Geekbrains

Разработка сервиса электронного учета книг и читателей

Программа: Разработчик

Специализация: Веб-разработка на Java

ФИО: Кондратенков Андрей Владиславович

Санкт-Петербург

2024

Оглавление

Введение

Развитие библиотечной деятельности неразделимо с внедрением передовых технологий и процессов с целью обеспечения наиболее качественного обслуживания читателей. Внедрение информационных систем – это не только тенденция времени, но и хорошая база для увеличения производительности и качества труда сотрудников библиотеки, эффективный способ своевременного обеспечения читателей необходимыми им данными и информацией, в этом заключается актуальность исследования.

Под веб–приложением понимается приложение, взаимодействующее с пользователем, который с любого устройства с помощью браузера просматривает страницы и отправляет запросы к серверной части, а сервер обрабатывает полученные запросы, производит вычисления и формирует новые web–страницы, которые отправляет обратно пользователю по протоколу HTTP. Данный способ организации работы приложений называется технология «клиент–сервер».

Тема проекта: Разработка сервиса электронного учета книг и читателей.

Цель: изучить вопрос создания информационной системы, автоматизирующей часть бизнес–процессов библиотеки с внедрением аутентификации и авторизации через ресурс сервер Keycloak.

Какие проблемы решает:

Данная дипломная работа представляет реализацию решения, которое позволяет быстро и безопасно получить пользователю доступ к информационной системе библиотеки не зависимо от места его нахождения. Для полноценной работы сотрудников требуется только наличие доступа к сети интернет и любого браузера, что предполагает возможность работы с любых доступных устройств.

Задачи:

1. произвести моделирование деятельности библиотеки
2. Изучить существующие подходы к проектированию API, предоставляющего доступ к данным.
3. Выполнить проектирование серверной части системы, выбрать технологии, которые станут использоваться в дальнейшем.
4. Разработать серверную часть системы.

Инструменты: Postman, Maven, Hibernate, Spring boot, JPA, Keycloak.

Состав команды: Кондратенков Андрей Владиславович (Разработчик).

Глава 1. Клиент-серверная архитектура.

Приложениями и сайтами одновременно могут пользоваться сотни и даже миллионы человек. Все они обращаются к одному компьютеру, который должен уметь обрабатывать запросы и присылать ответы. Такой подход называется клиент-серверной архитектурой. Она описывает, как происходит работа с пользователями, где хранятся данные и как обеспечивается их защита.

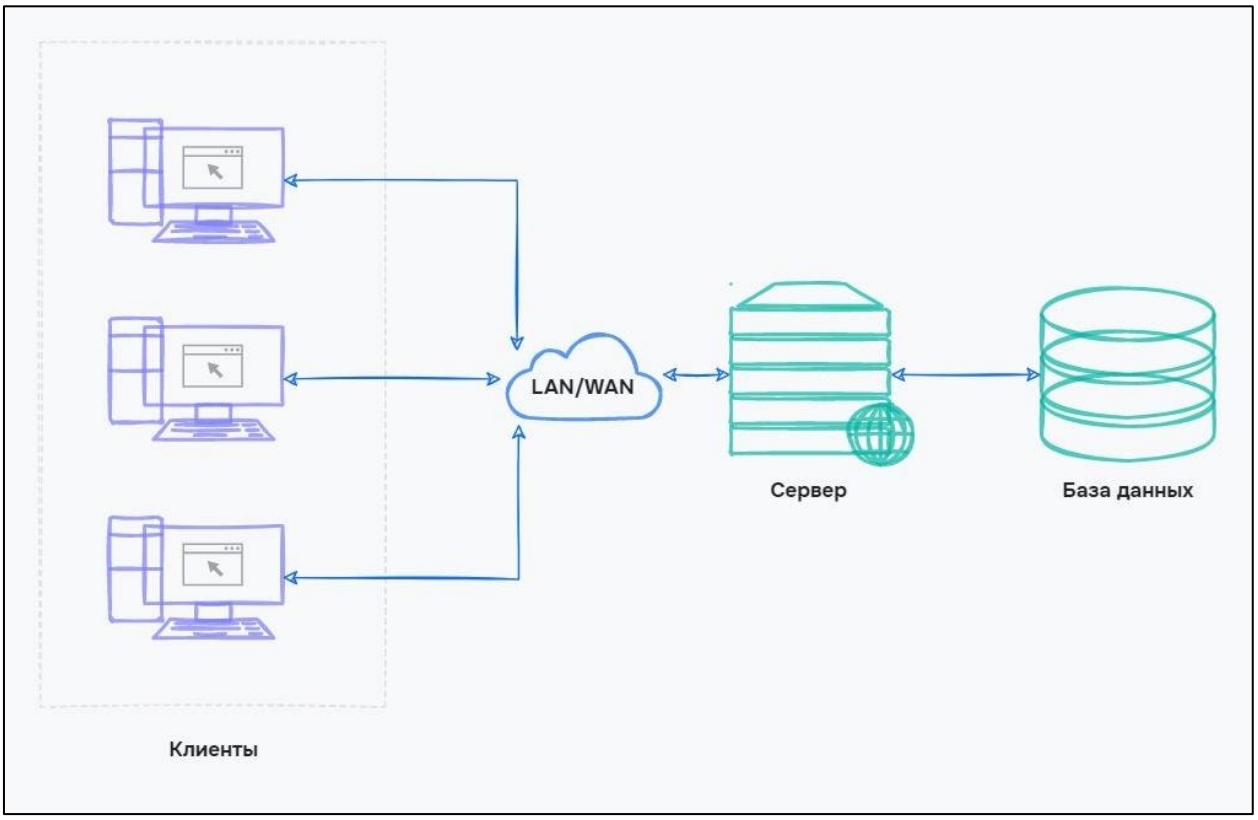
1.1 Клиент-серверная архитектура и ее компоненты

В клиент-серверной архитектуре используется три компонента:

Клиент - программа, которую мы используем в интернете. Чаще всего это браузер, но может быть и другая отдельная программа

Сервер - компьютер, на котором хранится сайт или приложение. Когда мы заходим на сайт библиотеки, мы обращаемся к серверу, на котором находится сайт.

База данных - программа, в которой хранятся все данные приложения. Для библиотеки это будет база читателей, книг и выдач книг.



1.2 Особенности клиента

Клиент - это всегда программа. Ее назначение - дать пользователю удобный способ взаимодействия с сервером.

1.3 Особенности сервера

Сервер - выделенный или специализированный компьютер для выполнения сервисного программного обеспечения. Основная задача сервера - бесперебойная работа и возможность обрабатывать миллионы запросов от пользователей.

Если сервер выполняет функции приложения и базы данных, то такая архитектура называется **двухуровневой**. Такой подход используют для небольших приложений, где нет большого количества клиентов. Хоть такой способ и проще, но его надежность небольшая. Если сервер взломают, то злоумышленники получат все данные.

Чтобы решить проблему безопасности в клиент-серверной архитектуре, используют базу данных. Она хранится отдельно от сервера. Сервер в этом случае выполняет роль логической машины, которая обрабатывает данные, но не хранит их. Данная модель будет использована в данной работе в рамках реализации интернет-магазина.

1.4 Особенности клиент-серверного взаимодействия

Зачастую, взаимодействие между клиентом и сервером происходит по одному из специальных протоколов: IP, HTTP, WebSocket и т. д. Более того, инициатором данного взаимодействия в большинстве случаев является именно клиент.

Если абстрагироваться от частных случаев и особенностей протоколов взаимодействия, то можно сказать, что клиент серверное взаимодействие происходит по следующей схеме:

1) клиент отправляет серверу некоторый запрос;

2) сервер выполняет какие-то действия на основе этого запроса;

3) сервер отправляет ответ.

В данном случае, можно сказать, что сервер предоставляет некоторый API для запросов клиента. К этому API предъявляется множество требований разной направленности.

Во-первых, к нему предъявляются функциональные требования. Это означает, что API должен предоставлять полный функционал системы, а именно позволять сделать с помощью запросов все обусловленные на этапе проектирования действия.

Также, API должен быть производительным. Это значит, что он не должен перегружать сеть взаимодействия, не зависимо от протокола, а также увеличивать вычислительную сложность на обработку ответов. Чтобы объяснить, что это значит, давайте рассмотрим следующий пример: сервер возвращает клиенту граф. Естественно, граф можно представить в разном виде: список смежностей, матрица смежностей, список ребер и т.д. Но если клиенту нужен граф в виде списка смежностей, а сервер возвращает его как матрицу смежностей, то клиенту необходимо дополнительно преобразовать его, что является повышением вычислительной сложности.

Следует отметить, что серверные API делятся на два вида: публичные и приватные. Публичным API называется API, который находится в открытом доступе и каждый может им воспользоваться. Такие API предоставляют крупные компании для работы с их данными или приложениями. К приватному же API зачастую имеет доступ только несколько серверов frontend и он создается специально для них в рамках некоторого одного проекта.

Последнее требование к API, которое следует обсудить – это его удобство. Под этим подразумевается то, что пользователи API должны без особых проблем использовать его после небольшой инструкции, или же полностью интуитивно.

1.5 Анализ существующих подходов к проектированию API

Для рассмотрения был выбран подход REST API.

Данные подходы будут рассмотрены исходя из следующих критериев:

1) возможность избежать over-fetching;

2) возможность избежать under-fetching;

3) структура кода.

Начнем мы с одного из старейших подходов в вебе - REST, который использует все возможности HTTP.

REST - Representational state transfer - архитектурный подход, который описывает набор правил, как организовать общение систем. У данного подхода есть ряд требований - отсутствие состояния, единообразие интерфейсов. В простом понимании это URI, который описывает, что за ресурс мы запрашиваем: список элементов или один элемент. Кроме того, используются HTTP-глаголы или HTTP-методы - какие действия мы произведем с ресурсом.

Методов достаточно много. Как правило, мы имеем дело с:

POST /users для создания нового пользователя;

GET /users для получения списка пользователей;

GET /users/1 для получения одного пользователя с ID=1;

PUT или PATCH /users/1 для изменения пользователя;

DELETE /users/1 для удаления.

В ответе от сервера мы получаем запрошенный ресурс / сообщение об ошибке и трехзначный код ответа. Код ответа показывает, что произошло с запросом пользователя и что, в случае ошибки, пошло не по плану. К примеру:

200 – запрос выполнен успешно;

401 – необходимо авторизоваться;

404 – ресурс не найден;

505 – сервер временно недоступен.

Такой подход предполагает наличие некоторого количества конечных точек серверного API, через которые и происходит взаимодействие с клиентом. Зачастую, REST-API подразумевает реализацию CRUD-интерфейса, используя

перечисленный выше протокол. В расширенном REST-API обычно добавляются вспомогательные поля для GET-запросов, такие как фильтр, пейджинг, сортировка и т.д. Стоит отметить, что добавление этих полей позволяет частично решить проблему over-fetching, т.к. клиент может запрашивать только те сущности, которые ему нужны, но при этом не позволяет выбирать поля, нужные клиенту, отправляя вместо этого все. Несмотря на то, что кажется, что этот подход достаточно строгий и структурированный, на самом деле ничего не мешает использовать эти методы и коды не по назначению, не структурировать код.

Глава 2. Проектирование архитектуры и структуры приложения.

2.1 База данных

В проекте используется база данных H2 для упрощения развертывания приложения, при желании можно поменять в application.yml на любую реляционную базу данных.

```
datasource:
  url: jdbc:h2:mem:test
  username: user
  password:
  driver-class-name: org.h2.Driver
```

2.2 Руководство пользователя

В проекте реализованы три основные сущности:

1. Book (Книга) - параметры: id, название.

2. Reader (Читатель) - параметры: id, имя.

3. Issue (Выдача) - параметры: id, id книги, id читателя, имя читателя, название книги, дата выдачи, дата сдачи, дата сдачи.

В приложении реализован следующий функционал:

1. Получение, добавление, изменение и удаление читателя;

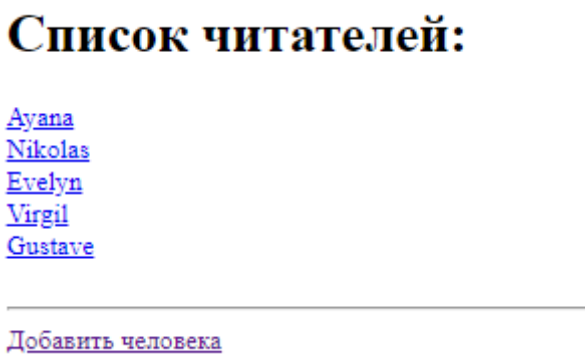
2. Получение, добавление, изменение и удаление книги;

3. Получения полного списка всех книг, людей и выдач;

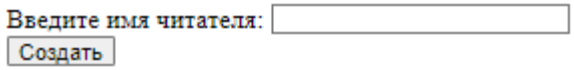
Все эндпоинты вы можете посмотреть через Swagger UI: http://localhost:8180/swagger-ui/index.html

2.2.1 Страницы, разработанные в приложении:

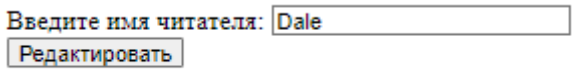
1. Список всех читателей: http://localhost:8180/ui/reader/all



2. Добавление читателя: http://localhost:8180/ui/reader/new



3. Редактирование данных читателя: http://localhost:8180/ui/reader/{id}/edit



4. Список всех книг: http://localhost:8180/ui/book/all

Список книг:

The Other Side of Silence
Lilies of the Field
Dance Dance Dance
Shall not Perish
A Handful of Dust

Добавить книгу

5. Добавление новой книги: http://localhost:8180/ui/book/new

Введите название книги:
Создать

6. Редактирование данных книги: http://localhost:8180/ui/book/{id}/edit

Введите название книги: The Other Side of Silence
Редактировать

7. Выдачи книг по читателю: http://localhost:8180/ui/issue/reader/{id}

Имя читателя: Dale

Книги:
Shall not Perish

Редактировать
Удалить читателя

8. Полный список выдачи книг: http://localhost:8180/ui/issue/all

Список выдачи книг:

ID	ID Читателя	Имя читателя	ID Книги	Название книги	Выдана	Возвращена
91a90024-0dd6-4a96-9b43-a01f332f4950	1095fd17-2b80-4c95-9933-32a02d8ccd40	Maximus	68097373-abed-4602-938f-7c6c172967d2	Look Homeward, Angel	2024-10-16	
b3b94dbc-58ec-4751-91c9-00eb5df73e34	120225f6-ac85-4be5-bc54-c430b570cb53	Elise	111bdcc8-a4e0-47aa-8237-30003ffef17d	In Death Ground	2024-10-07	
00efcf9c-84f9-4aeb-8181-4d27b54d9655	ed46dd94-2bbc-411d-a0ce-dd0f4501851a	Sonia	68097373-abed-4602-938f-7c6c172967d2	Look Homeward, Angel	2024-01-12	
3045aefa-d0dd-4ab5-b188-110afb6b91db	1d1aa832-543e-41e6-b354-8265a66e2653	Braulio	b392e725-200d-474f-8171-3e535a93b78a	Bury My Heart at Wounded Knee	2024-03-17	
94b21e9c-280c-4690-859c-a9cd531df467	6380f877-0874-4cd0-a1ab-6ea17ae2f2ed	Amy	905e9d00-7b5f-46e4-b4de-2efd7cc5bbb2	Postern of Fate	2024-10-24	

2.2.2 Реализация аутентификации и авторизации

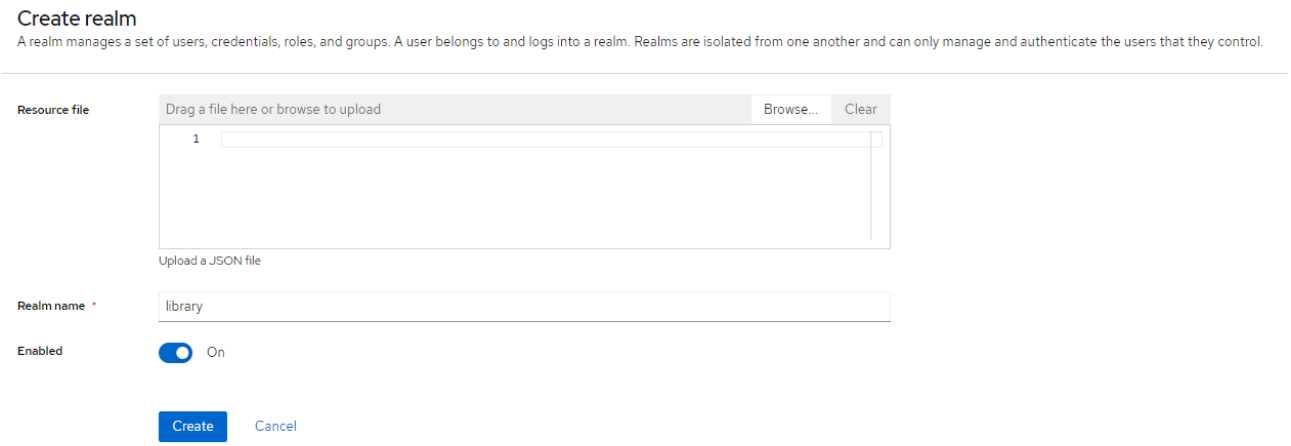
Вход в систему доступен только при получении JWT-токена на сервере авторизации (Keycloak), в качестве клиента можно использовать Postman или Keycloak.

Для доступа к ресурсам приложения нужно сделать следующее:

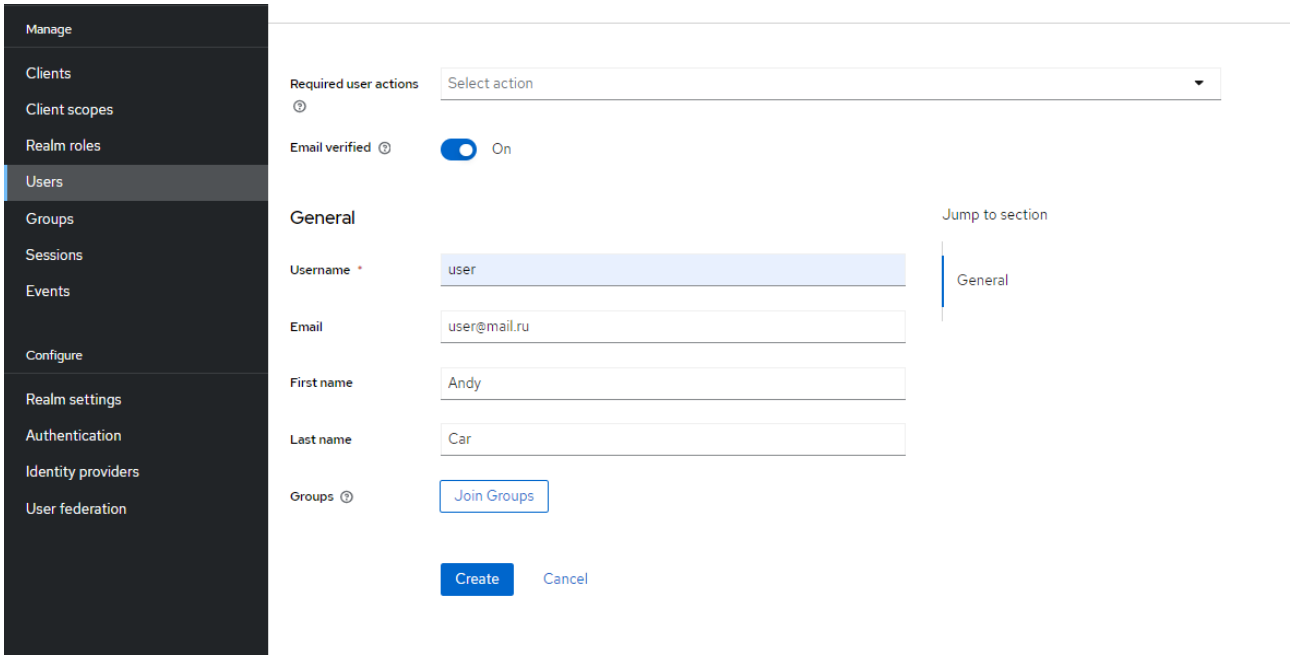
1. Запустите приложение
2. В командной строке выполните команду (должен быть установлен Docker):

docker run --name keycloak -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:latest start-dev

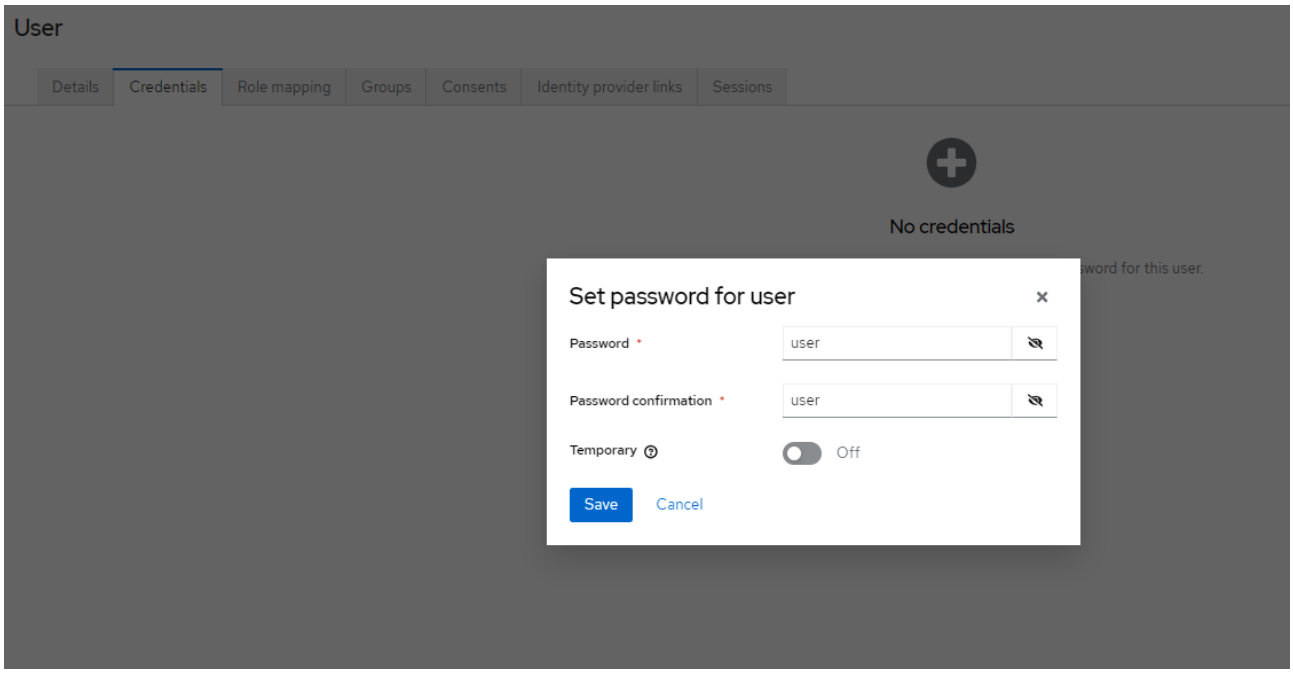
3. Зайдите в админку Keycloak в браузере: http://localhost:8080/
4. Введите логин “admin”, пароль “admin”
5. Создайте новый realm (изолированное пространство пользователей):



6. Добавьте пользователя через вкладку "Users", нажмите "Create new user":

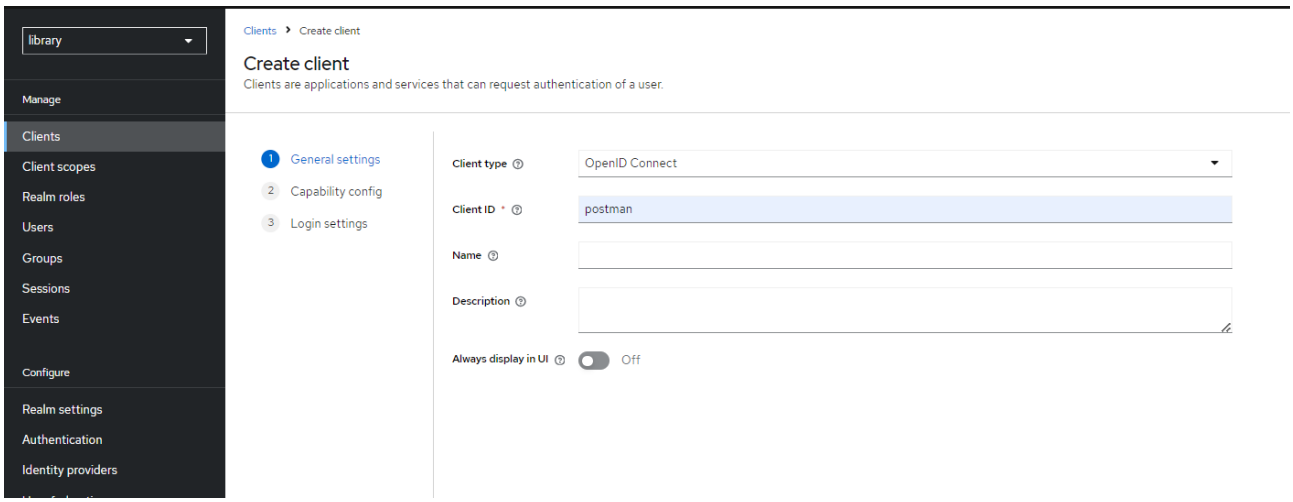


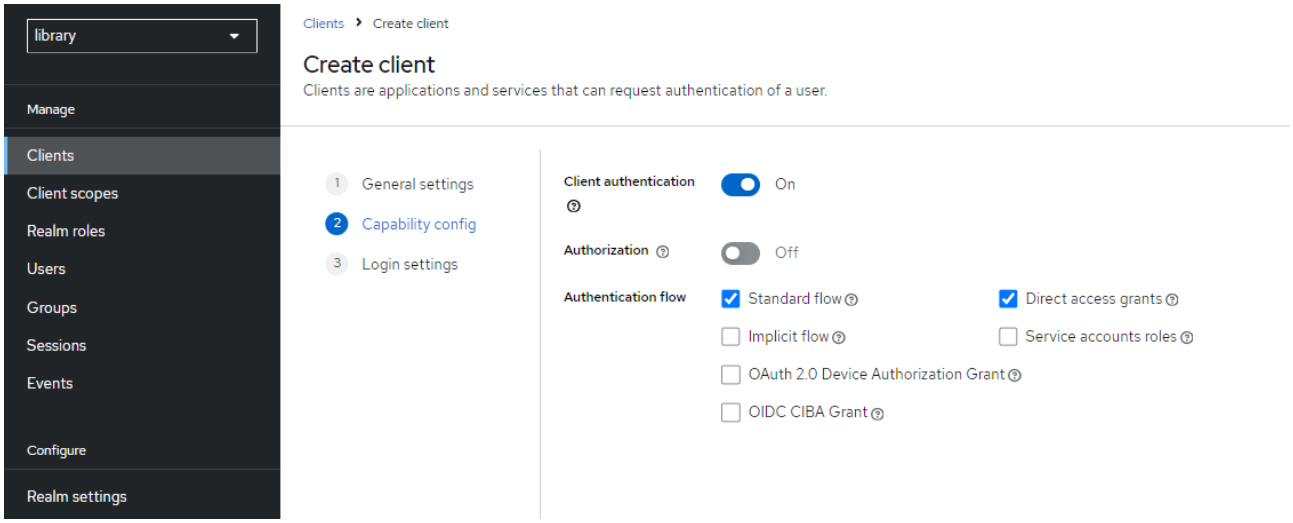
7. Далее зайдите в раздел "Credentials" и нажмите "Set password", после чего введите пароль и выключите обновление пароля:

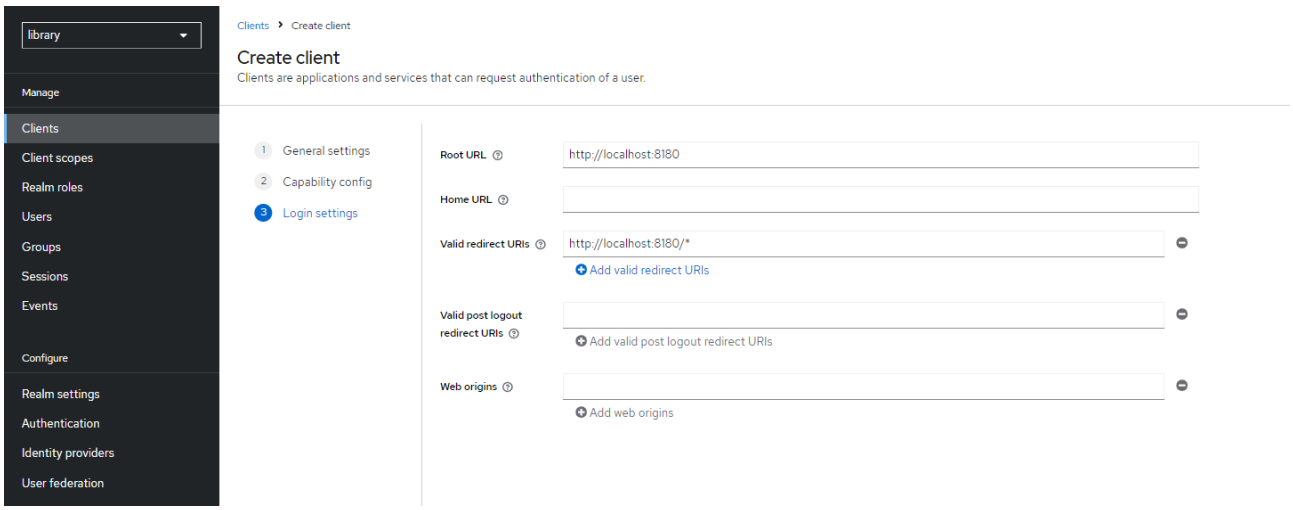


8. Перейдите во вкладку "Clients" и нажмите "Create client"

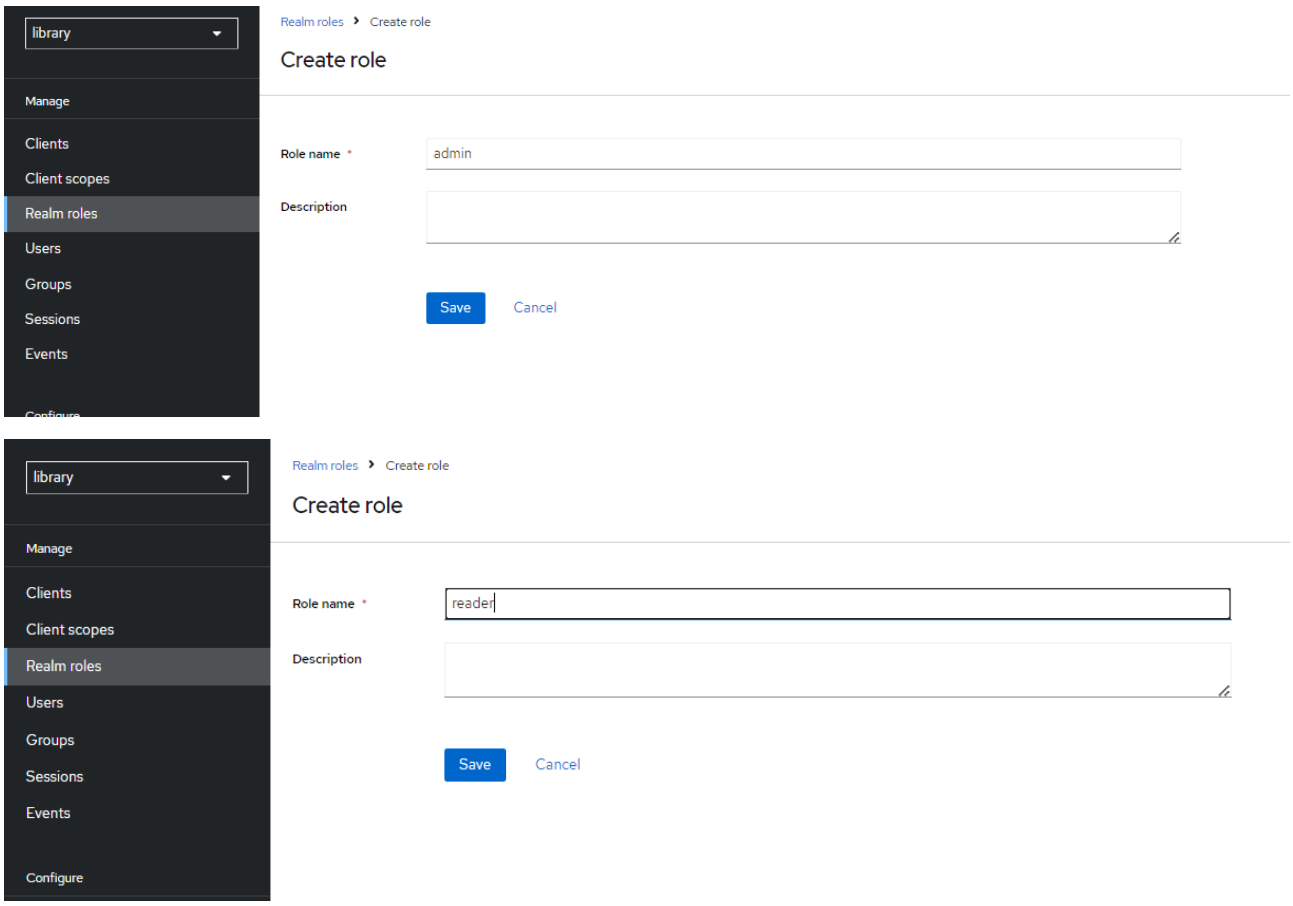
9. Добавьте клиента, будем использовать Postman или Keycloak client:



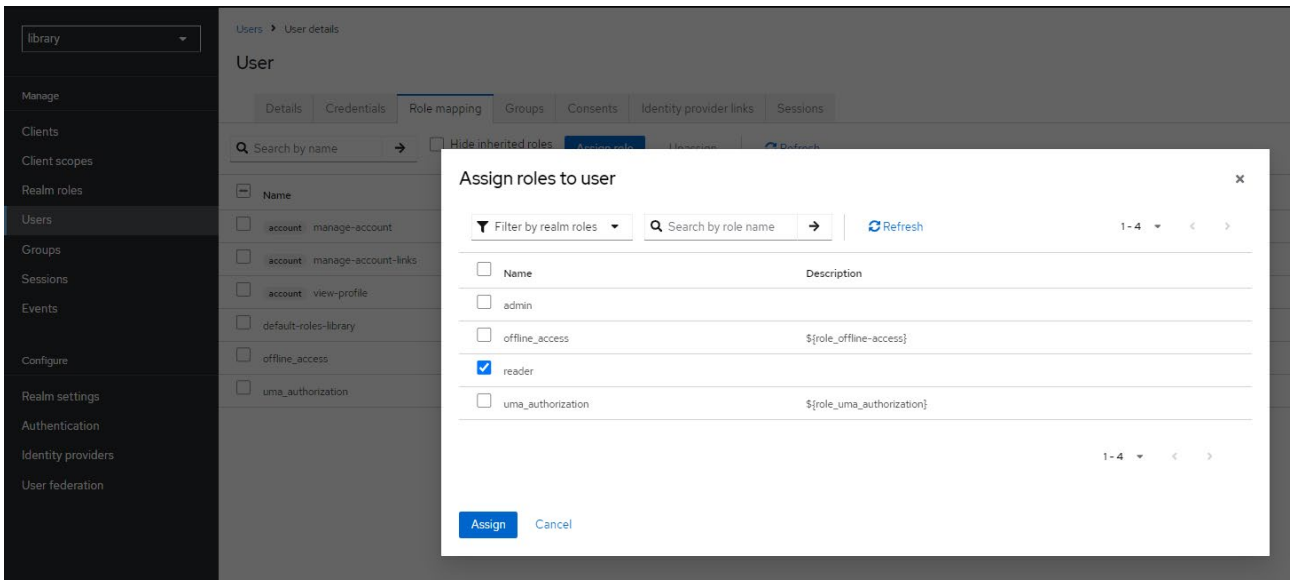




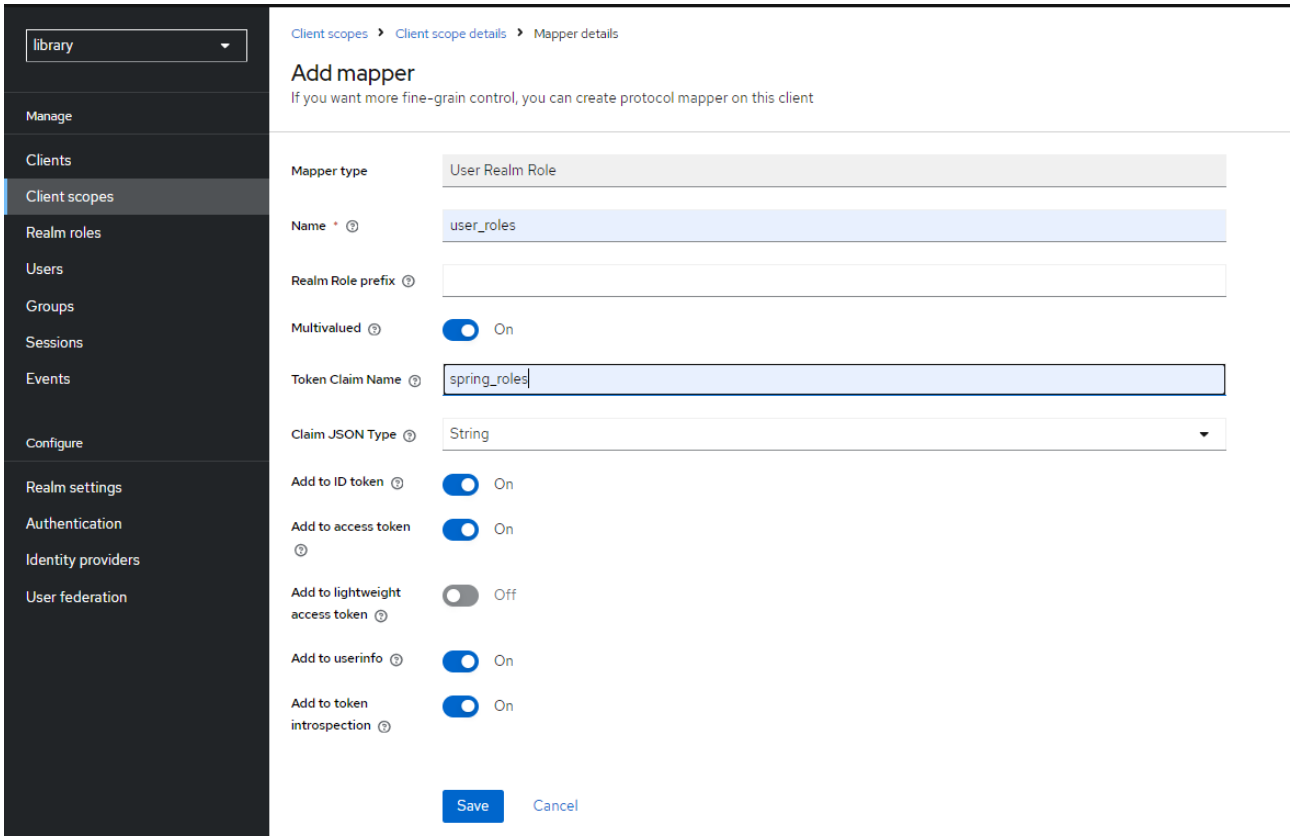
10. Добавьте роли пользователей, в проекте предусмотрены роли admin и reader:



11. Добавьте пользователю user роль reader (Users -> Role mapping -> Assign role):

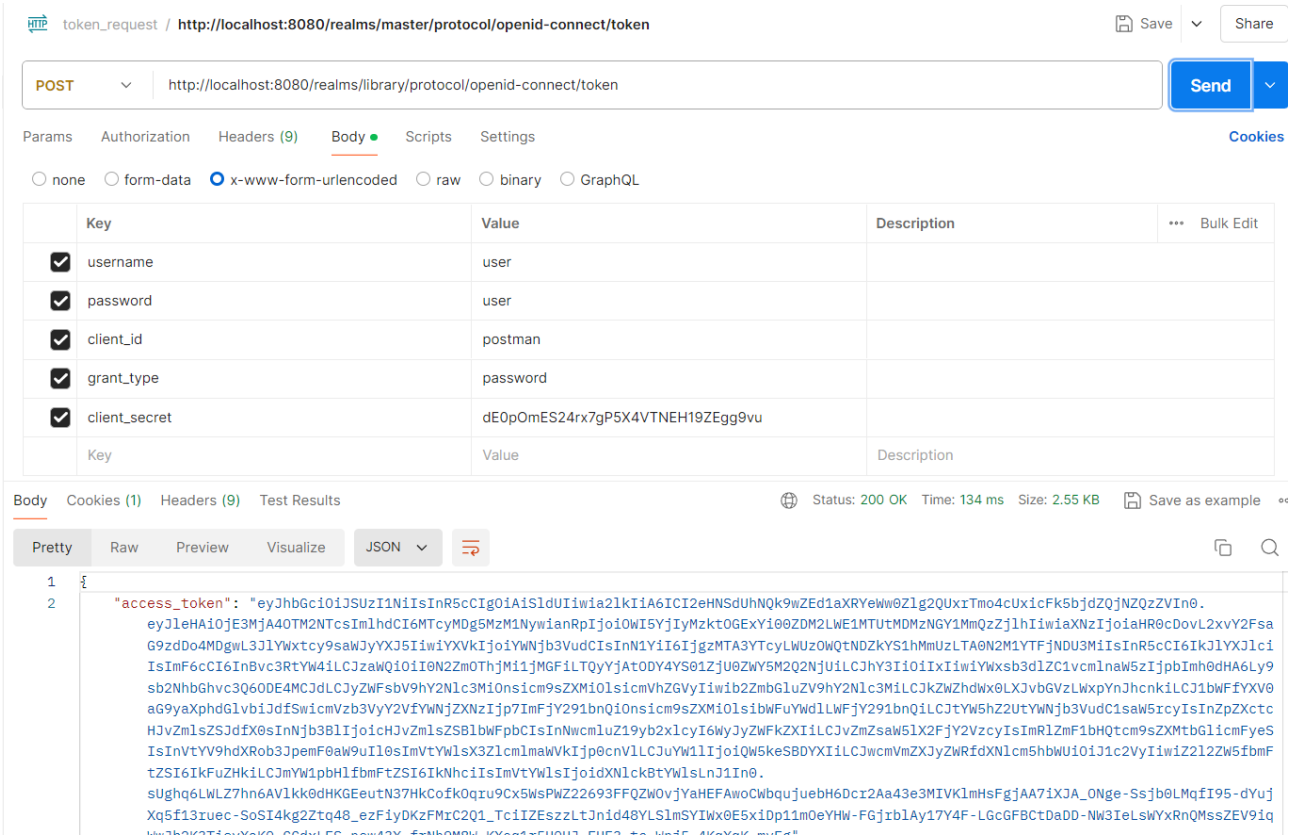


12. Создайте свой маппер для ролей юзера(Client Scopes -> roles -> Mappers -> Add Mapper By configuration -> User Realm Role):

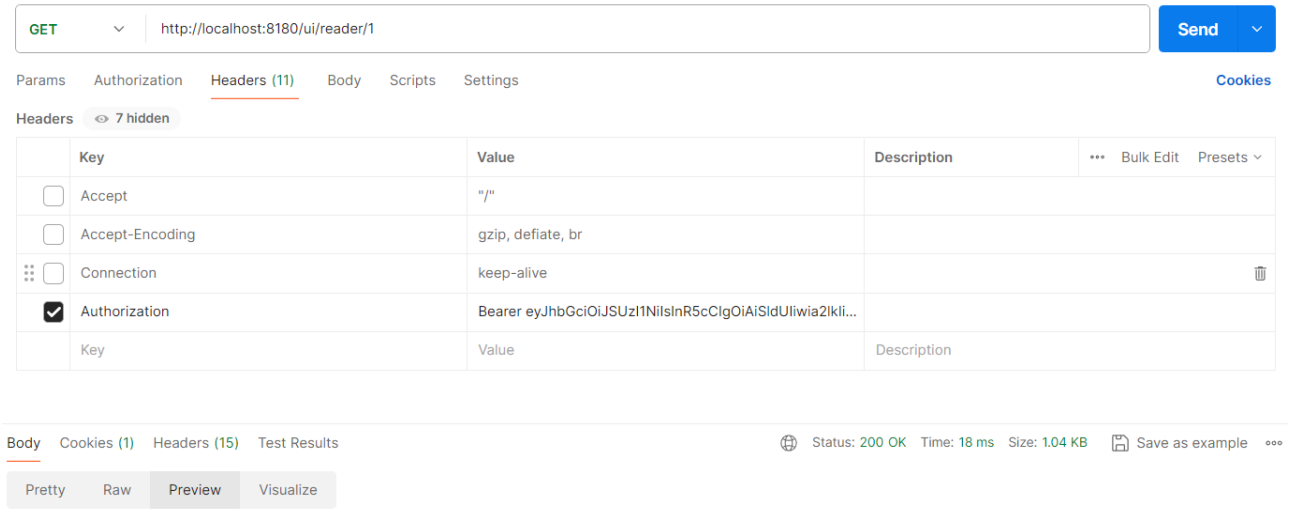


Если в качестве клиента используете Postman:

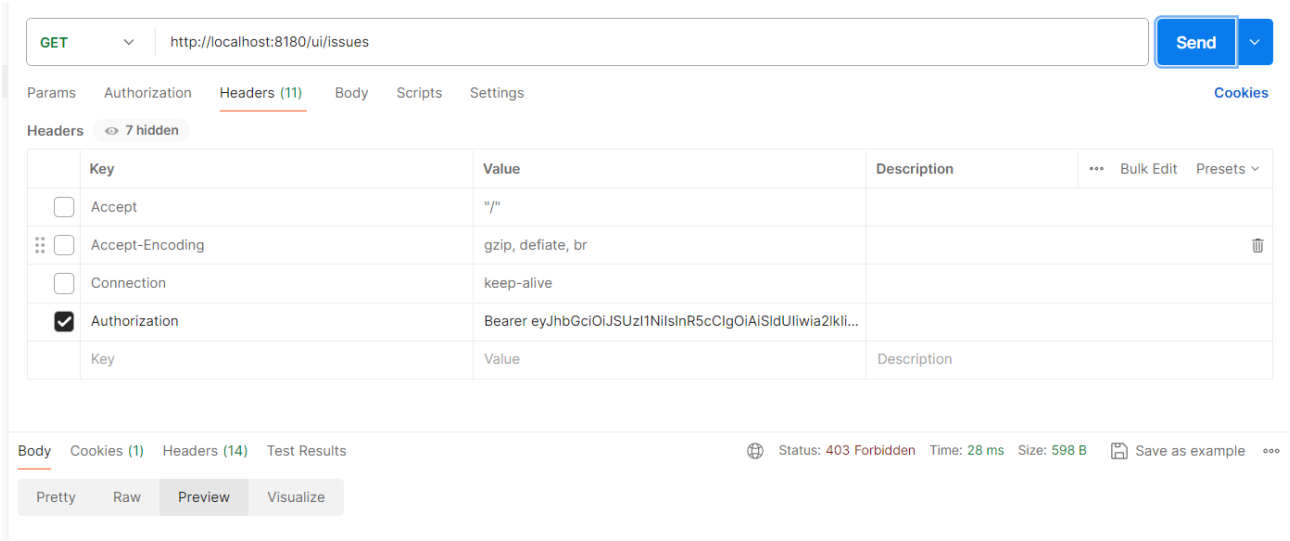
1. Отправьте POST-запрос в Keycloak для получения jwt-токена, client_secret следует скопировать из админки (Clients -> postman -> Credentials):



2. Отправьте GET-запрос в приложение, например, для получения читателя по id, предварительно скопируйте "access_token" в post-ответе и вставьте в заголовок Authorization:

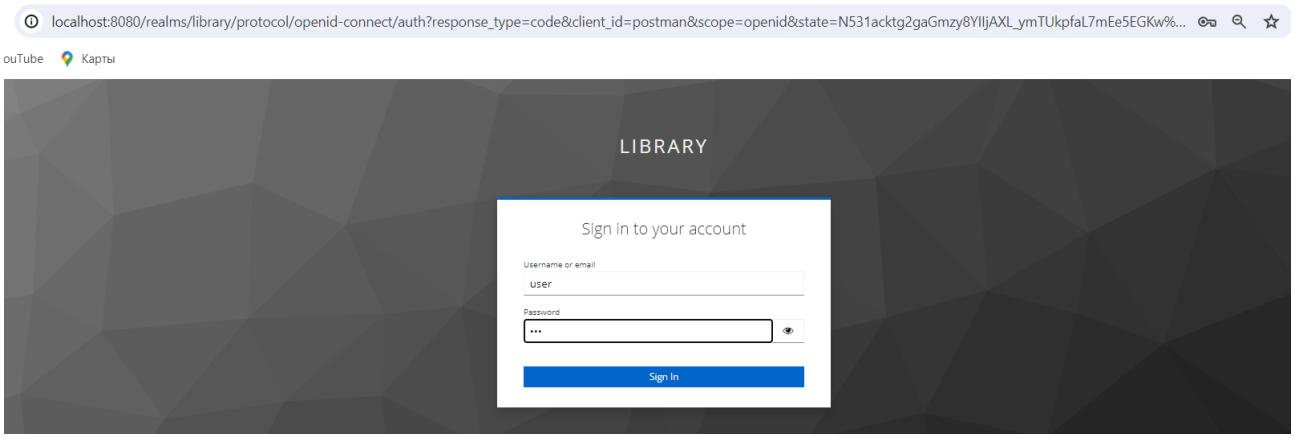


3. Если отправите запрос на получение всех выдач книг, то вы не получите информацию, т.к. у текущего пользователя нет таких прав:

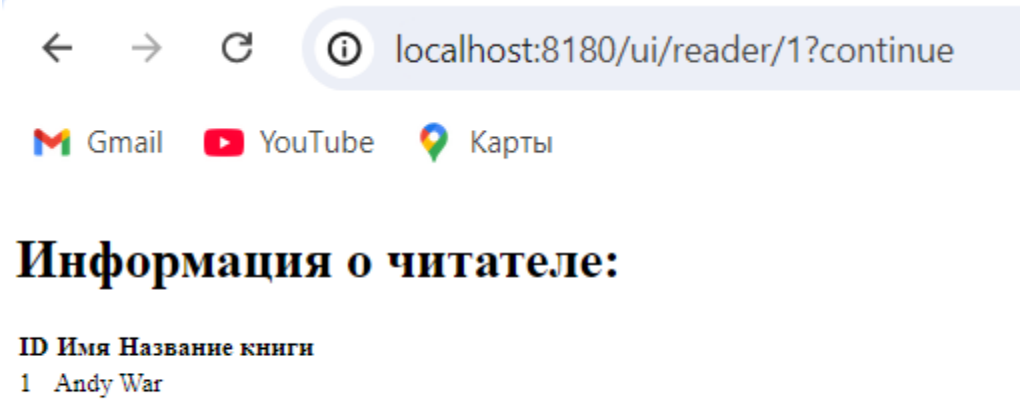


Если в качестве клиента используете браузер:

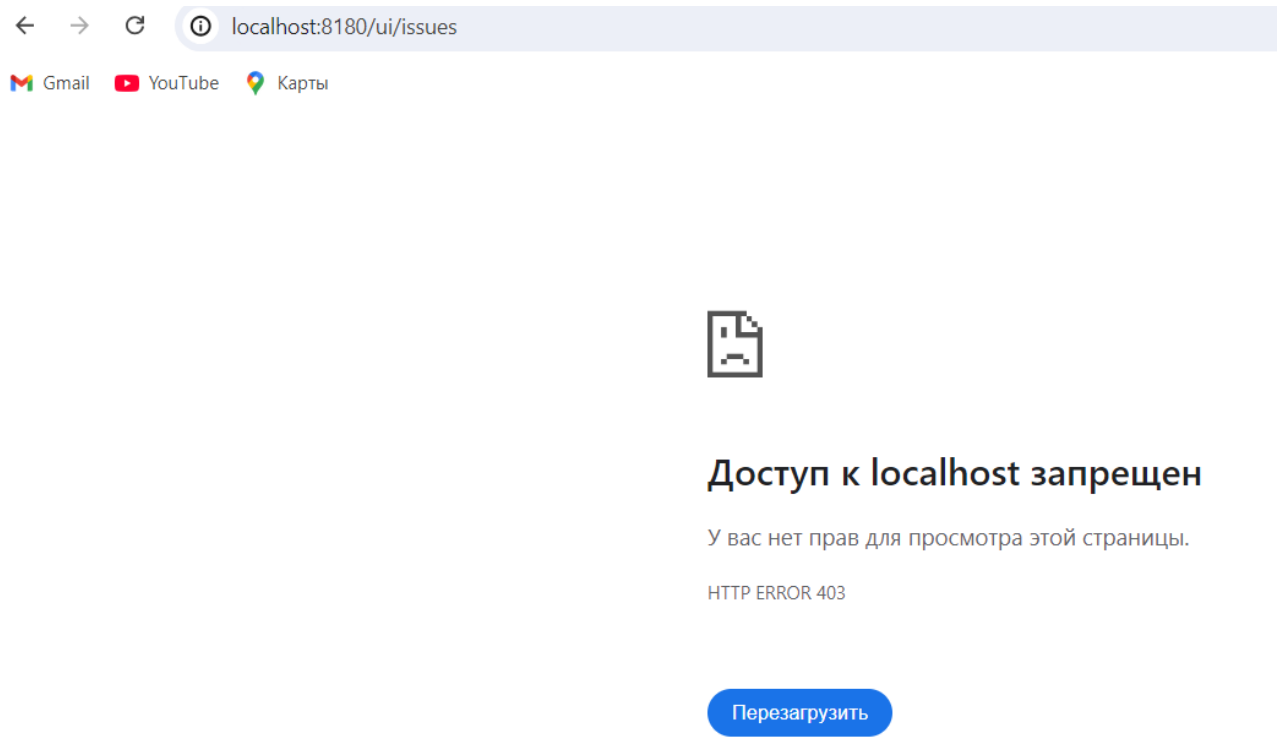
1. При попытке запроса к разным ресурсам, сначала нужно заполнить форму входа (логин и пароль) Keycloak-client, раннее вами был создан "user" с паролем "user":



2. Запрос на получение читателя по id:



3. Запрос на получение выдач книг, прав не хватает:


localhost:8180/ui/issues
Gmail
YouTube
Карты
Доступ к localhost запрещен
У вас нет прав для просмотра этой страницы.
HTTP ERROR 403
Перезагрузить

Заключение

В эпоху цифровых носителей, людям все легче черпать знания из разных источников, это повышает потребность в онлайн хранилищах и площадках, где можно найти интересующую нас литературу.

В рамках данного проекта была разработана информационная система. Система, предназначенная для хранения, поиска и обработки информации. В качестве инструментов разработки использовался универсальный фреймворк с открытым исходным кодом для Java-платформы, именуемый Spring. В качестве базы данных было использовано H2. В качестве сборщика проектов использовался Maven, являющейся подмножеством XML. Была использована Java 21 описывающая систему управления сохранением Java-объектов в удобной виде, и использующий сущности (entity). Сущность - это легковесный хранимый объект бизнес логики.

Для обеспечения безопасности были разработаны сервисы аутентификации и авторизации, поддерживаемые JWT-токеном через ресурс сервер Keycloak.

Данная работа позволила мне расширить познания в веб-разработке, поскольку был использован один из самых много функциональных и востребованных фреймворков на рынке. Итоговый проект можно считать как начальную часть крупного проекта, который можно запустить на отдельном сервере и при желании улучшить для дальнейшего коммерческого использования.

Список используемой литературы

1 Бауэр К. Java Persistence API и Hibernate [Текст] / Г.Кинг –ДМК Пресс,2018. – 652 с.

2 Моргунов Е. П. PostgreSQL. Основы языка SQL: учеб. Пособие [Текст] /
Е. П. Моргунов; под ред. Е. В. Рогова – БХВ-Петербург, 2018. - 336 с.

3 Уоллс К. Spring в действии [Текст]. / Уоллс К. ДМК Пресс, 2015. – 754
с.

4 Spring Framework Reference Documentation [Электронный ресурс] –
Режим доступа: https://spring-projects.ru/projects/spring-framework/.

5 Ликнесс Д. Бессерверные приложения: архитектура, шаблоны иреализация в Azure ,/ Ликнесс Д. - Microsoft Developer Division, 2018 г. – 60 с.

6 Bootstrap Documentation v4.1 [Электронный ресурс] – Режим доступа:https://getbootstrap.com/docs/4.1/getting-started/introduction/.

7 П. Нимейер, Леук Д. Программирование на Java [Текст]./ Леук Д. –Эксмо, 2017 г. - 1216 с.

8 Уэйн К. Алгоритмы на Java. Роберт Седжвик [Текст]. / Седжвик Р –Вильямс, 2019 г. – 848 с.

9 Обзор способов и протоколов аутентификации в веб-приложениях [Электронный ресурс]. / Д. Выростков, тематический блок Habr,2015. –Режим доступа: https://habr.com/ru/company/dataart/blog/262817/.

10 Краткое знакомство с Maven [Электронный ресурс]. / Сергей Штукатуров – портал Tproger, 2019. – Режим доступа: https://tproger.ru/articles/maven-short-intro/.

11 Ю.А.Родичев. Нормативная база и стандарты в области информационной безопасности [Текст]. / Ю. Родичев – Питер,2017. – 256 с.

Приложения

1. Входная точка в приложение

```java
@SpringBootApplication
@EnableConfigurationProperties(IssueProperties.class)
public class Application {
    no usages  new *
    public static void main(String[] args) { SpringApplication.run(Application.class, args); }

}
```

2. Пример эндпоинта для получения всех книг в JSON

```java
@GetMapping("/all")
@Operation(
        summary = "Get all books",
        description = "You'll get the list of all allowed books"
)
public ResponseEntity<List<Book>> getAll() {
    List<Book> books = bookService.getAllBooks();
    if (books.isEmpty()) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
    } else {
        return ResponseEntity.ok(books);
    }
}
```

3. Пример эндпоинта для получения выдач книг читателю в браузере

```java
no usages  andropol1 *
@GetMapping("/reader/{id}")
@Operation(
        summary = "Get all issues of the specific reader",
        description = "You'll get the list of all issues of specific reader. You need to write reader's id."
)
public String getReaderIssues(@PathVariable UUID id, Model model) {
    Optional<Reader> reader = readerService.getReaderById(id);
    if (reader.isPresent()) {
        model.addAttribute( attributeName: "reader", reader.get());
        model.addAttribute( attributeName: "issues", issueService.getAllIssuesByReaderId(id));
        return "issuesByReaderId";
    } else {
        return "redirect:/ui/reader/all";
    }
}
```

4. Пример сущности книги

```java
@Data
@Entity
@Table(name = "books")
@Schema(name = "Книга")
@NoArgsConstructor
@AllArgsConstructor
public class Book {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Schema(name = "ID")
    @Column(columnDefinition = "uuid")
    private UUID id;
    1 usage
    @Column(name = "name")
    @Schema(name = "Имя")
    private String name;
```

5. Пример слоя репозитория книг

```java
12 usages  new *
public interface ReaderRepository extends JpaRepository<Reader, UUID> {
    1 usage  new *
    Optional<Reader> findReaderByName(String name);
}
```

6. Конфигурация безопасности

```java
@EnableWebSecurity
public class SecurityConfiguration {
    no usages  new *
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    no usages  new *
    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        return httpSecurity.authorizeHttpRequests(conf -> conf
                        .requestMatchers( ...patterns: "/ui/reader/**", "/ui/books/**").authenticated()
                        .requestMatchers( ...patterns: "/ui/issue/**").hasAuthority("admin")
                        .requestMatchers( ...patterns: "/ui/issue/reader/**").hasAuthority("reader")
                        .anyRequest().permitAll())
                .csrf(AbstractHttpConfigurer::disable)
                .oauth2ResourceServer(conf -> conf
                        .jwt(Customizer.withDefaults()))
```

7. Слой сервиса с бизнес-логикой для книг:

```java
@Service
@RequiredArgsConstructor
public class BookService {
    5 usages
    private final BookRepository bookRepository;

    3 usages  andropol1 *
    public Optional<Book> getBookById(UUID id) {
        return bookRepository.findById(id);
    }

    2 usages  andropol1
    public List<Book> getAllBooks() { return bookRepository.findAll(); }

    1 usage  andropol1 *
    public void deleteBookById(UUID id) { bookRepository.deleteById(id); }
```

8. Валидация читателя

```java
@Component
public class ReaderValidator implements Validator {
    2 usages
    private final ReaderRepository readerRepository;

    no usages
    @Autowired
    public ReaderValidator(ReaderRepository readerRepository) { this.readerRepository = readerRepository; }

    @Override
    public boolean supports(Class<?> clazz) { return Reader.class.equals(clazz); }

    @Override
    public void validate(Object target, Errors errors) {
        Reader reader = (Reader) target;

        if (readerRepository.findReaderByName(reader.getName()).isPresent()) {
            errors.rejectValue( field: "name",  errorCode: "",  defaultMessage: "Человек с таким именем уже существует");
        }
```