МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

“Ижевский государственный технический университет

имени М.Т. Калашникова”

Кафедра “Программное обеспечение”

Работа защищена с оценкой

«\_\_\_\_\_\_\_\_\_\_»

Дата \_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине “Современные операционные системы и базы данных”

на тему “Разработка приложения для СУБД Solr”

Выполнил:

студент группы Б17-191-2 Иванов В.М.

Принял:

к.т.н., доцент кафедры Старыгин А.В.

Рецензия:

Степень достижения поставленной цели работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полнота разработки темы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

уровень самостоятельности работы обучающегося\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

недостатки работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ижевск 2023

**Содержание**

# **Постановка задачи**

В курсовой работе требуется:

1. Разработать базу данных (бд) с использованием не реляционного (NoSQL) подхода.
2. Разработать клиентское приложение для работы с созданной базой данных.

Требования к клиентскому приложению

1. Наличие общего интерфейса, позволяющего работать с базой данных
2. При работе с таблицами обеспечивается:

* Перемещение по записям
* Добавление новых записей
* Поиск записей по отдельным полям
* Задание фильтра по отдельным полям

**Ход работы**

В качестве предметной области было выбрано «Агрегатор фильмов». Приложение будет обеспечивать выдачу фильмов по заданному жанру и/или режиссёру. Поиск будет осуществляться по уникальному идентификатору (id), полнотекстовой поиск (по всем значимым полям), также будет реализован фильтр по жанрам. Каждая запись будет иметь вид, описанный в таблице 1.

Таблица 1. Описание полей и значений записей в базе данных

|  |  |  |
| --- | --- | --- |
| Название поля | Тип значения | Описание поля |
| id | String | Идентификатор записи |
| name | String | Название фильма |
| initial\_release\_date | Date | Дата выхода фильма |
| directed\_by | Array of String | Список режиссёров |
| genre | Array of String | Список жанров, к которым относят фильм |
| full\_text | String | Объединение значимых полей в виде строки, разделённой пробелами |
| \_version\_ | Unsigned Long | Версия записи |

Поля id, name, initial\_release\_date и genre определяются из предоставленных исходных данных. Поле full\_text и \_version\_ генерируется автоматически, при добавлении записи в базу данных.

Пример документа в базе данных в формате JSON:

{

"id":"9\_2005",

"name":"9",

"directed\_by":["Shane Acker"],

"genre":["Computer Animation",

"Animation",

"Apocalyptic and post-apocalyptic fiction",

"Science Fiction",

"Short Film",

"Thriller",

"Fantasy"],

"initial\_release\_date":"2005-04-21T00:00:00Z",

"full\_text":"9 Shane Acker Computer Animation Animation Apocalyptic and post-apocalyptic fiction Science Fiction Short Film Thriller Fantasy",

"\_version\_":1761739909715984384

}

Для хранения вышеописанных документов будет использована NoSQL СУБД Apache Solr. Apache Solr – платформа полнотекстового поиска с открытым исходным кодом, основанная на проекте Apache Lucene. Она включает в себя документоориентированную базу данных, собственный язык запросов, а также ряд инструментов, облегчающих создание индекса данных, разбиение бд на кластеры и размещение на сервере.

Настроить бд Solr можно, изменив файл «managed-shema.xml» соответствующего хранилища. Добавим вышеописанные поля в схему, это будет выглядеть следующим образом:

...

<field name="directed\_by" type="text\_general"/>

<field name="full\_text" type="text\_en\_splitting\_tight" indexed="true" stored="true"/>

<field name="genre" type="text\_general"/>

<field name="id" type="string" multiValued="false" indexed="true" required="true" stored="true"/>

<field name="initial\_release\_date" type="pdate" indexed="true" stored="true"/>

<field name="name" type="string" indexed="true" stored="true"/>

…

Описание атрибутов конфигурации:

* Атрибут «name» обозначает имя поля;
* «type» тип поля. Например, string или text используется для строковых значений, а pdate для дат;
* Если значение атрибута «Multivalued» истинно, значит поле является массивом данных;
* «indexed» означает, что это поле может участвовать в поиске по запросу;
* «stored» означает, что значение этого поля можно получить в ответе, а не только осуществлять по нему поиск.

Взаимодействовать с базой данных на Apache Solr возможно с помощью HTTP запросов, но в большинстве случаев это не удобно и не безопасно. Поэтому взаимодействие с бд будет реализовано по схеме, описанной на рисунке 1.

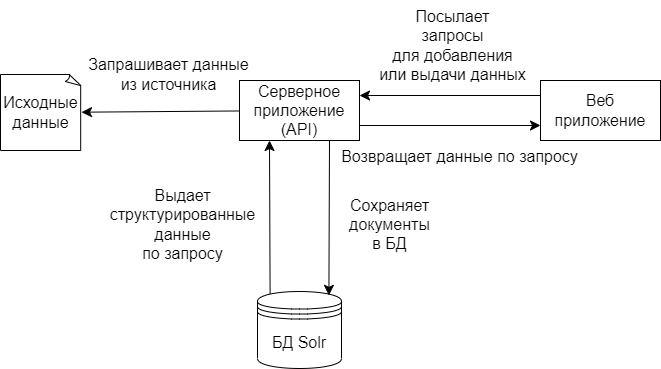


Рис. 1. Схема взаимодействия элементов приложения

Исходными файлами являются документы, описывающие фильмы по вышеописанному формату. Они хранятся в одном или нескольких файлах с расширением JSON.

Серверное приложение берет на себя функции посредника между исходными данными и бд Solr при создании индекса, и между бд Solr и клиентским приложением при осуществлении и обработки пользовательского запроса. Оно работает по принципам REST API, в упрощенном виде это значит, что:

* Запросы передаются по протоколу HTTP;
* Все данные запрашиваются через один URL-адрес;
* Разделяется логика клиентского приложения и сервера;
* Данные передаваемые данные сериализуются (например, в JSON).

Приложение будет реализовано с использованием технологии Java Spring и SolrJ. Исходный код представлен в приложении 1. Список и описание конечных точек приложения приведены в таблице 2. В таблице 3 описаны типы входящих и исходящих данных.

Таблица 2. Описание конечных точек приложения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Путь | Метод | Входные данные | Выходные данные | Описание |
| /film/{id} | GET | Id  (переменная пути) | ApiResponse | Получение записей по id |
| /film | POST | ApiRequest  (тело запроса) | ApiResponse | Получение записей по запросу |
| /film/add | POST | Query  (тело запроса) | нет | Добавление записи в бд |
| /genres | POST | Query  (тело запроса) | List<Map.Entry<String, Long>> | Получение списка жанров |
| /reindex | POST | нет | нет | Очищает базу данных, затем загружает документы из изначального источника |

Таблица 3. Описание типов данных API.

|  |  |  |
| --- | --- | --- |
| Название поля | Тип значения | Описание поля |
| Структура параметра ApiRequest | | |
| page | Integer | Страница выдачи |
| rows | Integer | Количество строк |
| query | Query | Описание запроса |
| Структура параметра Query | | |
| text | String | Идентификатор записи |
| genre | List<String> | Список жанров для фильтрации |
| Тип ответа ApiResponse | | |
| count | Integer | Количество найденных фильмов |
| films | List<Film> | Список фильмов для выдачи, в соответствии с параметрами пагинации |

Задачей клиентского приложения является отображение списка фильмов, фильтров, строк поиска, а также добавления фильма в базу данных. Разработка велась с использованием фрэймворка SvelteKit [4]. На рисунке 2 показана главная страница приложения. Слева приведен список жанров для фильтрации, справа от него строка полнотекстового поиска и поиска по идентификатору, ниже расположена таблица с описанием фильтров и пагинация.

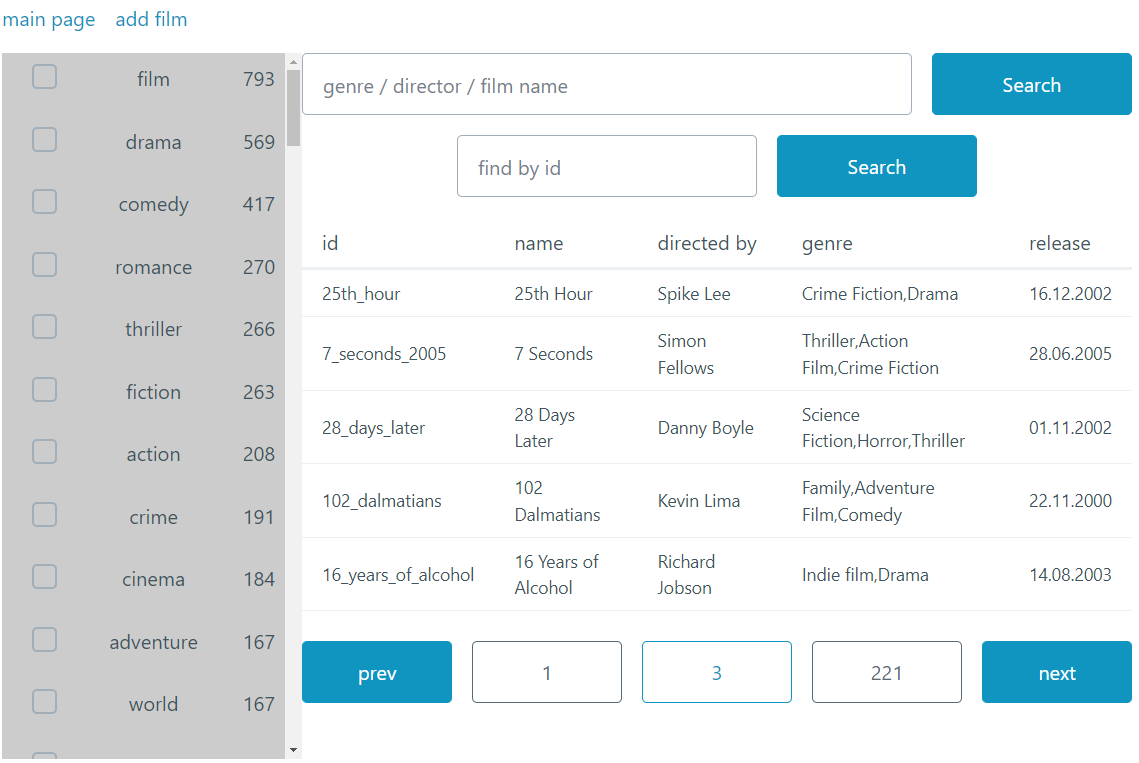


Рис. 2. Главная страница приложения

После нажатия на ссылку «add film» осуществляется переход на страницу с формой добавления фильма, ее вид приведен на рисунке 3. Все поля являются обязательными для ввода.

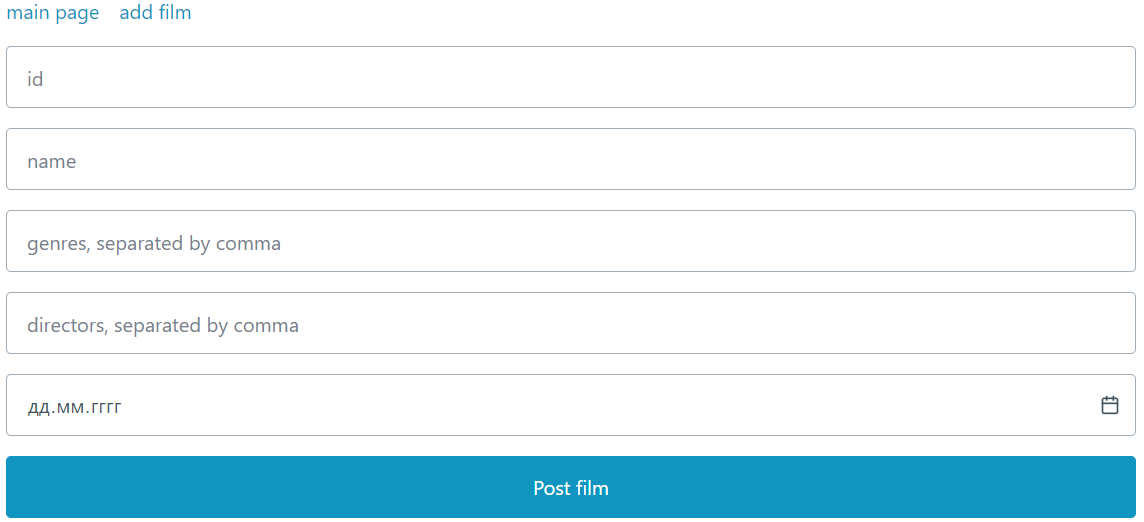


Рис. 3. Форма добавления фильма

# **Пример работы. Тестирование**

Развернем локально систему, описанную схемой (рис. 1). Для этого развернем локально Apache Solr, затем запустим серверное приложение, проиндексируем начальную выборку документов, запустим веб-приложение и протестируем его основной функционал. Для этого требуется скачать Solr с официального сайта, выполним команды:

1. Запустим Solr в режиме облака на двух узлах:
   1. solr start -cloud -p 8983 -s "D:\solr-9.1.1\example\cloud\node1\solr"
   2. solr start -cloud -p 7574 -s "D:\solr-9.1.1\example\cloud\node1\solr"
2. Создадим ядро Solr с названием «film»:
   1. solr create -c film
3. Отредактируем файл managed-shema.xml, добавив поля фильмов, описанных в таблице 1.

В качестве исходных данных будет использоваться выборка films [1] из официальных примеров Solr. Теперь запустим серверное приложение и произведем первоначальную индексацию, отправив запрос на конечную точку «/reindex». Результат работы показан на рисунке 4, на изображении показано, что в индексе хранится 1101 документ, с соответствующей структурой и содержанием.

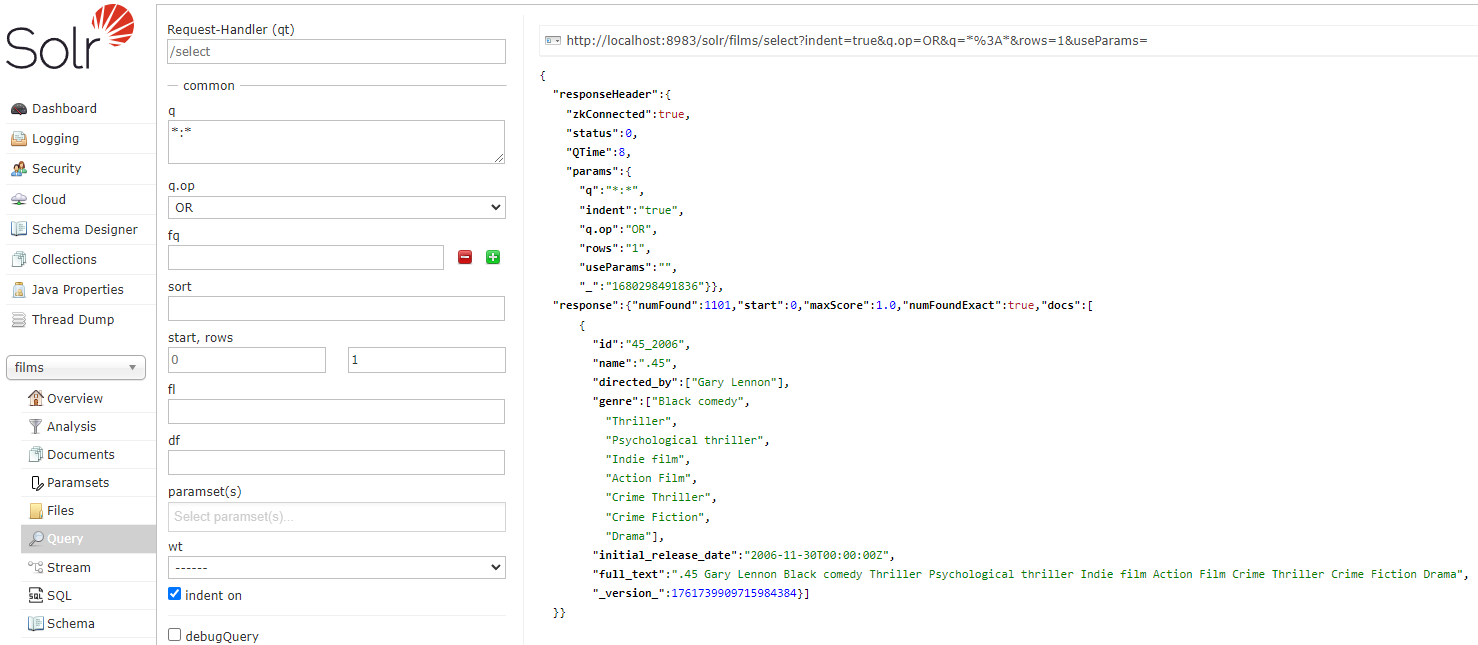


Рис. 4. Пример запроса к Solr через его собственный веб-интерфейс

Протестируем функционал клиентского приложения. Запросим, например, японскую комедию для этого в боковом фильтре выберем пункт «Comedy» введем в строку поиска «japanese». В результате мы получили 5 фильмов, соответствующих запросу (рис. 5). Нажав на кнопку с надписью «next», можно получить следующие 5 фильмов (рис. 6).

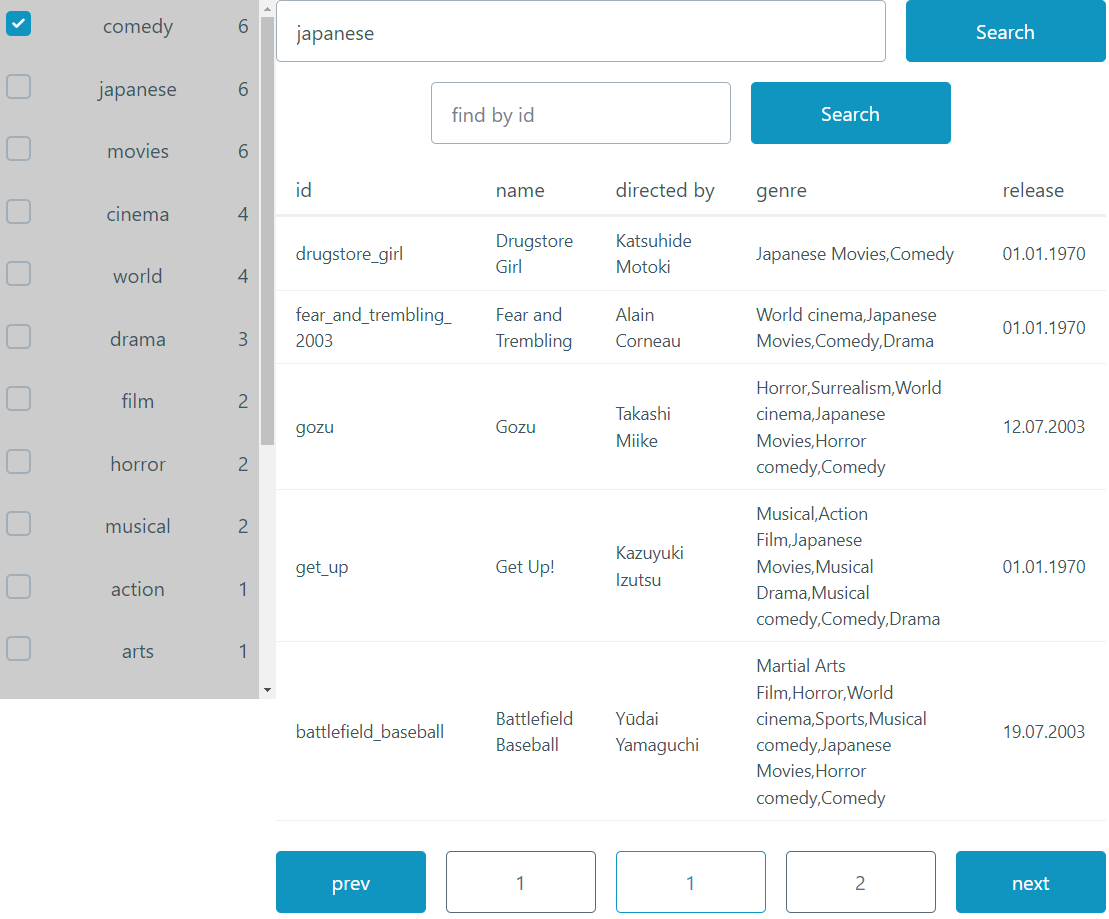


Рис. 5. Результат запроса японских комедий

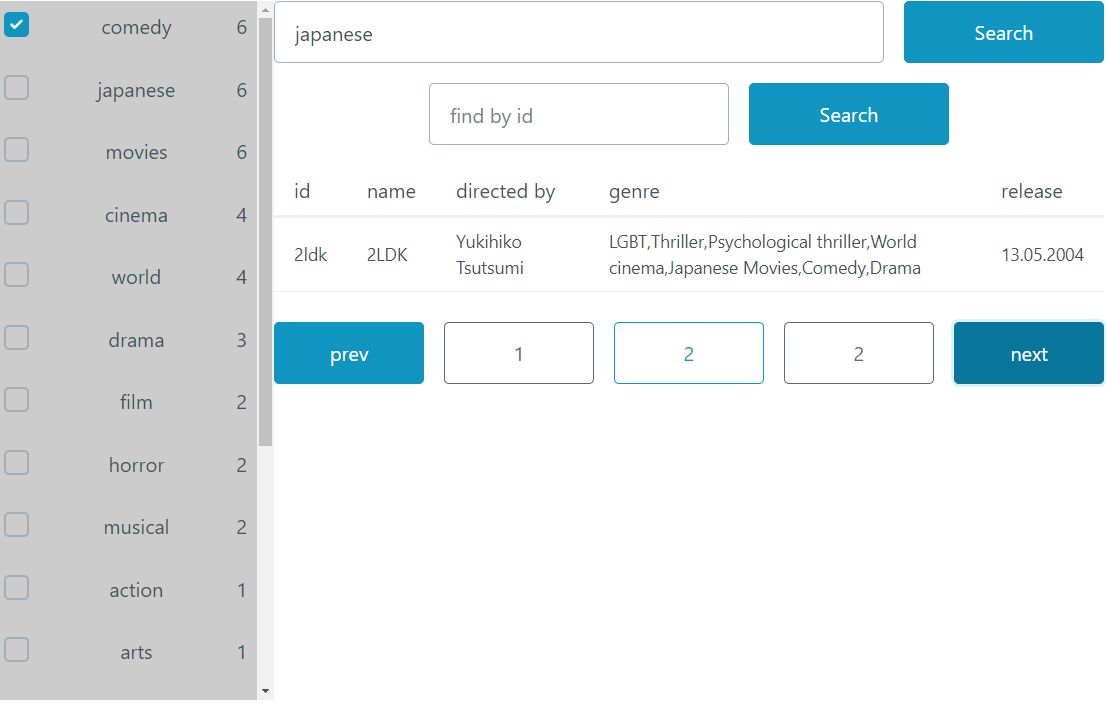


Рис. 6. Вторая страница результата запроса японских комедий

Запросим фильм «2LDK» по id. Для этого заполним соответствующее поле и выполним запрос.

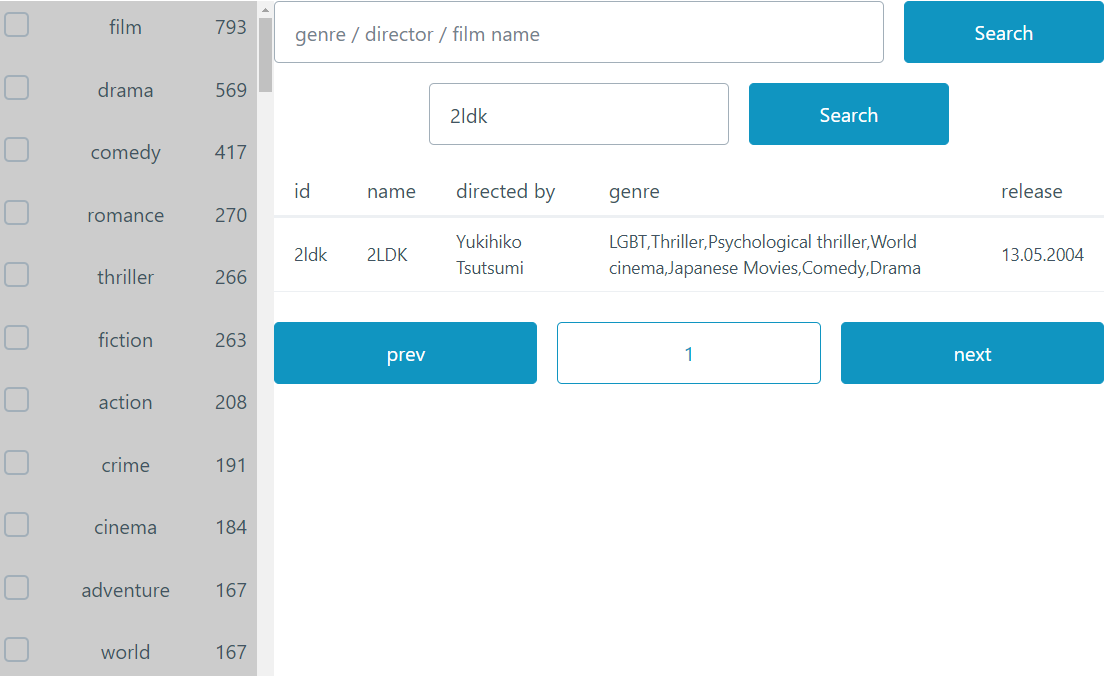


Рис. 7. Результат поиска фильма по id «2LDK»

Добавим новый фильм, воспользовавшись формой (рис. 8). Вернемся на главную страницу и запросим его по id.



Рис. 8. Заполненная форма добавления фильма

Запросим добавленный фильм по его идентификатору «qwe». На рисунке 9 показан результат запроса.

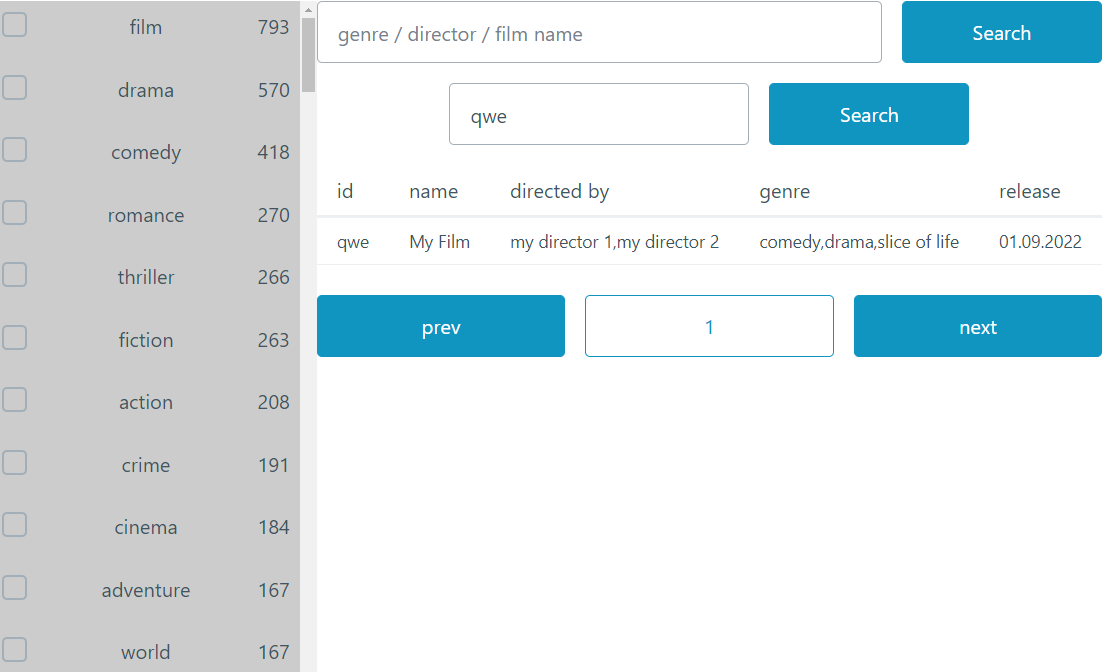


Рис. 9. Результат выдачи добавленного фильма по id

Попробуем найти добавленный фильм по имени режиссёра. На рисунке 10 показан результат запроса для режиссёра «my director 1».

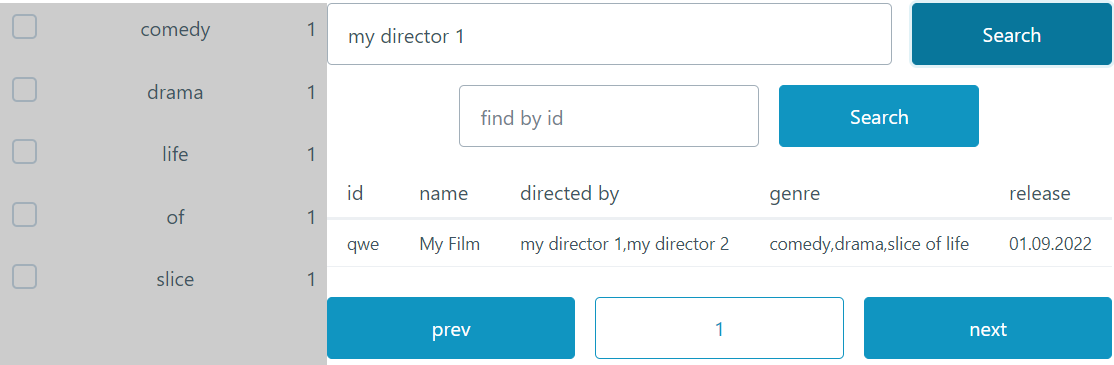


Рис. 10. Результат выдачи добавленного фильма по имени режиссёра

Исходя из выше полученных результатов можно сделать вывод о том, что приложение функционирует корректно, выполняет свои основные функции.

# **Вывод**

В результате курсовой работы была разработана база данных фильмов с использованием не реляционной СУБД Apache Solr. Создано серверное и клиентское приложение для работы с созданной базой данных. На основе проведенного тестирования можно сделать вывод о том, что приложение работает корректно, выполняет свои функции: перемещение по записям, добавление новых записей, поиск записей по отдельным полям, задание фильтра по отдельным полям.

# **Список использованных источников**

1. Пример выборки фильмов в официальном гайде Apache Solr [Электронный ресурс]. URL: https://solr.apache.org/guide/solr/latest/gettin g-started/tutorial-films.html
2. Официальная документация Apache [Электронный ресурс]. URL: https://solr.apache.org/guide/solr/latest/index.html
3. Официальная документация Java Spring [Электронный ресурс]. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/
4. Официальная документация Svelte и SvelteKit [Электронный ресурс]. URL: https://svelte.dev/docs

# **Приложение**

**Приложение 1. Исходный код серверного приложения (API)**

**ApiRequest**

package com.example.solrfilmsdemo.model;

public record ApiRequest(Integer page, Integer rows, Query query) implements Pageable {}

Pageable

package com.example.solrfilmsdemo.model;

public interface Pageable {

Integer page();

Integer rows();

default Integer getOffset() {

return (page() - 1) \* rows();

}

}

**ApiResponse**

package com.example.solrfilmsdemo.model;

import java.util.List;

public record ApiResponse(Integer count, List<Film> films) {}

Film

package com.example.solrfilmsdemo.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import org.apache.solr.client.solrj.beans.Field;

import java.util.Date;

import java.util.List;

@NoArgsConstructor

@Getter

@Setter

public class Film {

@Field

private String id;

public void setId(String id) {

this.id = id.replaceFirst("/.\*/", "");

}

@Field

private String name;

@JsonProperty("directed\_by")

@Field("directed\_by")

private List<String> directedBy;

@Field

private List<String> genre;

@JsonProperty("initial\_release\_date")

@Field("initial\_release\_date")

private Date initialReleaseDate;

@JsonIgnore

@Field("full\_text")

private String fullText;

public void calcFullText() {

this.fullText = String.format("%s %s %s", name, listToFullTextPart(directedBy), listToFullTextPart(genre));

}

private String listToFullTextPart(List<String> l) {

return l == null || l.isEmpty() ? "" : String.join(" ", l);

}

}

**Query**

package com.example.solrfilmsdemo.model;

import org.apache.logging.log4j.util.Strings;

import java.util.ArrayList;

import java.util.List;

public record Query(String text, List<String> genre) {

private List<String> joinQueryParts(String fieldName, String src) {

return joinQueryParts(fieldName, List.of(src.split(" ")));

}

private List<String> joinQueryParts(String fieldName, List<String> src) {

return src.stream().map(s -> String.format(fieldName + ":%s", s)).toList();

}

public String toSolrQuery() {

var queries = new ArrayList<String>();

if (!Strings.isEmpty(text)) {

queries.addAll(joinQueryParts("full\_text", text));

} else {

queries.add("full\_text:\*");

}

if (genre != null) {

queries.addAll(joinQueryParts("genre", genre));

}

return String.join(" && ", queries);

}

}

**ApiCOntrollers**

package com.example.solrfilmsdemo;

import com.example.solrfilmsdemo.model.ApiRequest;

import com.example.solrfilmsdemo.model.ApiResponse;

import com.example.solrfilmsdemo.model.Film;

import com.example.solrfilmsdemo.model.Query;

import lombok.RequiredArgsConstructor;

import org.apache.solr.client.solrj.SolrServerException;

import org.springframework.web.bind.annotation.\*;

import java.io.IOException;

import java.util.List;

import java.util.Map;

@RequiredArgsConstructor

@RestController("api/")

public class ApiControllers {

private final SolrService solrService;

@PostMapping("reindex/")

void reindexFilms() {

solrService.reindex();

}

@GetMapping("film/{id}")

ApiResponse getFilmsById(@PathVariable String id) {

return solrService.getFilmsById(id);

}

@PostMapping("film")

ApiResponse getFilms(@RequestBody ApiRequest request) {

return solrService.getFilms(request);

}

@PostMapping("genres")

List<Map.Entry<String, Long>> getGenres(@RequestBody Query request) {

return solrService.getGenres(request);

}

@PostMapping("film/add")

void getGenres(@RequestBody Film film) throws SolrServerException, IOException {

solrService.addFilm(film);

}

}

**SolrConfig**

package com.example.solrfilmsdemo;

import org.apache.solr.client.solrj.SolrClient;

import org.apache.solr.client.solrj.impl.CloudSolrClient;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import java.util.List;

@Configuration

public class SolrConfig {

@Bean

SolrClient solrClient(

@Value("${solr.hosts}") List<String> hosts,

@Value("${solr.collection.default}") String collection) {

var client = new CloudSolrClient.Builder(hosts).build();

client.setDefaultCollection(collection);

return client;

}

}

**SolrService**

package com.example.solrfilmsdemo;

import com.example.solrfilmsdemo.model.ApiRequest;

import com.example.solrfilmsdemo.model.ApiResponse;

import com.example.solrfilmsdemo.model.Film;

import com.example.solrfilmsdemo.model.Query;

import com.fasterxml.jackson.core.type.TypeReference;

import com.fasterxml.jackson.databind.ObjectMapper;

import lombok.RequiredArgsConstructor;

import org.apache.solr.client.solrj.SolrClient;

import org.apache.solr.client.solrj.SolrQuery;

import org.apache.solr.client.solrj.SolrServerException;

import org.springframework.core.io.ResourceLoader;

import org.springframework.stereotype.Service;

import java.io.IOException;

import java.util.List;

import java.util.Map;

@RequiredArgsConstructor

@Service

public class SolrService {

private final SolrClient solrClient;

private final ObjectMapper objectMapper;

private final ResourceLoader resourceLoader;

public void reindex() {

try {

solrClient.deleteByQuery("\*:\*");

var resource = resourceLoader.getResource("classpath:films.json");

var films = objectMapper.readValue(resource.getFile(), new TypeReference<List<Film>>() {

});

films.parallelStream().forEach(Film::calcFullText);

solrClient.addBeans(films);

} catch (SolrServerException | IOException e) {

throw new RuntimeException(e);

}

}

public ApiResponse getFilmsById(String id) {

var query = new SolrQuery("id:" + id);

try {

var response = solrClient.query(query);

return new ApiResponse((int) response.getResults().getNumFound(), response.getBeans(Film.class));

} catch (SolrServerException | IOException e) {

throw new RuntimeException(e);

}

}

public ApiResponse getFilms(ApiRequest request) {

var query = new SolrQuery(request.query().toSolrQuery());

query.setStart(request.getOffset());

query.setRows(request.rows());

try {

var response = solrClient.query(query);

return new ApiResponse((int) response.getResults().getNumFound(), response.getBeans(Film.class));

} catch (SolrServerException | IOException e) {

throw new RuntimeException(e);

}

}

public List<Map.Entry<String, Long>> getGenres(Query request) {

var query = new SolrQuery(request.toSolrQuery());

query.setRows(0);

query.setFacet(true);

query.addFacetField("genre");

query.setFacetMinCount(1);

try {

var response = solrClient.query(query);

return response.getFacetFields()

.get(0)

.getValues()

.stream()

.map(ff -> Map.entry(ff.getName(), ff.getCount()))

.toList();

} catch (SolrServerException | IOException e) {

throw new RuntimeException(e);

}

}

public void addFilm(Film film) throws SolrServerException, IOException {

film.calcFullText();

solrClient.addBean(film);

}

}

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.9</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>demo</name>

<description>solr-films-demo</description>

<properties>

<java.version>19</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<dependency>

<groupId>org.apache.solr</groupId>

<artifactId>solr-solrj</artifactId>

<version>9.1.1</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Application**

package com.example.solrfilmsdemo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

**Приложение 2. Исходный код веб-приложения**

**Categories.svelte**

<script lang="ts">  
 import type { GenreResponse } from '../types';  
 import { createEventDispatcher } from 'svelte';  
  
 export let genres: GenreResponse;  
 export let selectedGenres: GenreResponse;  
  
 $: genresKV = genres.map(recordToKV);  
  
 const dispatch = createEventDispatcher<{  
 select: { genre: string; selected: boolean };  
 }>();  
  
 function recordToKV(r: Record<string, number>) {  
 const key = Object.keys(r)[0];  
 return { key, value: r[key] };  
 }  
  
 function onGenreSelected(e: Event) {  
 const el = e.target as HTMLInputElement;  
 dispatch('select', { genre: el.value, selected: el.checked });  
 }  
</script>  
  
<ul>  
 {#each genresKV as g}  
 <li>  
 <label class="flex align-center justify-space-between hover-cursor"  
 ><input  
 type="checkbox"  
 on:change={onGenreSelected}  
 value={g.key}  
 checked={selectedGenres.includes(g.key)}  
 /><span>{g.key}</span><span>{g.value}</span></label  
 >  
 </li>  
 {/each}  
</ul>

**FilmsTable.svelte**

<script lang="ts">  
 import type { ApiResponse } from '../types';  
 import { useFormatter } from '../utils/useFormatter';  
  
 const { formatDate } = useFormatter();  
 export let films: ApiResponse['films'];  
</script>  
  
<table>  
 <thead>  
 <tr>  
 <th scope="col">id</th>  
 <th scope="col">name</th>  
 <th scope="col" nowrap>directed by</th>  
 <th scope="col">genre</th>  
 <th scope="col">release</th>  
 </tr>  
 </thead>  
 <tbody>  
 {#each films as f (f.id)}  
 <tr>  
 <td style="max-width: 200px; word-wrap: break-word">{f.id}</td>  
 <td>{f.name}</td>  
 <td>{f.directed\_by}</td>  
 <td>{f.genre}</td>  
 <td>{formatDate(f.initial\_release\_date)}</td>  
 </tr>  
 {/each}  
 </tbody>  
</table>

**Pagination.svelte**

<script lang="ts">

import { createEventDispatcher } from 'svelte';

export let amount: number;

export let currentPage: number;

export let rows: number;

$: pagesCount = Math.ceil(amount / rows);

const dispatch = createEventDispatcher<{ pageChange: number }>();

function nextPage() {

if (currentPage < pagesCount) {

dispatch('pageChange', currentPage + 1);

}

}

function prevPage() {

if (currentPage > 1) {

dispatch('pageChange', currentPage - 1);

}

}

</script>

<div class="grid">

<button on:click={prevPage}>prev</button>

{#if pagesCount > 1}

<button class="outline secondary">1</button>

<button class="outline">{currentPage}</button>

<button class="outline secondary">{pagesCount}</button>

{:else}

<button class="outline">{currentPage}</button>

{/if}

<button on:click={nextPage}>next</button>

</div>

**routes/+layout.svelte**

<script lang="ts">

import '@picocss/pico';

import '../app.css';

</script>

<header class="container">

<h1>Simple films aggregator</h1>

<nav>

<ul>

<li><a href="/">main page</a></li>

<li><a href="/add">add film</a></li>

</ul>

</nav>

</header>

<slot />

**routes/+page.svelte**

<script lang="ts">

import { useApi } from '../utils/useApi';

import type { ApiResponse, Query, GenreResponse } from '../types';

import { onMount } from 'svelte';

import FilmsTable from '../components/FilmsTable.svelte';

import Categories from '../components/Categories.svelte';

import Pagination from '../components/Pagination.svelte';

const { getFilmsByQuery, getFilmsById, getGenres } = useApi();

let apiResponse: ApiResponse = { count: 0, films: [] };

let genres: GenreResponse = [];

onMount(async () => {

genres = await getGenres(query.query);

apiResponse = await getFilmsByQuery(query);

});

const query: Query = { page: 1, rows: 5, query: { text: '', genre: [] } };

let id = '';

function setQueryText(e: InputEvent) {

query.query.text = (e.target as HTMLInputElement).value;

}

function setId(e: InputEvent) {

id = (e.target as HTMLInputElement).value;

}

async function setPage(e: CustomEvent<number>) {

query.page = e.detail;

apiResponse = await getFilmsByQuery(query);

}

function manageGenres(e: CustomEvent<{ genre: string; selected: boolean }>) {

const { genre, selected } = e.detail;

if (selected) {

query.query.genre.push(genre);

} else {

query.query.genre = query.query.genre.filter((g) => g != genre);

}

}

async function fetchFilmsByQuery() {

query.page = 1;

await Promise.all<ApiResponse | GenreResponse>([

getFilmsByQuery(query),

getGenres(query.query)

]).then((r) => {

apiResponse = r[0] as ApiResponse;

genres = r[1] as GenreResponse;

});

}

async function fetchFilmsById() {

await getFilmsById(id).then((r) => (apiResponse = r));

}

</script>

<div class="container flex">

<aside class="aside flex-grow-1">

<Categories {genres} selectedGenres={query.query.genre} on:select={manageGenres} />

</aside>

<main class="flex-grow-0">

<form class="flex gap-20 margin-0" on:submit|preventDefault={fetchFilmsByQuery}>

<input on:change={setQueryText} type="text" placeholder="genre / director / film name" />

<button style="max-width: 200px" type="submit">Search</button>

</form>

<form class="flex gap-20 justify-center margin-0" on:submit|preventDefault={fetchFilmsById}>

<input style="max-width: 300px" on:change={setId} type="text" placeholder="find by id" />

<button style="max-width: 200px" type="submit">Search</button>

</form>

<div>

{#if !apiResponse}

<progress />

{:else}

<FilmsTable films={apiResponse.films} />

{/if}

</div>

<Pagination

amount={apiResponse.count}

currentPage={query.page}

rows={query.rows}

on:pageChange={setPage}

/>

</main>

</div>

<style>

.aside {

height: 70vh;

width: 300px;

overflow: auto;

background-color: rgba(0, 0, 0, 0.2);

}

</style>

**routes/add/+page.svelte**

<script lang="ts">

import type { Film } from '../../types';

import { useApi } from '../../utils/useApi';

const { addFilm } = useApi();

const film: Film = {

id: '',

name: '',

directed\_by: [],

genre: [],

initial\_release\_date: ''

};

async function postFilm(e: SubmitEvent) {

const form = e.target as HTMLFormElement;

for (const f of Object.keys(film) as keyof Film) {

const v = (form.elements.namedItem(f) as HTMLInputElement).value;

if (film[f] instanceof Array) {

film[f] = v

.split(',')

.map((s) => s.trim())

.filter(Boolean);

} else {

film[f] = v;

}

}

await addFilm(film);

}

</script>

<form on:submit|preventDefault={postFilm} class="container">

<input type="text" name="id" placeholder="id" required />

<input type="text" name="name" placeholder="name" required />

<input type="text" name="genre" placeholder="genres, separated by comma" required />

<input type="text" name="directed\_by" placeholder="directors, separated by comma" required />

<input type="date" name="initial\_release\_date" placeholder="release date" required />

<button type="submit">Post film</button>

</form>

**useApi.ts**

import type { ApiResponse, Film, GenreResponse, Query } from '../types';

export const useApi = function () {

async function getFilmsById(id: string) {

return fetch(`/api/film/${id}`)

.then((r) => r.json())

.then((r: ApiResponse) => r)

.catch((r) => {

console.log(r);

return null;

});

}

async function getFilmsByQuery(q: Query) {

return fetch(`/api/film`, {

method: 'post',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(q)

})

.then((r) => r.json())

.then((r: ApiResponse) => r)

.catch((r) => {

console.log(r);

return null;

});

}

async function getGenres(q: Query['query']) {

return fetch('/api/genres', {

method: 'post',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(q)

})

.then((r) => r.json())

.then((r: GenreResponse) => r)

.catch((r) => {

console.log(r);

return null;

});

}

async function addFilm(f: Film) {

return fetch('/api/film/add', {

method: 'post',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(f)

})

.then((r) => {

if (!r.ok) throw new Error(r.statusText);

})

.then(() => alert('Film added'))

.catch((e) => alert(e));

}

return { getFilmsById, getFilmsByQuery, getGenres, addFilm };

};

**useFormatter.ts**

export const useFormatter = function () {

const dateTimeFormatter = new Intl.DateTimeFormat('ru-RU');

function formatDate(src: string) {

const date = new Date(src);

if (!isNaN(date.getTime())) {

return dateTimeFormatter.format(date);

}

return src;

}

return { formatDate };

};

**app.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<link rel="icon" href="%sveltekit.assets%/favicon.png" />

<meta name="viewport" content="width=device-width" />

<title>Solr demo - web app</title>

%sveltekit.head%

</head>

<body data-sveltekit-preload-data="hover">

<div style="display: contents">%sveltekit.body%</div>

</body>

</html>

**types.ts**

export interface Film {

id: string;

name: string;

directed\_by: Array<string>;

genre: Array<string>;

initial\_release\_date: string;

}

export interface ApiResponse {

count: number;

films: Array<Film>;

}

export interface Query {

page: number;

rows: number;

query: {

text: string;

genre: Array<string>;

};

}

export type GenreResponse = Array<Record<string, number>>

**package.json**

{

"name": "solr-web",

"version": "0.0.1",

"private": true,

"scripts": {

"dev": "vite dev",

"build": "vite build",

"preview": "vite preview",

"check": "svelte-kit sync && svelte-check --tsconfig ./tsconfig.json",

"check:watch": "svelte-kit sync && svelte-check --tsconfig ./tsconfig.json --watch",

"lint": "prettier --plugin-search-dir . --check . && eslint .",

"format": "prettier --plugin-search-dir . --write ."

},

"devDependencies": {

"@sveltejs/adapter-auto": "^2.0.0",

"@sveltejs/kit": "^1.5.0",

"@typescript-eslint/eslint-plugin": "^5.45.0",

"@typescript-eslint/parser": "^5.45.0",

"eslint": "^8.28.0",

"eslint-config-prettier": "^8.5.0",

"eslint-plugin-svelte3": "^4.0.0",

"prettier": "^2.8.0",

"prettier-plugin-svelte": "^2.8.1",

"svelte": "^3.54.0",

"svelte-check": "^3.0.1",

"tslib": "^2.4.1",

"typescript": "^5.0.0",

"vite": "^4.2.0"

},

"type": "module",

"dependencies": {

"@picocss/pico": "^1.5.7"

}

}