genre_id serial4 NOT NULL,  
    genre_name text NOT NULL,  
    CONSTRAINT genre_genre_name_key UNIQUE (genre_name),  
    CONSTRAINT genre_pkey PRIMARY KEY (genre_id)  
);
```

---

## 7.1.5 Создание таблицы book

```
CREATE TABLE public.book (  
    book_id serial4 NOT NULL,  
    title text NOT NULL,  
    author_id int4 NOT NULL,  
    genre_id int4 NOT NULL,  
    isbn varchar(13) NOT NULL,  
    year_publ int4 NOT NULL,  
    publisher_id int4 NOT NULL,  
    age_id int4 NOT NULL,  
    price numeric(6, 2) NOT NULL,  
    amount int4 NOT NULL,  
    CONSTRAINT book_isbn_key UNIQUE (isbn),  
    CONSTRAINT book_pkey PRIMARY KEY (book_id),  
    CONSTRAINT book_price_check CHECK ((price > 0.00)),  
    CONSTRAINT book_age_id_fkey FOREIGN KEY (age_id) REFERENCES public.ages(age_id),  
    CONSTRAINT book_author_id_fkey FOREIGN KEY (author_id) REFERENCES public.author(author_id),  
    CONSTRAINT book_genre_id_fkey FOREIGN KEY (genre_id) REFERENCES public.genre(genre_id),  
    CONSTRAINT book_publisher_id_fkey FOREIGN KEY (publisher_id) REFERENCES public.publisher(publisher_id)  
);  
  
-- Table Triggers  
  
create trigger amount_del after  
update  
    on  
    public.book for each row execute function aft_update();
```

---

## 7.1.6 Создание таблицы roles

```
CREATE TABLE public.roles (  
    cust_id int8 NOT NULL,  
    roles varchar(255) NULL,  
    CONSTRAINT fk57oyc0cua09my3do1mcwpq5b4 FOREIGN KEY (cust_id) REFERENCES public.customer(customer_id)  
);
```

---

## 7.1.7 Создание таблицы customer

```
CREATE TABLE public.customer (  
    customer_id serial4 NOT NULL,  
    username text NULL,  
    email text NULL,  
    customer_password text NULL,  
    CONSTRAINT customer_pkey PRIMARY KEY (customer_id)  
);
```

---

### 7.1.8 Создание таблицы city

```
CREATE TABLE public.city (  
    city_id serial4 NOT NULL,  
    city_name text NOT NULL,  
    CONSTRAINT city_city_name_key UNIQUE (city_name),  
    CONSTRAINT city_pkey PRIMARY KEY (city_id)  
);
```

---

### 7.1.9 Создание таблицы promo\_code

```
CREATE TABLE public.promo_code (  
    promo_code_id serial4 NOT NULL,  
    code text NOT NULL,  
    discount int4 NOT NULL,  
    CONSTRAINT promo_code_code_key UNIQUE (code),  
    CONSTRAINT promo_code_pkey PRIMARY KEY (promo_code_id)  
);
```

---

### 7.1.10 Создание таблицы ways\_of\_delivery

```
CREATE TABLE public.ways_of_delivery (  
    way_id serial4 NOT NULL,  
    ways text NOT NULL,  
    delivery_price numeric(6, 2) NULL,  
    CONSTRAINT ways_of_delivery_delivery_price_check CHECK ((delivery_price > 0.00)),  
    CONSTRAINT ways_of_delivery_pkey PRIMARY KEY (way_id),  
    CONSTRAINT ways_of_delivery_ways_key UNIQUE (ways)  
);
```

---

### 7.1.11 Создание таблицы buy

```
CREATE TABLE public.buy (  
    buy_id serial4 NOT NULL,  
    paid_or_not bool NOT NULL,  
    customer_id int4 NOT NULL,  
    way_id int4 NOT NULL,  
    promo_code_id int4 NULL,  
    order_comments text NULL,  
    date_of_buying date NOT NULL,  
    city_id int4 NOT NULL,  
    CONSTRAINT buy_pk PRIMARY KEY (buy_id, date_of_buying),  
    CONSTRAINT buy_city_id_fkey FOREIGN KEY (city_id) REFERENCES public.city(city_id),  
    CONSTRAINT buy_customer_id_fkey FOREIGN KEY (customer_id) REFERENCES public.customer(customer_id),  
    CONSTRAINT buy_promo_code_id_fkey FOREIGN KEY (promo_code_id) REFERENCES public.promo_code(promo_code_id),  
    CONSTRAINT buy_way_id_fkey FOREIGN KEY (way_id) REFERENCES public.ways_of_delivery(way_id)  
)  
PARTITION BY RANGE (date_of_buying);
```

---

### 7.1.12 Создание таблицы buy\_book

```
CREATE TABLE public.buy_book (  
    buy_book_id serial4 NOT NULL,  
    book_id int4 NULL,  
    buy_id int4 NULL,  
    order_date date NULL,  
    amount int4 NOT NULL,  
    CONSTRAINT buy_book_pkey PRIMARY KEY (buy_book_id),  
    CONSTRAINT buy_book_book_id_fkey FOREIGN KEY (book_id) REFERENCES public.book(book_id),  
    CONSTRAINT buy_book_buy_id_order_date_fkey FOREIGN KEY (buy_id,order_date) REFERENCES public.buy(buy_id,date_of_buying)  
);
```

---

## 7.2 Скрипты создания представлений

### 7.2.1 Создание представления book\_info

```
CREATE OR REPLACE VIEW public.book_info  
AS SELECT book.title,  
    author.author_name,  
    genre.genre_name,  
    book.isbn,  
    book.year_publ,  
    publisher.publisher_name,  
    ages.age_number,  
    book.price,  
    book.amount  
FROM book  
    JOIN author USING (author_id)  
    JOIN genre USING (genre_id)  
    JOIN publisher USING (publisher_id)  
    JOIN ages USING (age_id);
```

---

## 7.3 Скрипты создания хранимых процедур и функций

### 7.3.1 aft\_update

```
CREATE OR REPLACE FUNCTION public.aft_update()  
    RETURNS trigger  
    LANGUAGE plpgsql  
AS $function$  
begin  
    if new.amount = 0 then  
        delete from book where amount = 0;  
    end if;
```

```
return null;
end;
$function$
;
```

---

## 7.4 ПО «Книжный магазин»

### 7.4.1 Код сервера

#### Файл MvcConfig.java

```
@Configuration
public class MvcConfig implements WebMvcConfigurer {

    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/login").setViewName("login");
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/static/**")
            .addResourceLocations("classpath:/static/");
    }
}
```

---

#### Файл WebSecurityConfig.java

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    private final CustomerService customerService;
    public WebSecurityConfig(CustomerService customerService, PasswordEncoder passwordEncoder) {
        this.customerService = customerService;
        this.passwordEncoder = passwordEncoder;
    }

    private final PasswordEncoder passwordEncoder;

    @Bean
    public static PasswordEncoder getPasswordEncoder() {
        return new BCryptPasswordEncoder(8);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/", "/registration", "/static/**", "/userPage").permitAll()
            .anyRequest().authenticated()
    }
}
```

```

        .and()
        .formLogin()
        .loginPage("/login").passwordParameter("customer_password").defaultSuccessUrl("/userPage").permitAll()
        .and()
        .logout()
        .permitAll()
    };
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(customerService)
        .passwordEncoder(passwordEncoder);
}
}

```

## Файл BuyBookController.java

```

@Controller
public class BuyBookController {
    private final WaysRepo waysRepo;
    private final BookRepo bookRepo;
    private final CityRepo cityRepo;
    private final CustomerRepo customerRepo;
    private final BuyRepo buyRepo;
    private final BuyBookRepo buyBookRepo;
    private final PromoRepo promoRepo;

    public BuyBookController(WaysRepo waysRepo, BookRepo bookRepo,
                             CityRepo cityRepo, CustomerRepo customerRepo,
                             BuyRepo buyRepo, BuyBookRepo buyBookRepo, PromoRepo promoRepo) {
        this.waysRepo = waysRepo;
        this.bookRepo = bookRepo;
        this.cityRepo = cityRepo;
        this.customerRepo = customerRepo;
        this.buyRepo = buyRepo;
        this.buyBookRepo = buyBookRepo;
        this.promoRepo = promoRepo;
    }

    @GetMapping("/buybook/{name}/{book_id}")
    public String orderInfo(
        Model model,
        @RequestParam(required = false) Book book
    ) {
        Iterable<City> city = cityRepo.findAll();
        model.addAttribute("city", city);
        Iterable<WaysOfDelivery> waysOfDeliveries = waysRepo.findAll();
        model.addAttribute("waysOfDeliveries", waysOfDeliveries);
        return "buyBook";
    }

    @PostMapping("/buybook/{name}/{book_id}")
    public String buy(@RequestParam("citySelect") String citySelect,
                     @RequestParam(required=false, name="order_comments") String order_comments,
                     @RequestParam(required=false, name="promo_code") String promo_code,
                     @RequestParam("waysSelect") String waysSelect,
                     @ModelAttribute("book_id") long id,
                     @ModelAttribute("name") String name,

```

```

        Model model
    ) throws IOException {
        Book book = bookRepo.findBookByBook_id(id);
        City city = cityRepo.findCityByCity_name(citySelect);
        WaysOfDelivery way = waysRepo.findWaysOfDeliveriesByWays(waysSelect);
        Customer customer = customerRepo.findByUsername(name);
        PromoCode promoCode = promoRepo.findPromoCodeByCode(promo_code);
        Buy buy = new Buy(new Date(),true, promoCode, customer, way, order_comments, city);
        buyRepo.save(buy);

        int sum = book.getPrice() + way.getDelivery_price();
        if(promoCode != null)
        {
            double discount = (100 - promoCode.getDiscount()) / 100.0;
            sum *= discount;
            model.addAttribute("ifDiscount", promo_code);
        }
        BuyBook buyBook = new BuyBook(new Date(), sum, book, buy);
        buyBookRepo.save(buyBook);
        model.addAttribute("orderNumber", buyBook.getBuy_book_id());
        model.addAttribute("sum", sum);
        Integer amount = book.getAmount();
        book.setAmount(amount - 1);
        bookRepo.save(book);
        return "afterBuy";
    }
}

```

---

## Файл ControllerUtils.java

```

public class ControllerUtils {
    static Map<String, String> getErrors(BindingResult bindingResult) {
        Collector<FieldError, ?, Map<String, String>> collector = Collectors.toMap(
            fieldError -> fieldError.getField() + "Error",
            FieldError::getDefaultMessage
        );
        return bindingResult.getFieldErrors().stream().collect(collector);
    }
}

```

## Файл MainController.java

```

@Controller
@PreAuthorize("hasAuthority('ADMIN')")
public class MainController {

    private final BookRepo bookRepo;

    private final AuthorRepo authorRepo;

    private final GenreRepo genreRepo;

    private final PublisherRepo publisherRepo;

    private final AgeRepo ageRepo;
}

```



```

public MainController(BookRepo bookRepo, AuthorRepo authorRepo, GenreRepo genreRepo,
PublisherRepo publisherRepo, AgeRepo ageRepo) {
    this.bookRepo = bookRepo;
    this.authorRepo = authorRepo;
    this.genreRepo = genreRepo;
    this.publisherRepo = publisherRepo;
    this.ageRepo = ageRepo;
}

@GetMapping("/main")
public String main(@RequestParam(required = false, defaultValue = "") String filter,
@RequestParam(required = false, defaultValue = "") String filterAuthor, Model model) {
    Iterable<Book> books = bookRepo.findAll();
    books = bookRepo.findAll();
    model.addAttribute("books", books);
    model.addAttribute("filter", filter);
    model.addAttribute("filterAuthor", filterAuthor);
    return "main";
}
/**/
@PostMapping("/main")
public String add(
    @RequestParam String title,
    @RequestParam String isbn,
    @RequestParam Integer year_publ,
    @RequestParam Integer price,
    @RequestParam String author,
    @RequestParam String genre,
    @RequestParam String publisher,
    @RequestParam Integer age,
    @RequestParam Integer amount,
    Model model
) throws IOException {

    Publisher publisherFromDB = publisherRepo.findPublisherByPublisher_name(publisher);
    Genre genreFromDB = genreRepo.findGenreByGenre_name(genre);
    Age ageFromDB = ageRepo.findAgeByAge_number(age);
    Author authorFromDB = authorRepo.findAuthorByAuthor_name(author);

    if (publisherFromDB == null)
    {
        publisherFromDB = new Publisher(publisher);
        publisherRepo.save(publisherFromDB);
    }
    if (authorFromDB == null)
    {
        authorFromDB = new Author(author);
        authorRepo.save(authorFromDB);
    }
    if (ageFromDB == null)
    {
        ageFromDB = new Age(age);
        ageRepo.save(ageFromDB);
    }
    if (genreFromDB == null)
    {
        genreFromDB = new Genre(genre);
        genreRepo.save(genreFromDB);
    }
}

```

```

    }

    Book book = new Book(title, isbn, year_publ,
        price, amount, authorFromDB,
        genreFromDB, publisherFromDB, ageFromDB);
    bookRepo.save(book);

    Iterable<Book> books = bookRepo.findAll();
    model.addAttribute("books", books);
    return "main";
}

@GetMapping("/edit/{book_id}")
public String bookEdit(
    Model model,
    @RequestParam(required = false) Book book
) {
    /*Iterable<Book> books = bookRepo.findBookByBook_id(curr.getBook_id());
    model.addAttribute("books", books);*/

    return "bookEdit";
}

@PostMapping("/edit/{book_id}")
public String updateAmount(
    @ModelAttribute("book_id") long id,
    @RequestParam("amount") Integer amount
) throws IOException {
    Book book = bookRepo.findBookByBook_id(id);
    if (!StringUtils.isEmpty(amount)) {
        book.setAmount(amount);
    }

    bookRepo.save(book);

    return "redirect:/main";
}
}

```

## Файл RegistrationController.java

```

@Controller
public class RegistrationController {
    private final CustomerService customerService;

    public RegistrationController(CustomerService customerService) {
        this.customerService = customerService;
    }

    @GetMapping("/registration")
    public String registration(){
        return "registration";
    }

    @PostMapping("/registration")
    public String addCustomer(@Valid Customer customer, BindingResult bindingResult, Model model){

```

```

        if (bindingResult.hasErrors()) {
            Map<String, String> errors = ControllerUtils.getErrors(bindingResult);

            model.mergeAttributes(errors);

            return "registration";
        }
        if(!customerService.addCustomer(customer))
        {
            model.addAttribute("message", "Customer with this email already exists!");
            return "registration";
        }
        return "redirect:/login";
    }
}

```

## Файл UserController.java

```

@Controller
@RequestMapping("/user")
public class UserController {
    private final CustomerService customerService;

    public UserController(CustomerService customerService) {
        this.customerService = customerService;
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @GetMapping
    public String userList(Model model) {
        model.addAttribute("users", customerService.findAll());

        return "userList";
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @GetMapping("/{customer}")
    public String userEditForm(@PathVariable Customer customer, Model model){
        model.addAttribute("user", customer);
        model.addAttribute("roles", Role.values());
        return "userEdit";
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @PostMapping
    public String userSave(@RequestParam Map<String, String> form,
        @RequestParam("userId") Customer customer){
        customerService.saveCustomer(customer, form);
        return "redirect:/user";
    }

    @GetMapping("profile")
    public String getProfile(Model model, @AuthenticationPrincipal Customer customer) {
        model.addAttribute("username", customer.getUsername());
        model.addAttribute("email", customer.getEmail());

        return "profile";
    }
}

```

```

@PostMapping("profile")
public String updateProfile(
    @AuthenticationPrincipal Customer customer,
    @RequestParam String email,
    @RequestParam String customer_password
) {
    customerService.updateProfile(customer, customer_password, email);

    return "redirect:/user/profile";
}
}

```

## Файл UserPageController.java

```

@Controller
public class UserPageController {
    private final BookRepo bookRepo;

    public UserPageController(BookRepo bookRepo) {
        this.bookRepo = bookRepo;
    }

    @GetMapping("/")
    public String greeting(@RequestParam(required = false, defaultValue = "") String filter,
        @RequestParam(required = false, defaultValue = "") String filterAuthor,
        @RequestParam(required = false, defaultValue = "") String filterGenre,
        Model model) {
        Iterable<Book> books = bookRepo.findAll();
        if(filter != null && !filter.isEmpty()) {
            books = bookRepo.findByTitle(filter);
        } else if(filterAuthor != null && !filterAuthor.isEmpty()) {
            books = bookRepo.findByAuthor(filterAuthor);
        } else if(filterGenre != null && !filterGenre.isEmpty()) {
            books = bookRepo.findByGenre(filterGenre);
        } else
            books = bookRepo.findAll();
        model.addAttribute("books", books);
        model.addAttribute("filter", filter);
        model.addAttribute("filterAuthor", filterAuthor);
        return "greeting";
    }

    @GetMapping("/userPage")
    public String main(@RequestParam(required = false, defaultValue = "") String filter,
        @RequestParam(required = false, defaultValue = "") String filterAuthor,
        @RequestParam(required = false, defaultValue = "") String filterGenre,
        Model model) {
        Iterable<Book> books = bookRepo.findAll();
        if(filter != null && !filter.isEmpty()) {
            books = bookRepo.findByTitle(filter);
        } else if(filterAuthor != null && !filterAuthor.isEmpty()) {
            books = bookRepo.findByAuthor(filterAuthor);
        } else if(filterGenre != null && !filterGenre.isEmpty()) {
            books = bookRepo.findByGenre(filterGenre);
        } else
            books = bookRepo.findAll();
        model.addAttribute("books", books);
        model.addAttribute("filter", filter);
        model.addAttribute("filterAuthor", filterAuthor);
    }
}

```

```

        return "userPage";
    }
}

```

## Файл Author.java

```

@Entity
@Table(name = "author")
@Data
public class Author{

    @Id
    @Column(name = "author_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long author_id;

    @Column(name = "author_name")
    private String author_name;

    public Author() {
    }

    public Author(String author_name) {
        this.author_name = author_name;
    }
}

```

## Файл Book.java

```

@Entity
@Table(name = "book", schema = "public")
@Data
public class Book {
    public Book(){}

    @Id
    @Column(name = "book_id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long book_id;

    @Column(name = "title")
    private String title;

    @Column(name = "isbn")
    private String isbn;

    @Column(name = "year_publ")
    private Integer year_publ;

    @Column(name = "price")
    private Integer price;

    @Column(name = "amount")
    private Integer amount;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private Author author;
}

```

```

@ManyToOne
@JoinColumn(name = "genre_id")
private Genre genre;

@ManyToOne
@JoinColumn(name = "publisher_id")
private Publisher publisher;

@ManyToOne
@JoinColumn(name = "age_id")
private Age age;

public Book(String title, String isbn, Integer year_publ,
            Integer price, Integer amount, Author author,
            Genre genre, Publisher publisher, Age age) {
    this.title = title;
    this.isbn = isbn;
    this.year_publ = year_publ;
    this.price = price;
    this.amount = amount;
    this.author = author;
    this.genre = genre;
    this.publisher = publisher;
    this.age = age;
}
}

```

## Файл Buy.java

```

@Entity
@Table(name = "buy")
@Data
public class Buy {

    @Id
    @Column(name = "buy_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long buy_book_id;

    @Column(name = "date_of_buying")
    private Date date_of_buying;

    @Column(name = "paid_or_not")
    private boolean paid_or_not;

    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;

    @ManyToOne
    @JoinColumn(name = "way_id")
    private WaysOfDelivery waysOfDelivery;

    @Column(name = "order_comments")
    private String order_comments;

    @ManyToOne
    @JoinColumn(name = "city_id")
    private City city;
}

```

```

@ManyToOne
@JoinColumn(name = "promo_code_id")
private PromoCode promoCode;

public Buy(Date date_of_buying, boolean paid_or_not, PromoCode promoCode, Customer customer,
WaysOfDelivery waysOfDelivery, String order_comments, City city) {
    this.date_of_buying = date_of_buying;
    this.paid_or_not = paid_or_not;
    this.promoCode = promoCode;
    this.customer = customer;
    this.waysOfDelivery = waysOfDelivery;
    this.order_comments = order_comments;
    this.city = city;
}

public Buy() { }
}

```

## Файл BuyBook.java

```

@Entity
@Table(name = "buy_book")
@Data
public class BuyBook {
    @Id
    @Column(name = "buy_book_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long buy_book_id;

    @Column(name = "order_date")
    private Date order_date;

    @Column(name = "amount")
    private Integer orderPrice;

    @OneToOne
    @JoinColumn(name = "book_id")
    private Book orderBook;

    @OneToOne
    @JoinColumn(name = "buy_id")
    private Buy buy;

    public BuyBook(Date order_date, Integer orderPrice, Book orderBook, Buy buy) {
        this.order_date = order_date;
        this.orderPrice = orderPrice;
        this.orderBook = orderBook;
        this.buy = buy;
    }

    public BuyBook() {}
}

```

## Файл Customer.java

```

@Entity
@Table(name = "customer")
@Data

```

```

public class Customer implements UserDetails {
    @Id
    @Column(name = "customer_id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long customer_id;

    @NotBlank(message = "Username cannot be empty")
    @Column(name = "username")
    private String username;

    @NotBlank(message = "Password cannot be empty")
    @Column(name = "customer_password")
    private String customer_password;

    @Email(message = "Email is not correct")
    @NotBlank(message = "Email cannot be empty")
    @Column(name = "email")
    private String email;

    @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
    @CollectionTable(name = "roles", joinColumns = @JoinColumn(name = "cust_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> roles;

    public boolean isAdmin(){
        return roles.contains(Role.ADMIN);
    }

    public boolean isUser(){
        return roles.contains(Role.USER);
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return getRoles();
    }

    @Override
    public String getPassword() {
        return customer_password;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override

```



```

    public boolean isEnabled() {
        return true;
    }
}

```

## Файл WaysOfDelivery.java

```

@Entity
@Table(name = "ways_of_delivery")
@Data
public class WaysOfDelivery {
    @Id
    @Column(name = "way_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long way_id;

    @Column(name = "ways")
    private String ways;

    @Column(name = "delivery_price")
    private Integer delivery_price;
}

```

## Файл AuthorRepo.java

```

@Repository
public interface AuthorRepo extends JpaRepository<Author, Long> {
    @Query("select a from Author a where a.author_name=?1")
    Author findAuthorByAuthor_name(String name);
}

```

## Файл BookRepo.java

```

@Repository
public interface BookRepo extends CrudRepository<Book, Long> {
    List<Book> findByTitle(String title);

    @Query("select b from Book b where b.book_id=?1")
    Book findBookByBook_id(Long id);

    @Query("select b from Book b join Author a on b.author.author_id = a.author_id where a.author_name=?1")
    List<Book> findByAuthor(String author);

    @Query("select b from Book b join Genre a on b.genre.genre_id = a.genre_id where a.genre_name=?1")
    List<Book> findByGenre(String genre);
}

```

## Файл CustomerRepo.java

```

@Repository
public interface CustomerRepo extends JpaRepository<Customer, Long> {
    @Query("select c from Customer c where c.username=?1")
    Customer findByUsername(String username);

    Customer findCustomerByEmail(String email);
}

```

## Файл WaysRepo.java

```

@Repository
public interface WaysRepo extends JpaRepository<WaysOfDelivery, Long> {
    WaysOfDelivery findWaysOfDeliveriesByWays(String ways);
}

```

## Файл PublisherRepo.java

```

@Repository
public interface PublisherRepo extends JpaRepository<Publisher, Long> {
    @Query("select p from Publisher p where p.publisher_name=?1")
    Publisher findPublisherByPublisher_name(String name);
}

```

## Файл CustomerService.java

```

@Service
public class CustomerService implements UserDetailsService {
    private final CustomerRepo customerRepo;
    private final PasswordEncoder passwordEncoder;

    public CustomerService(CustomerRepo customerRepo, PasswordEncoder passwordEncoder) {
        this.customerRepo = customerRepo;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Customer customer = customerRepo.findByUsername(username);
        if(customer == null)
        {
            throw new UsernameNotFoundException("Customer not found!");
        }
        return customer;
    }

    public List<Customer> findAll() {
        return customerRepo.findAll();
    }

    public boolean addCustomer(Customer customer){
        //Customer custFromDB = customerRepo.findByUsername(customer.getUsername());
        Customer custFromDB = customerRepo.findCustomerByEmail(customer.getEmail());
        if(custFromDB != null)
        {
            return false;
        }
        customer.setRoles(Collections.singleton(Role.USER));
        customer.setCustomer_password(passwordEncoder.encode(customer.getCustomer_password()));
        customerRepo.save(customer);
        return true;
    }

    public void saveCustomer(Customer customer, Map<String, String> form) {
        Set<String> roles = Arrays.stream(Role.values())
            .map(Role::name)
            .collect(Collectors.toSet());
        customer.getRoles().clear();
        for(String key : form.keySet())

```

```

    {
        if(roles.contains(key))
        {
            customer.getRoles().add(Role.valueOf(key));
        }
    }
    customerRepo.save(customer);
}

public void updateProfile(Customer customer, String password, String email) {
    String userEmail = customer.getEmail();
    boolean isEmailChanged = (email != null && !email.equals(userEmail)) ||
        (userEmail != null && !userEmail.equals(email));

    if (isEmailChanged) {
        customer.setEmail(email);
    }

    if (!StringUtils.isEmpty(password)) {
        customer.setCustomer_password(passwordEncoder.encode(password));
    }

    customerRepo.save(customer);
}
}

```

## 7.4.2 Код пользовательского клиента

### Файл common.ftlh

```

<#macro page>
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>BookShop</title>
        <link rel="stylesheet" href="/static/style.css">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
            integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
            crossorigin="anonymous">

    </head>
    <body>
        <#include "navbar.ftlh">
        <div class="container mt-4">
            <#nested>
        </div>
        <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
            integrity="sha384-q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
            crossorigin="anonymous"></script>
        <script src="https://cdn.jsdelivr.net/npm/popper.js/1.14.3/umd/popper.min.js"

```

```

    integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAAdn689aCwoqBJiSnjAK/18WvCWPIpM49"
    crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
integrity="sha384-ChfqquxZUCnJSK3+MXmPNiYE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
crossorigin="anonymous"></script>

</body>
</html>
</#macro>

```

## Файл navbar.ftlh

```

<#include "security.ftlh">

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <#if isAdmin>
        <a class="navbar-brand" href="/main">BOOKS FOR EVERYONE</a>
    <#elseif isUser>
        <a class="navbar-brand" href="/userPage">BOOKS FOR EVERYONE</a>
    <#else>
        <a class="navbar-brand" href="/">BOOKS FOR EVERYONE</a>
    </#if>

    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"
    aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <!--<li class="nav-item">
                <a class="nav-link" href="/">greeting(убрать)</a>
            </li>-->
            <#if isUser>
                <li class="nav-item">
                    <a class="nav-link" href="/userPage">Книги</a>
                </li>
            <#elseif isAdmin>
                <li class="nav-item">
                    <a class="nav-link" href="/userPage">Книги</a>
                </li>
            <#else>
                <li class="nav-item">
                    <a class="nav-link" href="/">Книги</a>
                </li>
            </#if>

            <#if isAdmin>
                <li class="nav-item">
                    <a class="nav-link" href="/user">Список пользователей</a>
                </li>
            </#if>
            <#if user??>
                <li class="nav-item">
                    <a class="nav-link" href="/user/profile">Профиль</a>
                </li>
            </#if>
        </ul>
    </div>

```

```

<div class="navbar-text mr-3">${name}</div>
<form action="/logout" method="post">
    <input type="hidden" name="_csrf" value="${_csrf.token}">
    <button class="btn btn-primary" type="submit">Выйти</button>
</form>
</div>
</nav>

```

## Файл security.ftlh

```

<#assign known = Session.SPRING_SECURITY_CONTEXT??>
<#if known>
    <#assign
        user = Session.SPRING_SECURITY_CONTEXT.authentication.principal
        name = user.getUsername()
        email = user.getEmail()
        isAdmin = user.isAdmin()
        isUser = user.isUser()
    >
    <#else>
        <#assign
            name = ""
            isAdmin = false
            isUser = false
        >
    </#if>

```

## Файл afterBuy.ftlh

```

<#import "parts/common.ftlh" as c>
<#include "parts/security.ftlh">
<link href="http://fonts.googleapis.com/css?family=PT+Sans:regular,italic,bold,bolditalic" rel="stylesheet" type="text/css" />
<@c.page>

    <h2>Заказ № ${orderNumber} на сумму ${sum} <span class="rubl">₽8399;</span></h2>
    <h5 style="color:#00008B"><#if ifDiscount??>Применен промокод: ${ifDiscount}</#if></h5>
</div>
<h2>Спасибо за Ваш заказ</h2>
<h3>Он будет доставлен в ближайшее время!</h3>
<h3>${name}, ждем Вас снова!</h3>
<h5>Вернуться на <a href="/userPage">Главную</a></h5>
</@c.page>

```

## Файл buyBook.ftlh

```

<#include "parts/security.ftlh">
<#import "parts/common.ftlh" as c>
<@c.page>
<div class="form-group mt-10">
    <form method="post">
        <div class="form-group mt-10">
            <label>Город доставки: </label>
            <select name="citySelect" class="form-control selectpicker mb-3">
                <#if city?has_content>
                    <#list city as way>
                        <option value="${way.city_name}">${way.city_name}</option>
                    </#list>
                </#if>
            </select>

```

```

<input type="text" class="form-control mb-3" name="address" placeholder="Адрес">
<input type="text" class="form-control mb-3" name="order_comments" placeholder="Комментарии к заказу">
<input type="text" class="form-control mb-3" name="promo_code" placeholder="Промокод">
<label>Способ доставки: </label>
<select name="waysSelect" class="form-control selectpicker mb-3">
  <#if waysOfDeliveries?has_content>
    <#list waysOfDeliveries as way>
      <option value="{way.ways}">{way.ways} {way.delivery_price}</option>
    </#list>
  </#if>
</select>
</div>
<input type="hidden" name="_csrf" value="{_csrf.token}" />
<div class="form-group">
  <button type="submit" class="btn btn-primary mt-1 ">Оплатить</button>
</div>
</form>
</div>
</@c.page>

```

## Файл greeting.ftlh

```

<#import "parts/common.ftlh" as c>
<link href="http://fonts.googleapis.com/css?family=PT+Sans:regular,italic,bold,bolditalic" rel="stylesheet" type="text/css" />

<@c.page>
  <h3>BOOKS FOR EVERYONE</h3>

  <h5 class="mb-3">Для покупки книг - авторизируйтесь <a href="/login">Войти</a></h5>

  <div class="form-row">
    <div class="form-group col-md-10">
      <form method="get" action="/main" style="display:inline">
        <input type="text" name="filter" placeholder="Введите название книги" value="{filter?ifExists}">
        <button class="btn btn-primary ml-1" type="submit">Найти</button>
      </form>
      <form method="get" action="/main" style="display:inline">
        <input class="ml-3" type="text" name="filterAuthor" placeholder="Введите автора"
          value="{filterAuthor?ifExists}">
        <button class="btn btn-primary ml-1" type="submit">Найти</button>
      </form>
      <form method="get" action="/userPage" style="display:inline">
        <input class="ml-3" type="text" name="filterGenre" placeholder="Введите жанр" value="{filterGenre?ifExists}">
        <button class="btn btn-primary ml-1" type="submit">Найти</button>
      </form>
    </div>
  </div>
  <div class="card-columns">
    <#list books as book>
      <div class="card bg-light my-3">
        <div class="card-body m-1">
          <h4 class="card-title" style="color:#191970">{book.title}</h4>
          <h6 class="card-subtitle mb-2" style="color:#00008B"> {book.author.author_name}</h6>
          <p class="card-text"><b>Жанр:</b> {book.genre.genre_name}</p>
          <p class="card-text"><b>Возрастное ограничение:</b> {book.age.age_number}</p>
          <p class="card-text"><b>Издательство:</b> {book.publisher.publisher_name}</p>
          <p class="card-text"><b>Год издания:</b> {book.year_public}</p>
          <p class="card-text"><b>ISBN:</b> {book.isbn}</p>
          <h5 class="card-text" style="color:#00008B"><b>{book.price}</b><span class="rubl">&#8399;</span></h5>
        </div>
      </div>
    </#list>
  </div>
</@c.page>

```

```

        <!--p class="card-text"><b>Количество:</b> ${book.amount}</p-->
    </div>
</div>
<#else>
    Нет книг в наличии
</#list>
</div>
</@c.page>

```

## Файл login.ftlh

```

<#import "parts/common.ftlh" as c>
<#include "parts/security.ftlh">
<@c.page>
    <#if Session?? && Session.SPRING_SECURITY_LAST_EXCEPTION??>
        <div class="alert alert-danger" role="alert">
            ${Session.SPRING_SECURITY_LAST_EXCEPTION.message}
        </div>
    </#if>
    <h2 class="mb-3">Вход в личный кабинет</h2>
    <form action="/login" method="post">
        <div class="form-group row">
            <label class="col-sm-2 col-form-label"> Имя: </label>
            <div class="col-sm-5">
                <input type="text" name="username"
                    class="form-control ${usernameError??}?string('is-invalid', '')}"
                    placeholder="Имя пользователя"/>
                <#if usernameError??>
                    <div class="invalid-feedback">
                        ${usernameError}
                    </div>
                </#if>
            </div>
        </div>
        <div class="form-group row">
            <label class="col-sm-2 col-form-label"> Пароль: </label>
            <div class="col-sm-5">
                <input type="password" name="customer_password"
                    class="form-control ${customer_passwordError??}?string('is-invalid', '')}"
                    placeholder="Пароль"/>
                <#if customer_passwordError??>
                    <div class="invalid-feedback">
                        ${customer_passwordError}
                    </div>
                </#if>
            </div>
        </div>
        <input type="hidden" name="_csrf" value="${_csrf.token}">
        <button class="btn btn-primary mb-3" type="submit">Войти</button>
    </form>
    <div>Еще нет аккаунта? <a href="/registration"> Зарегистрироваться</a></div>
</@c.page>

```

## Файл profile.ftlh

```

<#import "parts/common.ftlh" as c>
<#include "parts/security.ftlh">
<@c.page>
    <h5>${name}</h5>

```

```

<form method="post">
  <div class="form-group row">
    <label class="col-sm-2 col-form-label"> Email: </label>
    <div class="col-sm-5">
      <input type="text" name="email" class="form-control" placeholder="Email" value="{email!''}"/>
    </div>
  </div>
  <div class="form-group row">
    <label class="col-sm-2 col-form-label"> Пароль: </label>
    <div class="col-sm-5">
      <input type="text" name="customer_password" class="form-control" placeholder="Пароль"/>
    </div>
  </div>
  <input type="hidden" name="_csrf" value="{_csrf.token}">
  <button class="btn btn-primary mb-3" type="submit">Сохранить</button>
</form>
</@c.page>

```

## Файл registration.ftlh

```

<#import "parts/common.ftlh" as c>
<@c.page>
  <#if message??>
    <div class="alert alert-warning" role="alert">
      {message}
    </div>
  </#if>
<h2 class="mb-3">Регистрация</h2>
<form action="/registration" method="post">
  <div class="form-group row">
    <label class="col-sm-2 col-form-label"> Имя: </label>
    <div class="col-sm-5">
      <input type="text" name="username"
        class="form-control {(usernameError??)?string('is-invalid', '')}"
        placeholder="Имя пользователя"/>
      <#if usernameError??>
        <div class="invalid-feedback">
          {usernameError}
        </div>
      </#if>
    </div>
  </div>
  <div class="form-group row">
    <label class="col-sm-2 col-form-label"> Email: </label>
    <div class="col-sm-5">
      <input type="email" name="email"
        class="form-control {(emailError??)?string('is-invalid', '')}"
        placeholder="Email"/>
      <#if emailError??>
        <div class="invalid-feedback">
          {emailError}
        </div>
      </#if>
    </div>
  </div>
  <div class="form-group row">
    <label class="col-sm-2 col-form-label"> Пароль: </label>
    <div class="col-sm-5">
      <input type="password" name="customer_password"

```



```

        class="form-control ${customer_passwordError??}?string('is-invalid', '')}"
        placeholder="Пароль"/>
    <#if customer_passwordError??>
        <div class="invalid-feedback">
            ${customer_passwordError}
        </div>
    </#if>
</div>
</div>
<input type="hidden" name="_csrf" value="${_csrf.token}">
<button class="btn btn-primary mb-3" type="submit">Зарегистрироваться</button>
</form>
<div><a href="/login">Уже зарегистрирован(a)</a></div>
</@c.page>

```

## Файл userPage.ftlh

```

<#import "parts/common.ftlh" as c>
<#include "parts/security.ftlh">
<link href="http://fonts.googleapis.com/css?family=PT+Sans:regular,italic,bold,bolditalic" rel="stylesheet" type="text/css" />

<@c.page>

    <div class="form-row">
        <div class="form-group col-md-10">
            <form method="get" action="/userPage" style="display:inline">
                <input type="text" name="filter" placeholder="Введите название книги" value="${filter?ifExists}">
                <button class="btn btn-primary ml-1" type="submit">Найти</button>
            </form>
            <form method="get" action="/userPage" style="display:inline">
                <input class="ml-3" type="text" name="filterAuthor" placeholder="Введите автора" value="${filterAuthor?ifExists}">
                <button class="btn btn-primary ml-1" type="submit">Найти</button>
            </form>
            <form method="get" action="/userPage" style="display:inline">
                <input class="ml-3" type="text" name="filterGenre" placeholder="Введите жанр" value="${filterGenre?ifExists}">
                <button class="btn btn-primary ml-1" type="submit">Найти</button>
            </form>
        </div>
    </div>
    <div class="card-columns">
        <#list books as book>
            <div class="card bg-light my-3">
                <div class="card-body m-1">
                    <h4 class="card-title" style="color:#191970">${book.title}</h4>
                    <h6 class="card-subtitle mb-2" style="color:#00008B"> ${book.author.author_name}</h6>
                    <p class="card-text"><b>Жанр:</b> ${book.genre.genre_name}</p>
                    <p class="card-text"><b>Возрастное ограничение:</b> ${book.age.age_number}</p>
                    <p class="card-text"><b>Издательство:</b> ${book.publisher.publisher_name}</p>
                    <p class="card-text"><b>Год издания:</b> ${book.year_publ}</p>
                    <p class="card-text"><b>ISBN:</b> ${book.isbn}</p>
                    <h4 class="card-text" style="color:#00008B"><b>${book.price}</b><span class="rubl">&#8399;</span></h4>
                    <a class="col btn btn-primary" href="/buybook/${name}/${book.book_id}">
                        Купить
                    </a>
                </div>
            </div>
        </#list>
        <#else>
            Нет книг в наличии
        </#list>
    </div>

```

```
</div>
</@c.page>
```

---

### 7.4.3 Код клиента-администратора системы

#### Файл bookEdit.ftlh

```
<#import "parts/common.ftlh" as c>
<@c.page>
<div class="form-group mt-3">
    <form method="post">
        <div class="form-group">
            <input type="number" class="form-control" name="amount" placeholder="Количество экземпляров" required min="0">
        </div>
        <input type="hidden" name="_csrf" value="{_csrf.token}" />
        <div class="form-group">
            <button type="submit" class="btn btn-primary">Обновить количество</button>
        </div>
    </form>
</div>
</@c.page>
```

#### Файл main.ftlh

```
<#import "parts/common.ftlh" as c>
<#include "parts/security.ftlh">
<link href="http://fonts.googleapis.com/css?family=PT+Sans:regular,italic,bold,bolditalic" rel="stylesheet" type="text/css" />

<@c.page>
<#if isAdmin>
    <a class="btn btn-primary" data-toggle="collapse" href="#collapseExample" role="button" aria-expanded="false"
        aria-controls="collapseExample">
        Добавить Книгу
    </a>
    <div class="collapse" id="collapseExample">
        <div class="form-group mt-3">
            <form method="post">
                <div class="form-group">
                    <input type="text" class="form-control" name="title" placeholder="title" required>
                </div>
                <div class="form-group">
                    <input type="text" class="form-control" name="isbn" placeholder="isbn" required minlength="13" maxlength="13">
                </div>
                <div class="form-group">
                    <input type="number" class="form-control" name="year_publ" placeholder="year_publ" required min="1425"
                        max="2022">
                </div>
                <div class="form-group">
                    <input type="number" class="form-control" name="price" placeholder="price" required min="1">
                </div>
                <div class="form-group">
                    <input type="text" class="form-control" name="author" placeholder="author" required>
                </div>
                <div class="form-group">
                    <input type="text" class="form-control" name="genre" placeholder="genre" required>
                </div>
            </form>
        </div>
    </div>
</#if>
</@c.page>
```

```

        </div>
        <div class="form-group">
            <input type="text" class="form-control" name="publisher" placeholder="publisher" required>
        </div>
        <div class="form-group">
            <input type="number" class="form-control" name="age" placeholder="age" required min="0" max="18">
        </div>
        <div class="form-group">
            <input type="number" class="form-control" name="amount" placeholder="amount" required min="1">
        </div>
        <input type="hidden" name="_csrf" value="{{_csrf.token}}" />
        <div class="form-group">
            <button type="submit" class="btn btn-primary">Добавить</button>
        </div>
    </form>
</div>
</div>
</#if>
<div class="card-columns">
    <#list books as book>
        <div class="card bg-light my-3">
            <div class="card-body m-1">
                <h4 class="card-title" style="color:#191970">{{book.title}}</h4>
                <h6 class="card-subtitle mb-2" style="color:#00008B"> {{book.author.author_name}}</h6>
                <p class="card-text"><b>Жанр:</b> {{book.genre.genre_name}}</p>
                <p class="card-text"><b>Возрастное ограничение:</b> {{book.age.age_number}}+</p>
                <p class="card-text"><b>Издательство:</b> {{book.publisher.publisher_name}}</p>
                <p class="card-text"><b>Год издания:</b> {{book.year_publ}}</p>
                <p class="card-text"><b>ISBN:</b> {{book.isbn}}</p>
                <h5 class="card-text" style="color:#00008B"><b>{{book.price}}</b><span class="rubl">&#8399;</span></h5>
                <p class="card-text"><b>Количество:</b> {{book.amount}}</p>
                <a class="col btn btn-primary" href="/edit/{{book.book_id}}">Редактировать</a>
            </div>
        </div>
    <#else>
        Нет книг в наличии
    </#list>
</div>
</@c.page>

```

## Файл userList.ftlh

```

<#import "parts/common.ftlh" as c>
<@c.page>
    <b>Список пользователей</b>
<table>
    <thead>
        <tr>
            <th>Имя пользователя</th>
            <th>Роль</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        <#list users as user>
            <tr>
                <td>{{user.username}}</td>
                <td><#list user.roles as role>{{role}}<#sep>, </#list></td>
            </tr>
        </#list>
    </tbody>
</table>

```

```

        <td><a href="/user/${user.customer_id}">Редактировать</a> </td>
    </tr>
</#list>
</tbody>
</table>
</@c.page>

```

## Файл userEdit.ftlh

```

<#import "parts/common.ftlh" as c>
<@c.page>
    <h5>Редактирование роли пользователя</h5>
    <form action="/user" method="post">
        <#list roles as role>
            <div>
                <label><input type="checkbox" name="${role}" ${user.roles?seq_contains(role)?string("checked", "")}>
                    ${role}</label>
            </div>
        </#list>
        <input type="hidden" value="${user.customer_id}" name="userId">
        <input type="hidden" name="_csrf" value="${_csrf.token}">
        <button class="btn btn-primary mb-3" type="submit">Save</button>
    </form>
</@c.page>

```

---