Предположим, перед нами стоит задача разработать веб-приложение, в котором пользователи будут сохранять и использовать для некоторых целей собственную коллекцию фильмов. Для реализации данной цели необходимо реализовать полноценное full-stack приложение. Для написания back-end будем использовать spring boot. Для написания front-end воспользуемся react. Базу данных будем хранить в MySQL.

Первым делом займемся инициализацией back-end части (сервер в дальнейшем).

Для это воспользуемся полноценной версией приложения Intelij IDEA.

Выберем во вкладке создания нового проекта приложение Spring Initializer, выберем путь проекта, версию языка Java, а так же введем название проекта как указано на рисунке 1, после чего кликаем на “далее”.

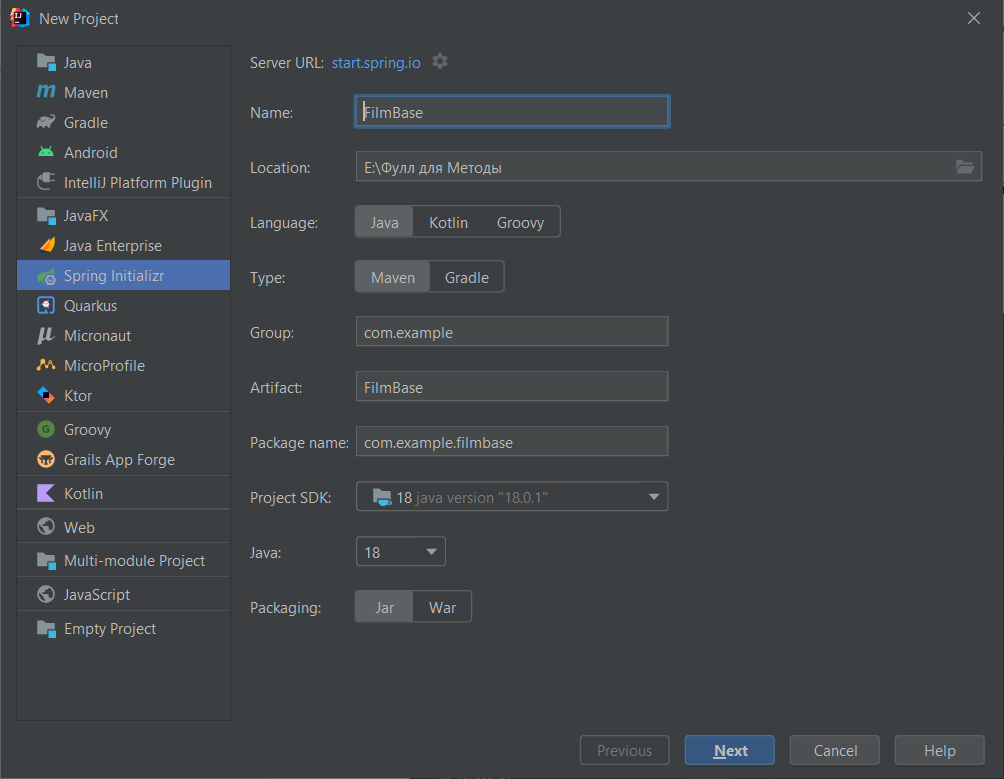


Рис. 1

В появившемся окне выбираем зависимости, которые будут включены в проект при его инициализации (рис. 2), после чего кликаем на “Finish” и завершаем инициализации сервера.

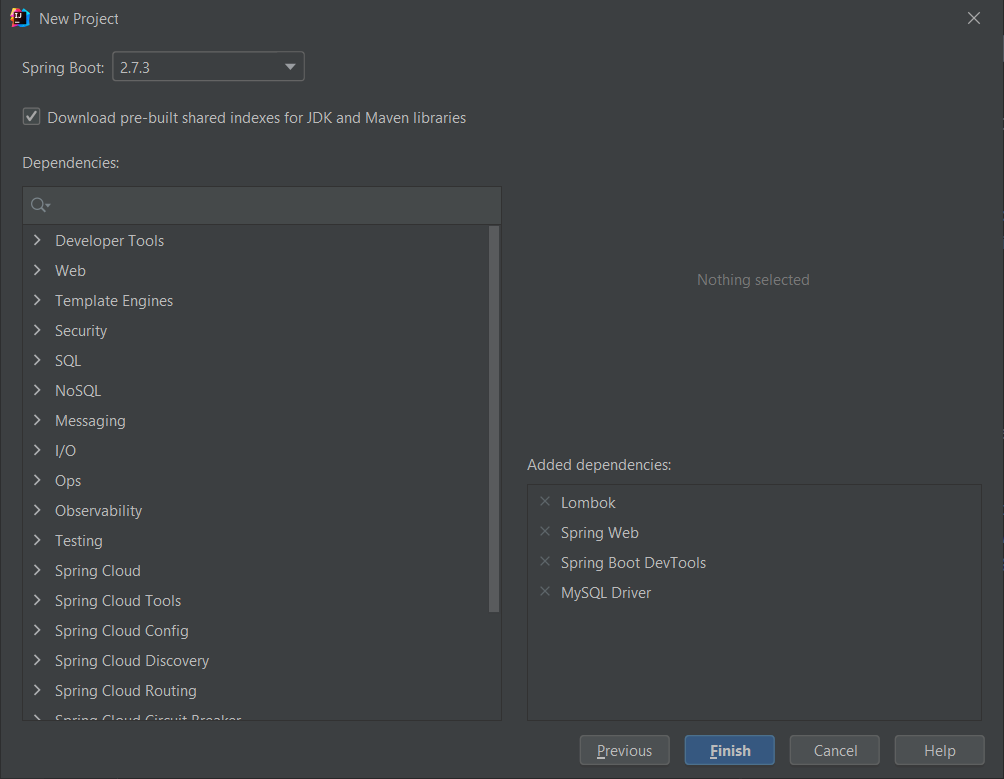


Рис. 2

После генерации и открытия проекта в среде разработки обязательно пометьте исполняющую директорию для корректного отображения и запуска кода, как показано на рисунке 3.

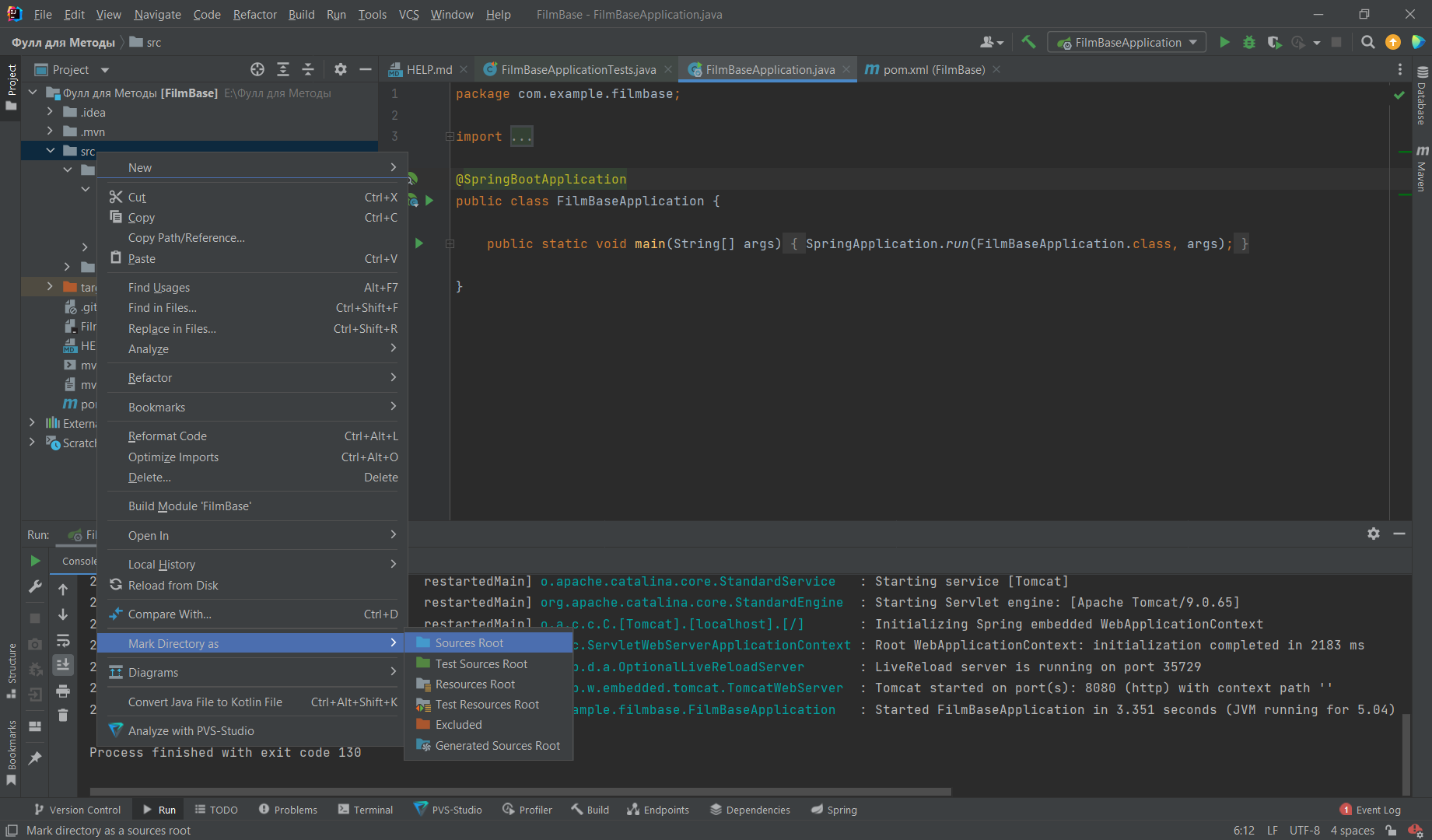


Рисунок 3

Теперь приступим к созданию базы данных, в которой будут храниться пользователи, книги, а также связи между ними. Для базы данных будем использовать MySQL, визуальным помощников в работе с которой будем использовать MySQL WorkBench

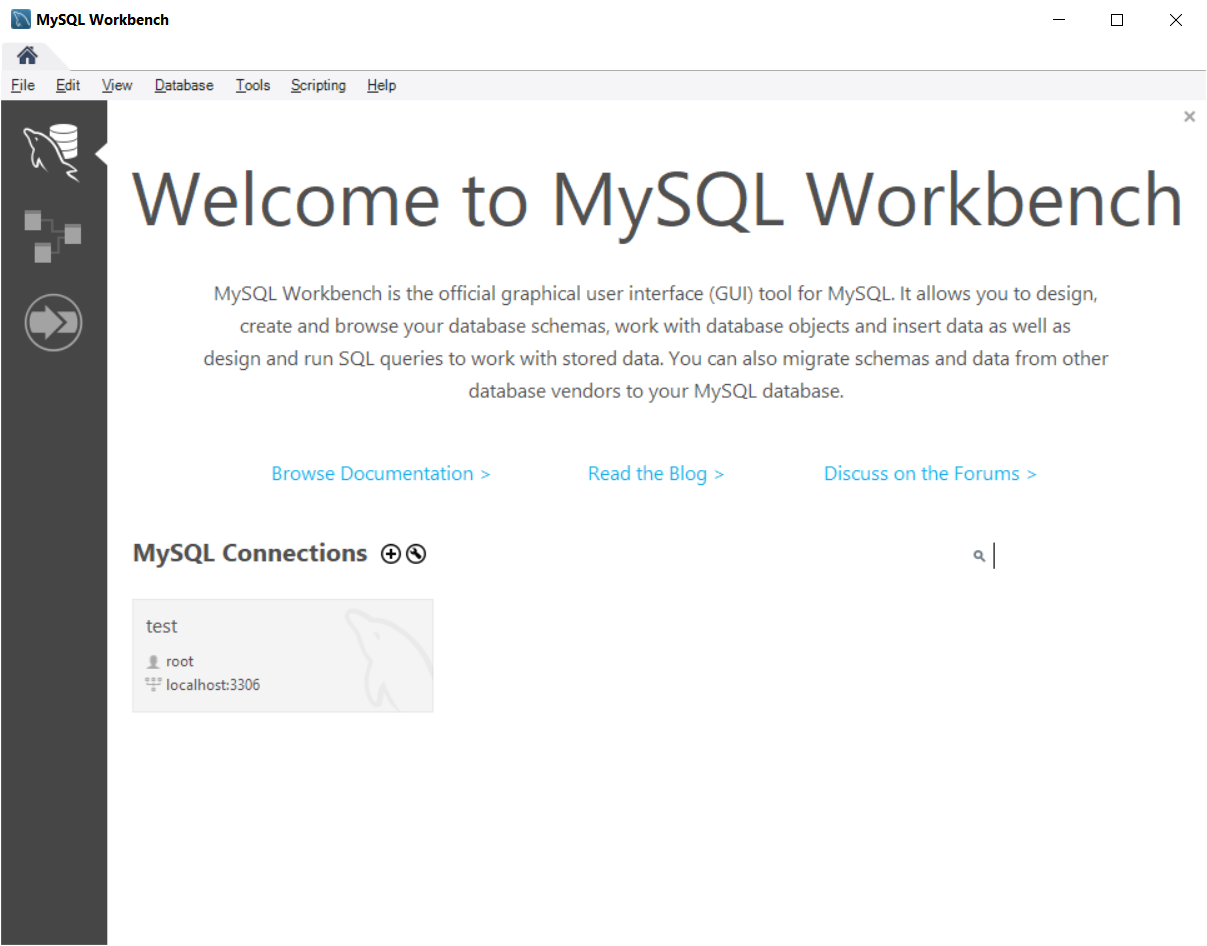


Рисунок 4.

Создадим новую базу данных для нашего приложения. Для этого кликнем на значок плюса рядом с надписью MySQL Connection (Рисунок 5)

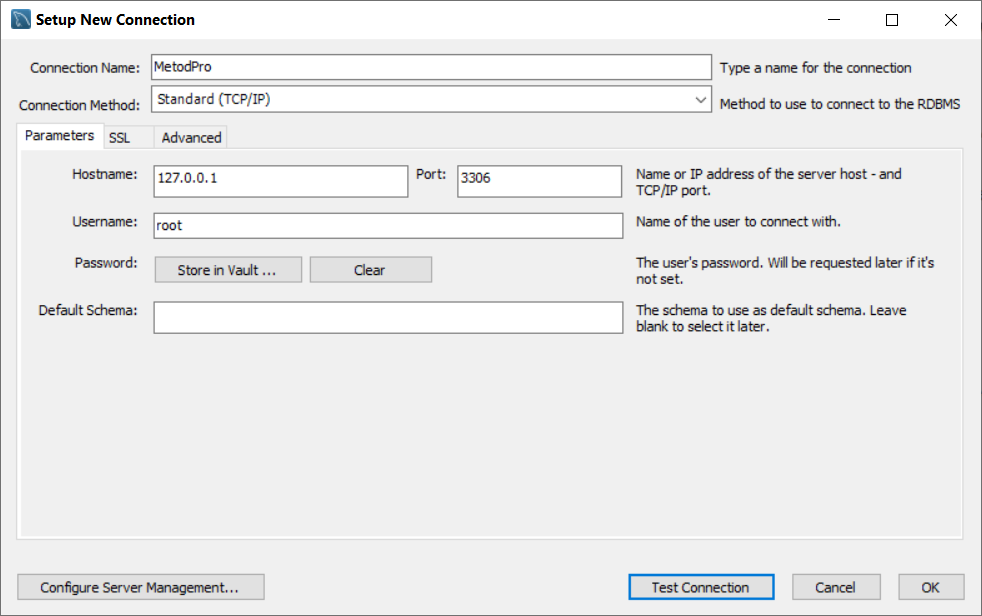


Рисунок 5.

Введем название базы, все остальное оставляем по умолчанию. После чего нажимаем “Test Connection”. Программа потребует ввести админовский пароль, в стандарте он “admin”. Если все ввели верно – появиться окно с сообщением о созданной базе и стабильном подключении к ней (рис. 6).

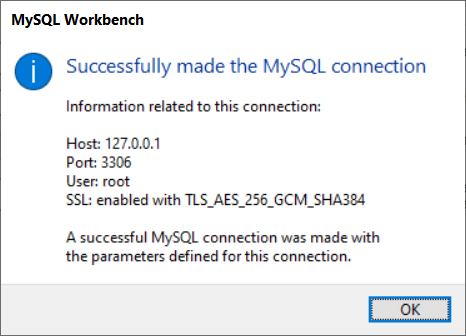


Рисунок 6.

После успешного создания откроем нашу базу данных двойным кликов левой кнопкой мыши по ней. В появившемся окне в панели “Schemas” кликаем по пустой области и выбираем “create new schema”. Вводим ее название и и нажимаем “Apply” (рис. 7).

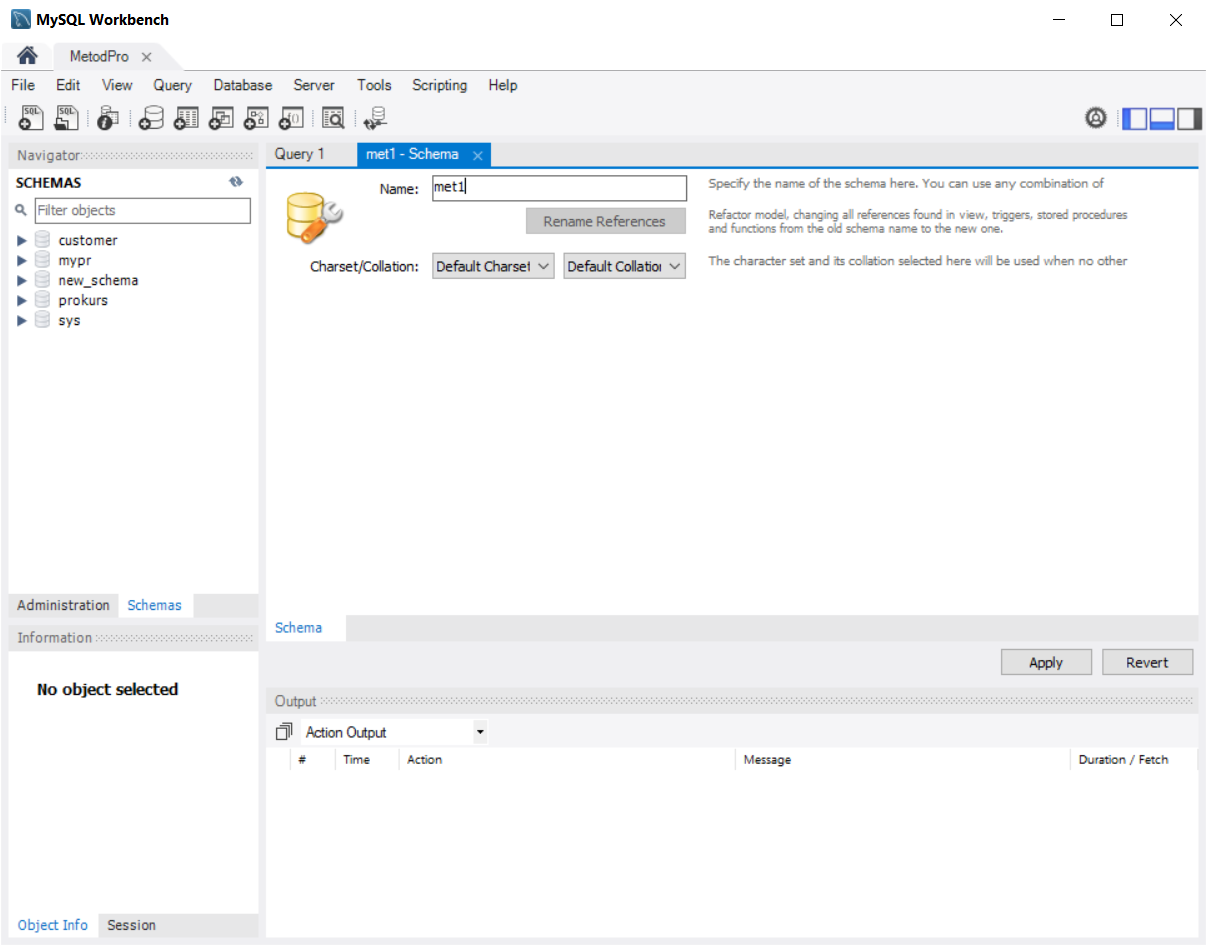


Рисунок 7.

После правильного создания базы, заходим в нее, и в разделе “tables” кликаем правой клавишей и создаем нужные для нас в проекте таблицы.

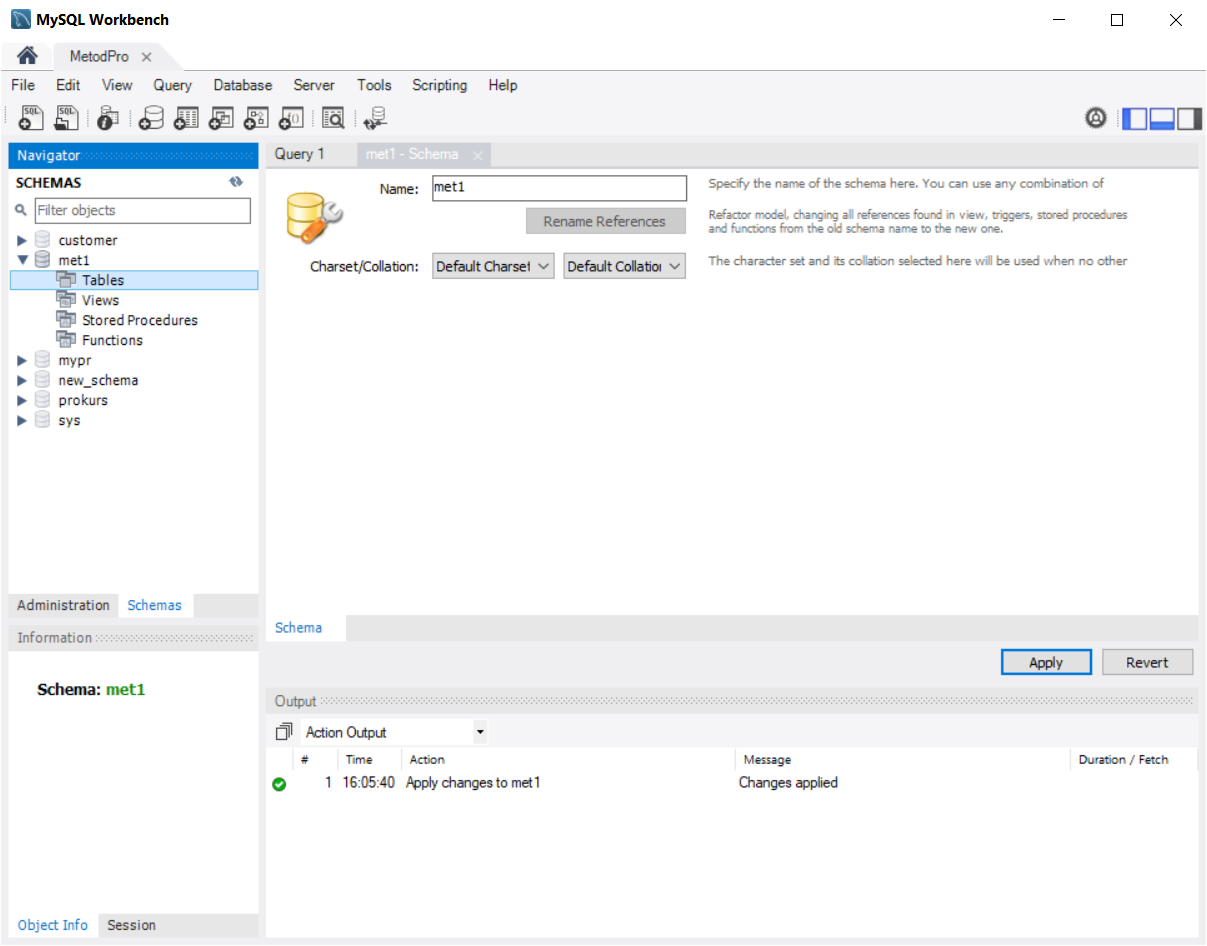


Рисунок 8.

Есть также альтернативный вариант, который позволяет не создавать таблицы и все поля этих таблиц явно, без участия WorkBench. Создавать таблицы будем в Intellij Idea, а если более детально, то просто пропишем все поля и таблицы, после чего соберем и запустим проект, тем самым, благодаря драйверу MySQL, который мы подключали в зависимости при инициализации проекта, будут созданы все прописанные нами таблицы, поля, а также зависимости между полями и таблицами.

Для начала в корне проекта создадим папки для более удобной работы с проектом: Сущности, Репозитории, Контроллеры (рис. 9).

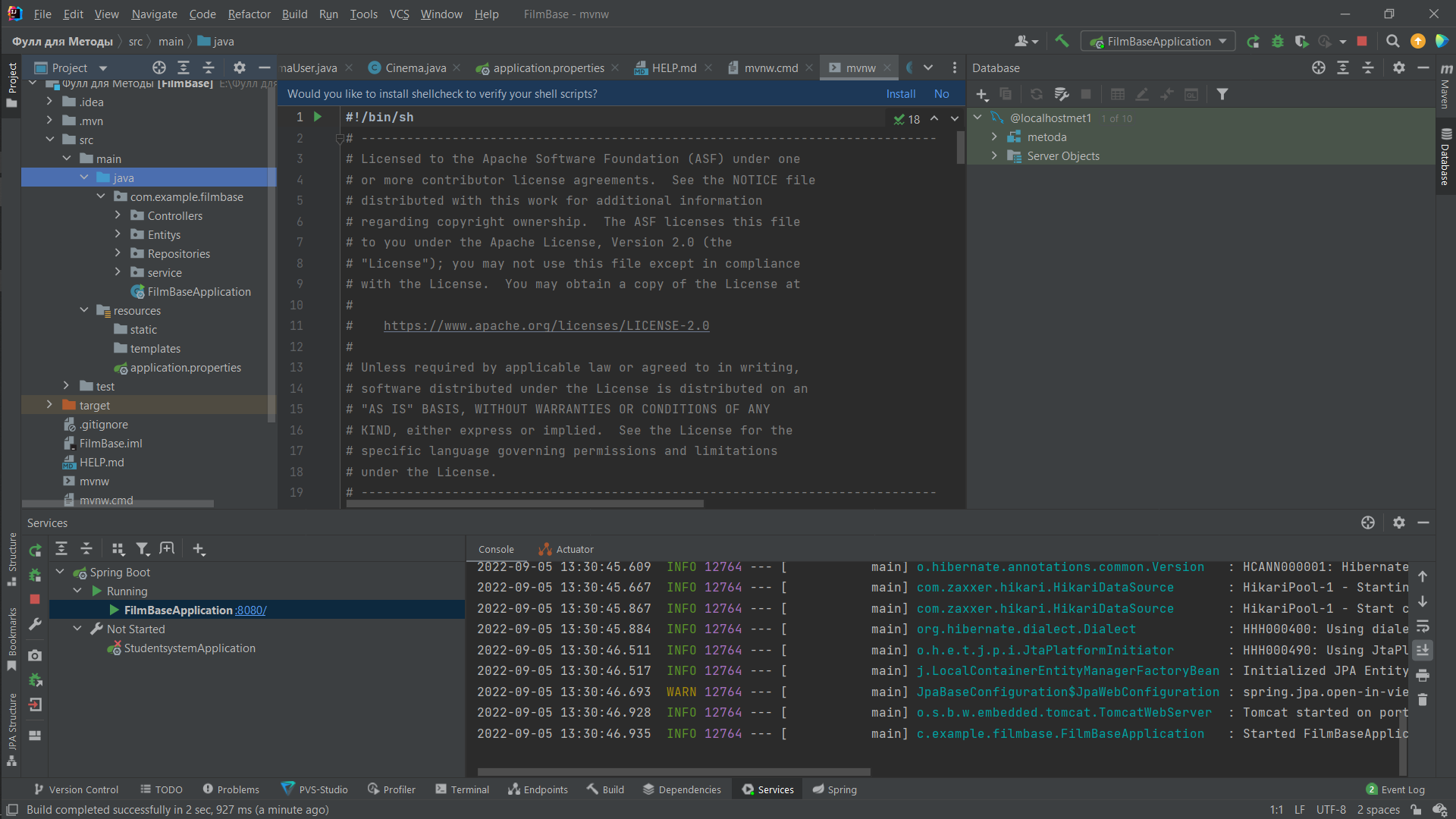


Рисунок 9

Важно! Созданные выше папки должны находиться в папке с запускаемым классом проекта! В ином случае, база данных попросту не будет инициализироваться, и ее придется прописывать самостоятельно через workbench

Настройки подключения к базе прописаны в файле, в папке resources->application.properies:

#configuration  
spring.jpa.hibernate.ddl-auto=update  
spring.datasource.url=jdbc:mysql://localhost:3306/met1  
spring.datasource.username=root  
spring.datasource.password=admin  
  
spring.jackson.serialization.*FAIL\_ON\_EMPTY\_BEANS*=false  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

Ddl-auto – важный элемент, в данном случае его параметр равен – update, что означает обновление базы в соответствии с той версией которая прописана в сущностях кода. Следующие три строки понятны: адрес базы данных, логин и пароль. Следующая строка отключает серриализацию между сущностями. В частности, предотвращает процесс зацикливания. И последняя строка нужна только в старых версиях спринга. Оставляем ее на всякий случай, она подключает драйвер для работы с базой данных без участия в этом зависимостей, прописанных в файле pom.xml. Все зависимости этого файла представлены ниже:

<?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>2.7.3</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.arjuncodes</groupId>  
 <artifactId>studentsystem</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>studentsystem</name>  
 <description>Full stack application using Spring Boot and react</description>  
 <properties>  
 <java.version>11</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </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>

В папке Entitys (сущности) мы и пропишем необходимые для проекта таблицы и поля. Для этого создается обычный класс, в название которого мы и пропишем название будущей таблицы. После чего, перед объявлением класса пропишем аннотацию, которая позволит спрингу понять, что это сущность базы данных (@Enyity).

Далее создадим первое integer поле, которое выступит в качестве id таблицы и будет иметь автоматическую генерацию.

После чего пропишем, наподобие описанного выше поля, все остальные, необходимые в нашей таблице поля, создадим конструктор без параметров, конструктор с параметрами, а также геттеры и сеттеры для всех полей (делается это при помощи аннотаций от Lombook). Также создаем сразу же нужные нам вторичные ключи для работы. На рисунке 10 представлена таблица пользователей, которая имеет сразу три внешних ключа. Все отношения от это таблицы имеют отношение один ко многим (@OneToMany). После написания данной аннотации в скобках указываем, к какому полю связанной таблицы будем связывать данную таблицу. В нашем случае, будет таблица взаимоотношений между пользователями, в которой мы будем сохранять дружеские связи между пользователями. По итогу получаем следующего вида класс-сущность (рис. 10).

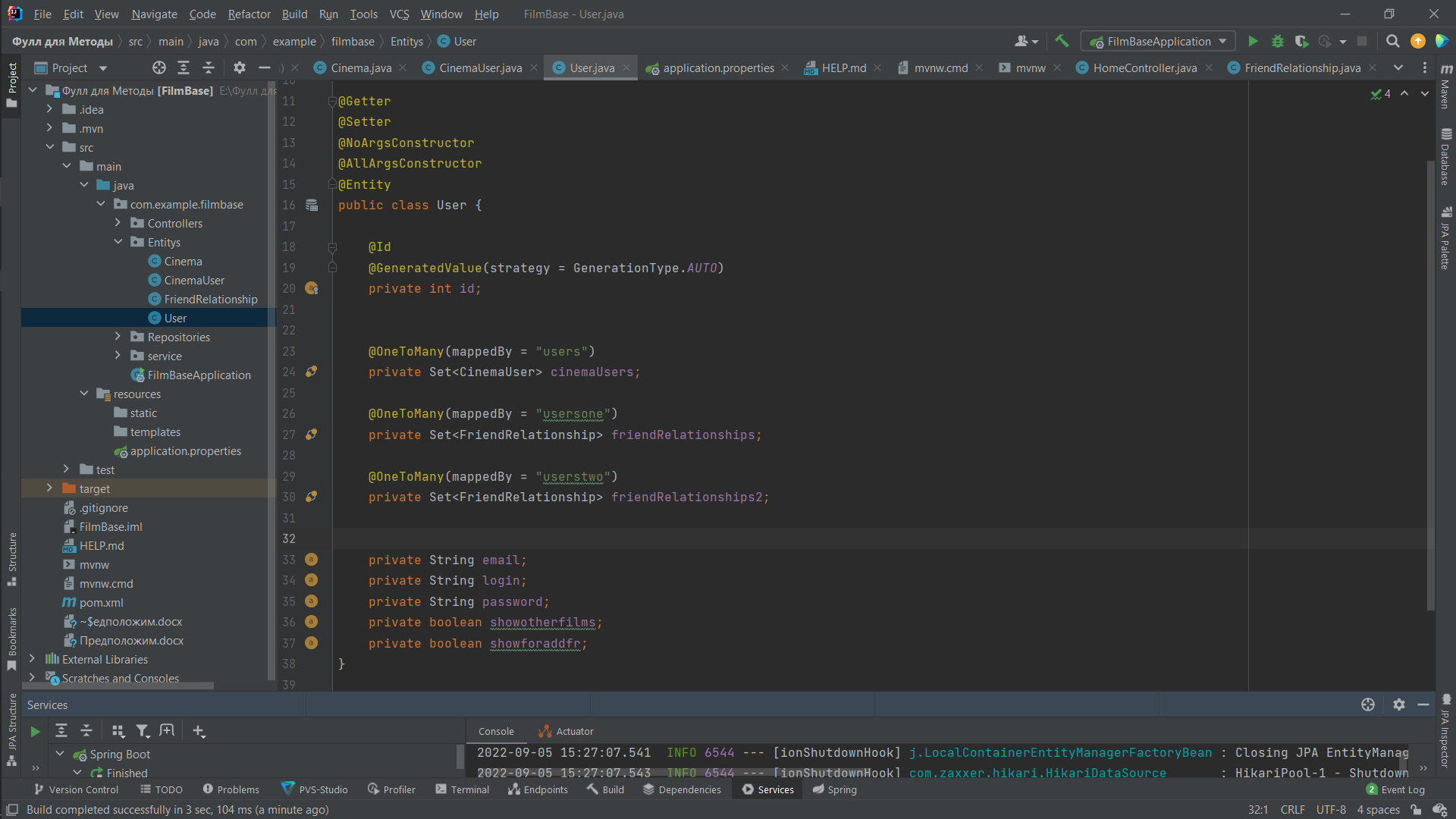


Рисунок 10.

Всего создаем четыре сущности: Пользователь, Фильм, Фильм Пользователя, Отношения между пользователями. Выше мы рассмотрели отношение один ко многим. Во избежание явно бесконечного зацикливания при получении данных после запросов к базе оставляем аннотацию @JsonIgnore, в связях один-ко-многим.

package com.example.filmbase.Entitys;  
  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import javax.persistence.\*;  
import java.util.Set;  
  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
public class Cinema {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private int id;  
  
  
 @JsonIgnore  
 @OneToMany(mappedBy = "cinemas")  
 private Set<CinemaUser> cinemaUsers;  
  
 private String urlimage;  
 private String headname;  
 private String director;  
 private String zhanr;  
 private String aboutis;  
 private int year;  
 private int marks;  
 private float rating;  
}

Соответственно на другом конце провода у нас будет отношения многие к одному. Рассмотрим на примере сущности “Фильм Пользователя” (рис. 11).

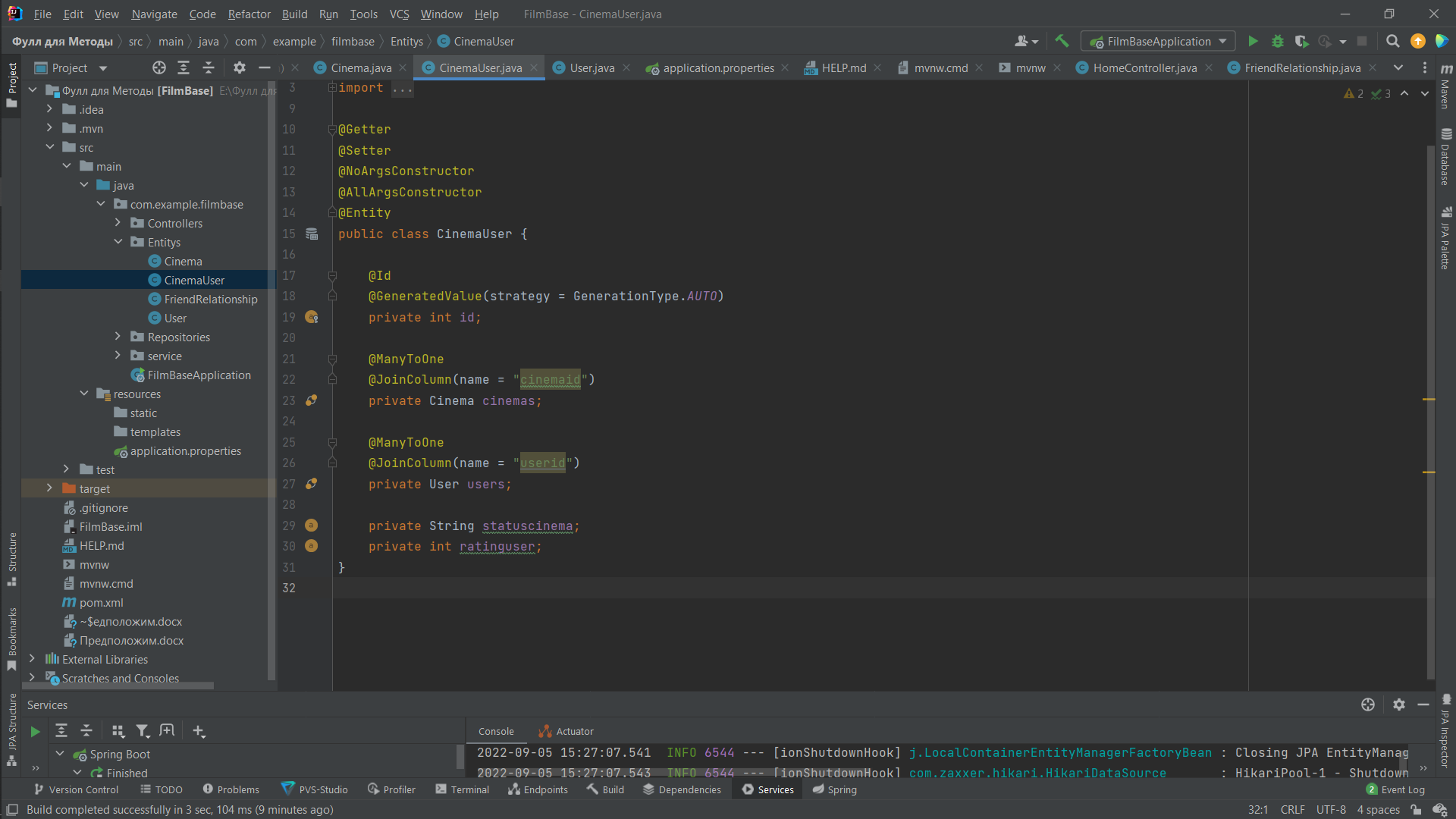


Рисунок 11

В данном случае, после аннотации многие к одному мы явно указываем какое поле у нас будет связываться с таблицей. Для этого используется аннотация @JoinColumn.

Далее идем в папку репозитория и создаем для всех сущностей интерфейсные репозитории для возможности делать запросы в базу и кода. Делаем это для всех сущностей по примеру ниже (рис. 12).

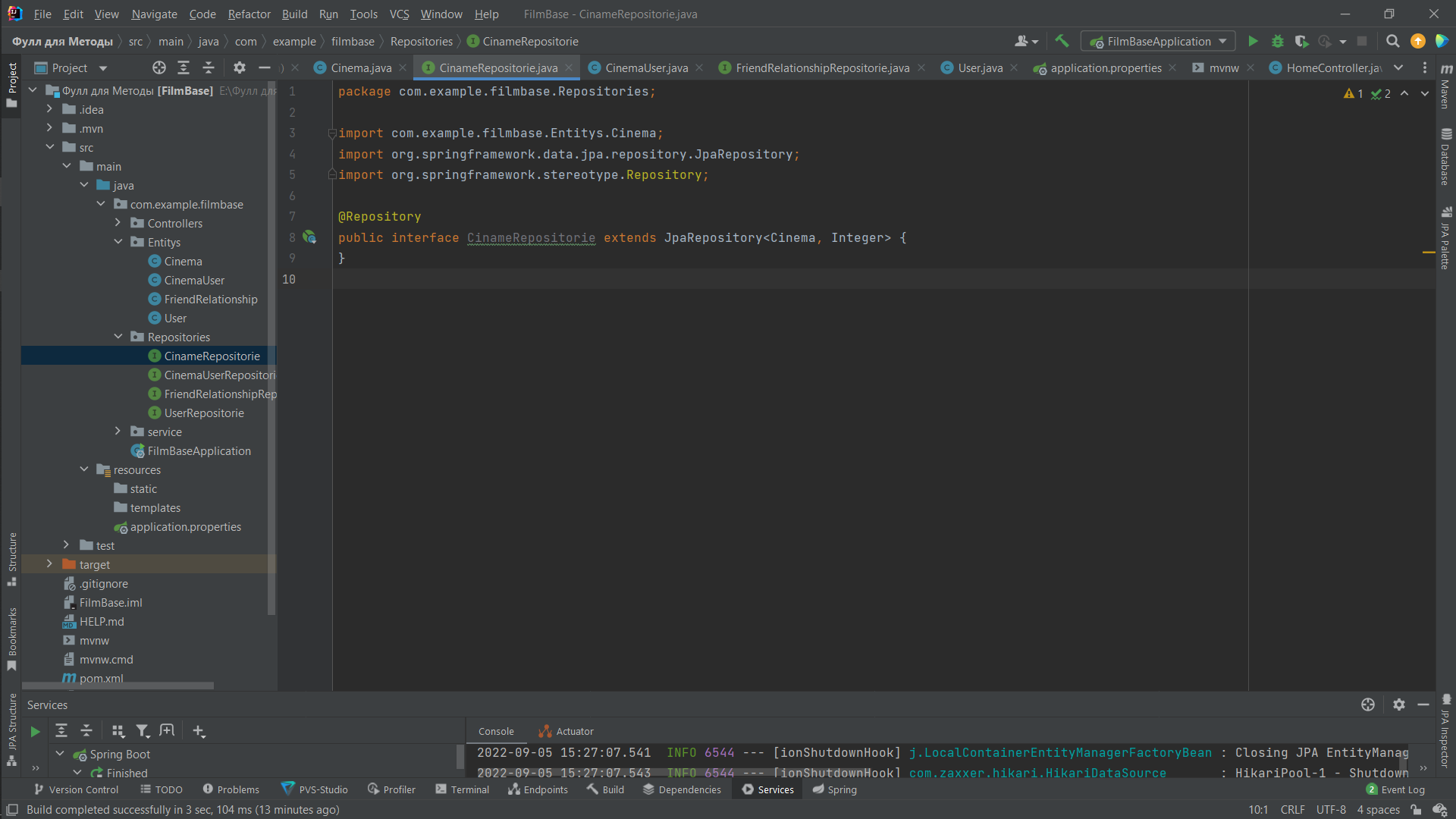


Рисунок 12

Далее создадим папку Service в которой мы дополним связь между кодом и базой данных. Создаем интерфейс для работы с сущностью пользователь (рис. 13).

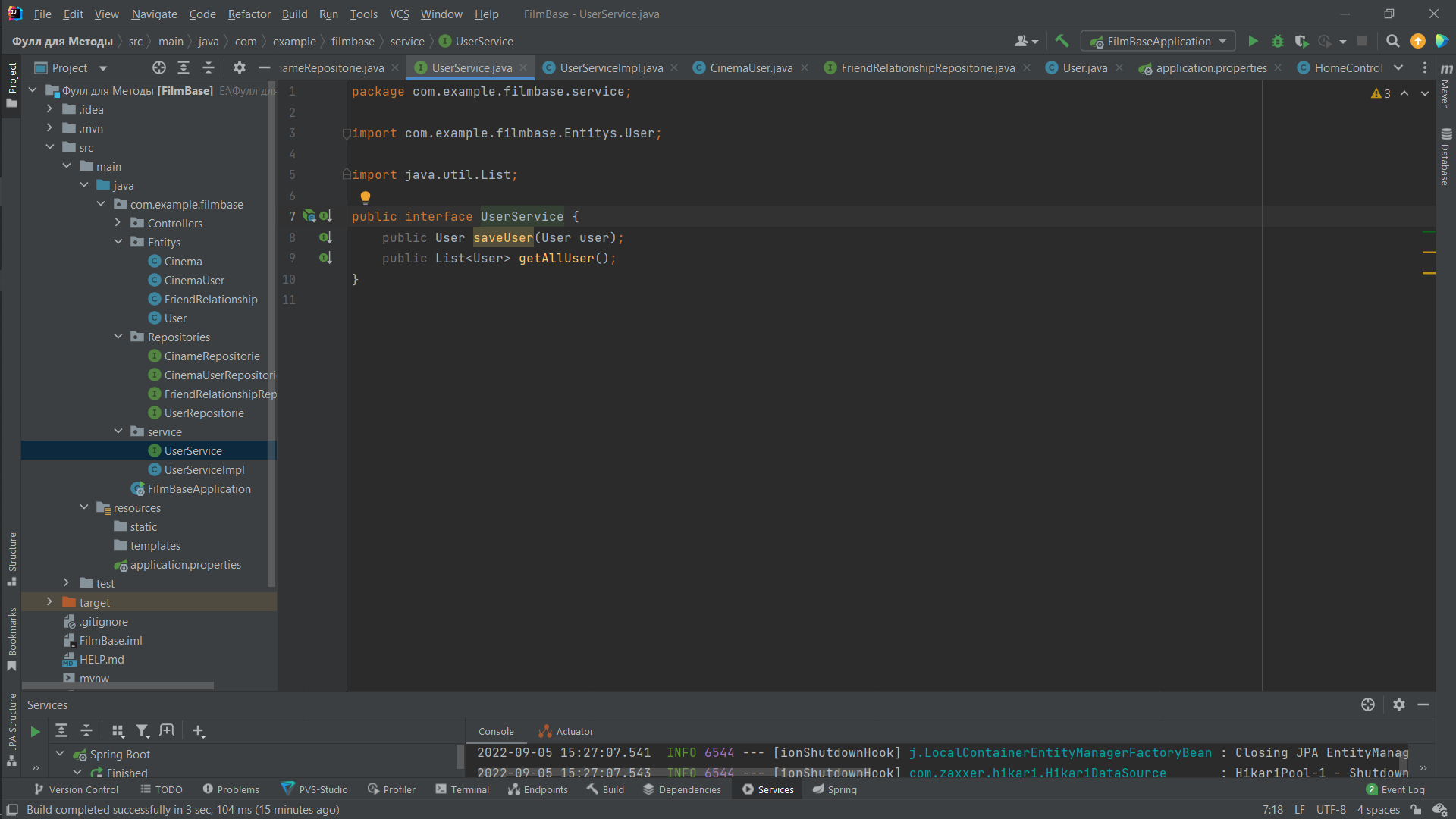


Рисунок 13.

В данном интерфейсе мы определяем два метода (пока что): для сохранения новых пользователей и для изъятия информации о всех пользователях из базы данных. Данные методы мы реализуем в следующем классе, представленном на рисунке 14.

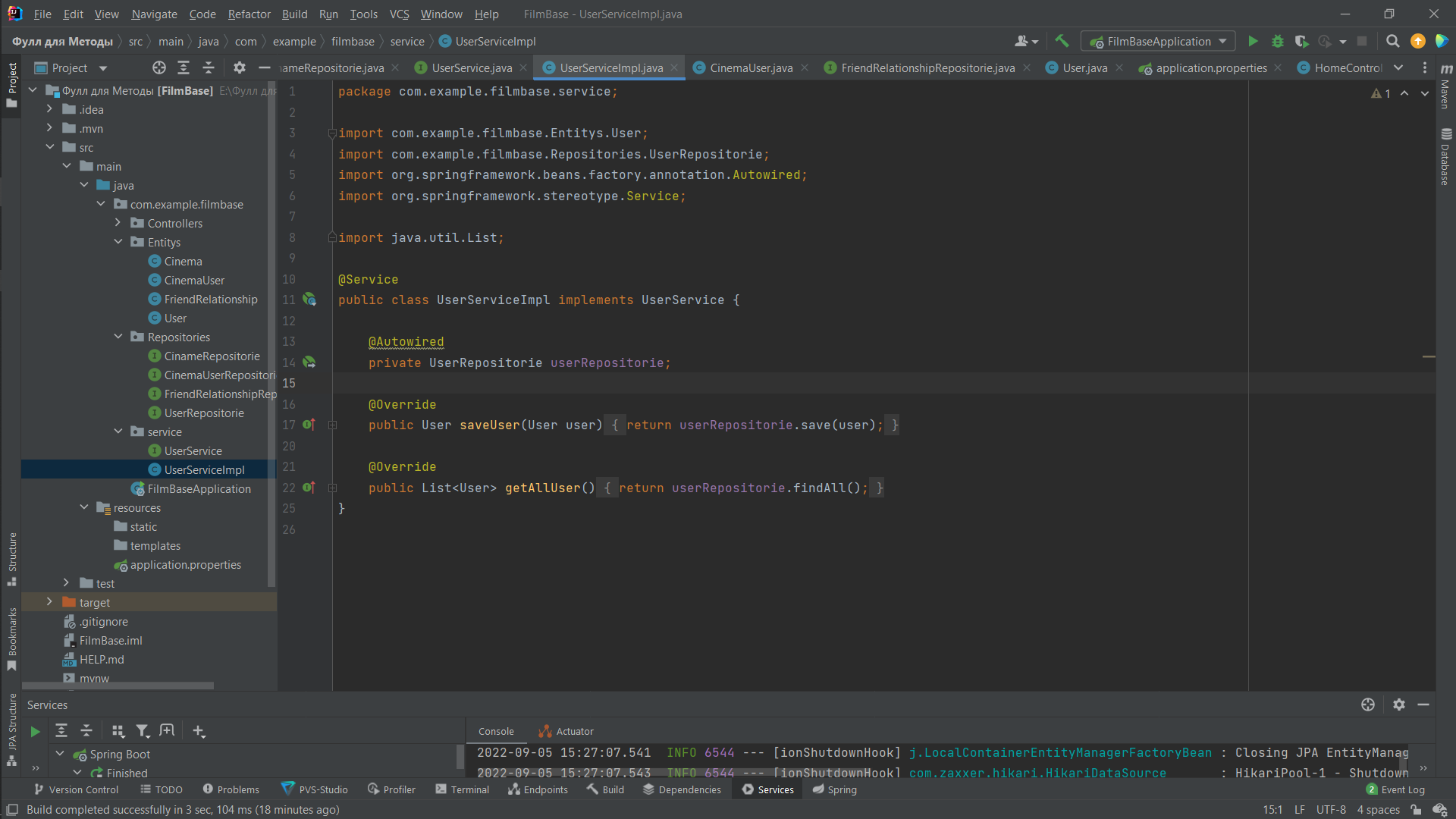


Рисунок 14.

Данный класс помечаем аннотацией @Service и наследуемся от интерфейса UserService. При помощи аннотации @Autowired обозначим созданный ранее репозиторий пользователей. А также реализуем функции сохранение и поиска в базе данных пользователей.

Такие же сервисы пропишем для каждой сущности.

Далее переходим к созданию обработки запросов от клиента на сервере. Для этого в папке контроллеров создадим класс HomeController. В которой пропишем некоторые функции для обработки запросов (рис. 15).

Перед объявлением класса добавим несколько важных аннотаций: @RestController – обозначаем, что данный класс является обработчиком запросов; @RequestMapping – маппим адрес для нашего обработчика (именно этот адрес будет значится после слэша стандартного localhost:8080.

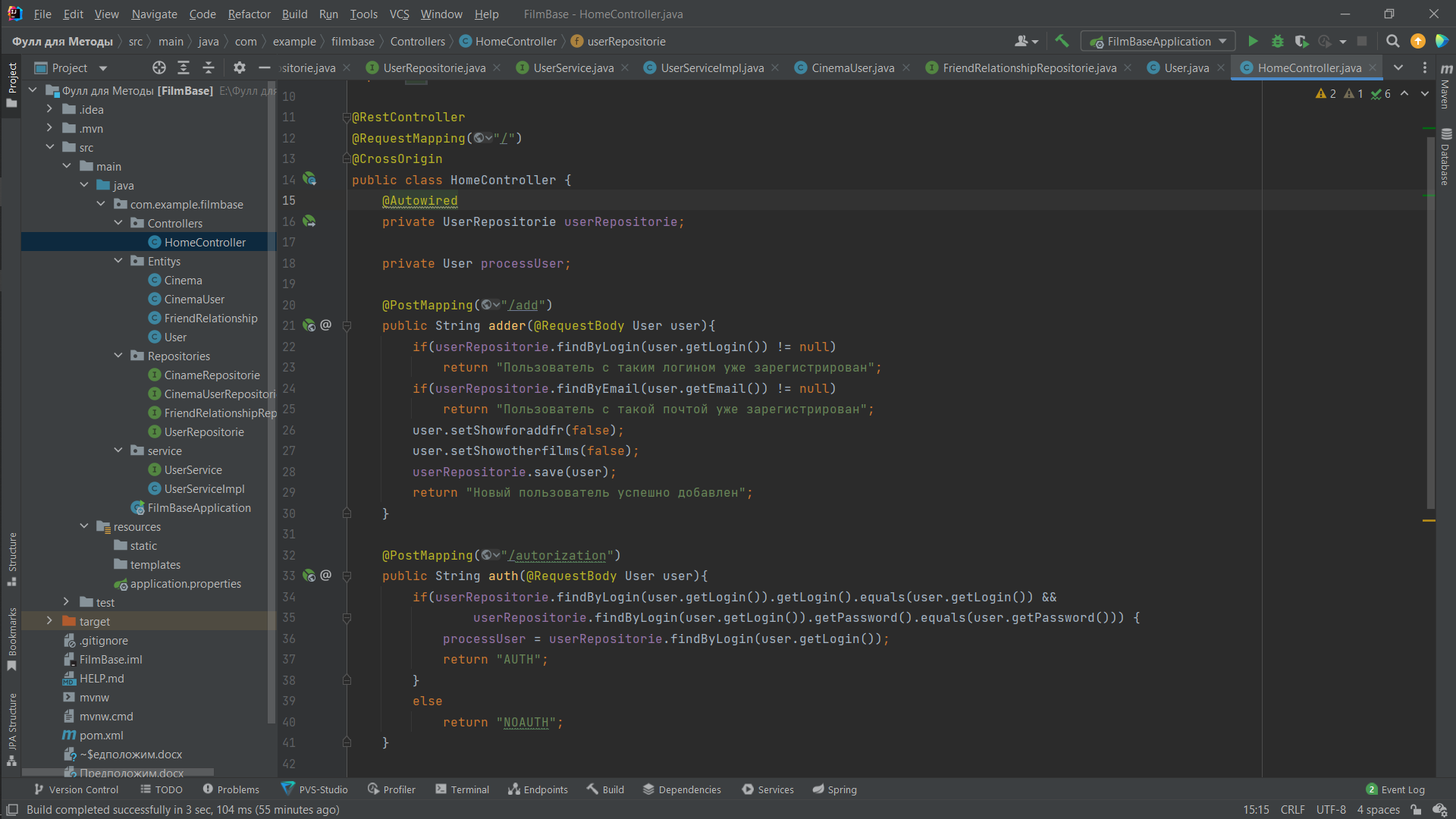


Рисунок 15.

Также отметим аннотацию @CrossOrigin которая служит для игнорирования заголовков пакетной передачи. В браузерах зачастую могут возникнуть проблемы с обработкой заголовков. Также, во избежание подобной ошибки рекомендуется, во время локальной разработки, устновить в браузер специальное расширение для игнорирования заголовков. Например, для браузеров, написанных на движке Chrome подойдет расширение CORS Unblock.

Далее в коде класса автоварим репозиторий пользователей, а также создаем приватный объект класса пользователь для того, чтобы запоминать информацию о авторизованном пользователе на сервере.

Аннотация PostMapping указывает на то, что в данной функции будет происходить прием информации, а аннотация RequestBody помечает экземпляр принимаемого класса. В скобках после PostMapping указываем адрес обработчика для данной функции. В данном классе реализованы функции регистрации и авторизации.

Теперь оставим ненадолго работу с сервером и перейдем к инициализации клиентской части приложения. Для этого на вашем компьютере должен быть установлен Node.js, npm или yarn (в данном случае используем npm) и Visual Studio Code. После установки следует проверить правильность и версию установленных ПО, как на рисунке ниже (рис. 16).

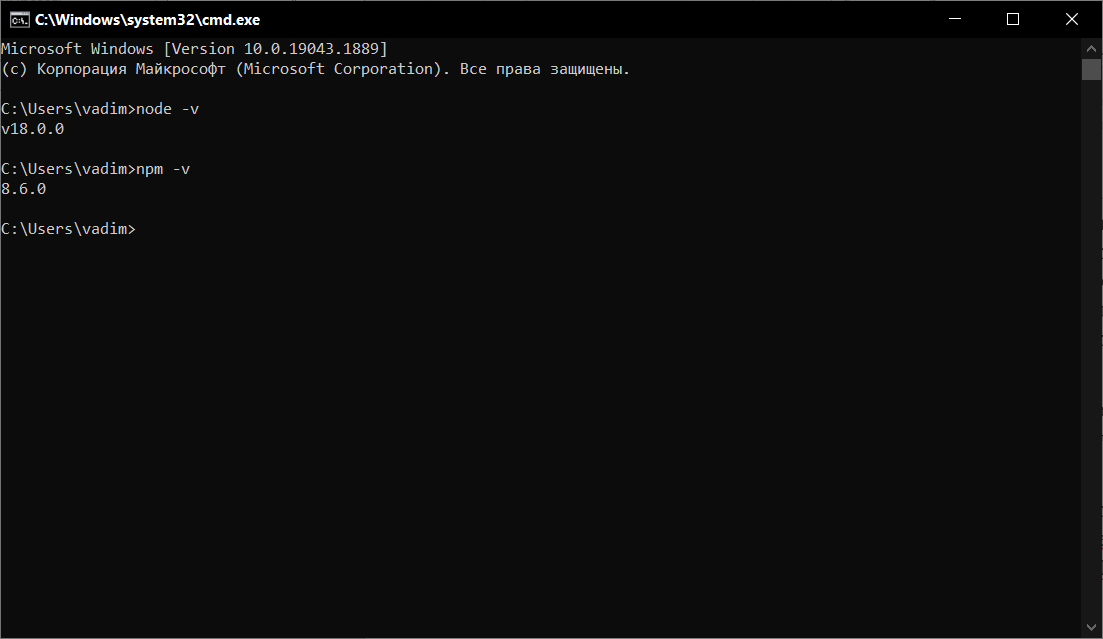


Рисунок 16.

Открываем VCode (Visual Studio Code). На верхнее панели жмем Terminal – New Terminal (рис. 17).



Рисунок 17.

Для начала перейдем в директорию, в которой будем инициализировать клиентскую часть. После чего пропишем следующую команду для инициализации проекта (рис 18).

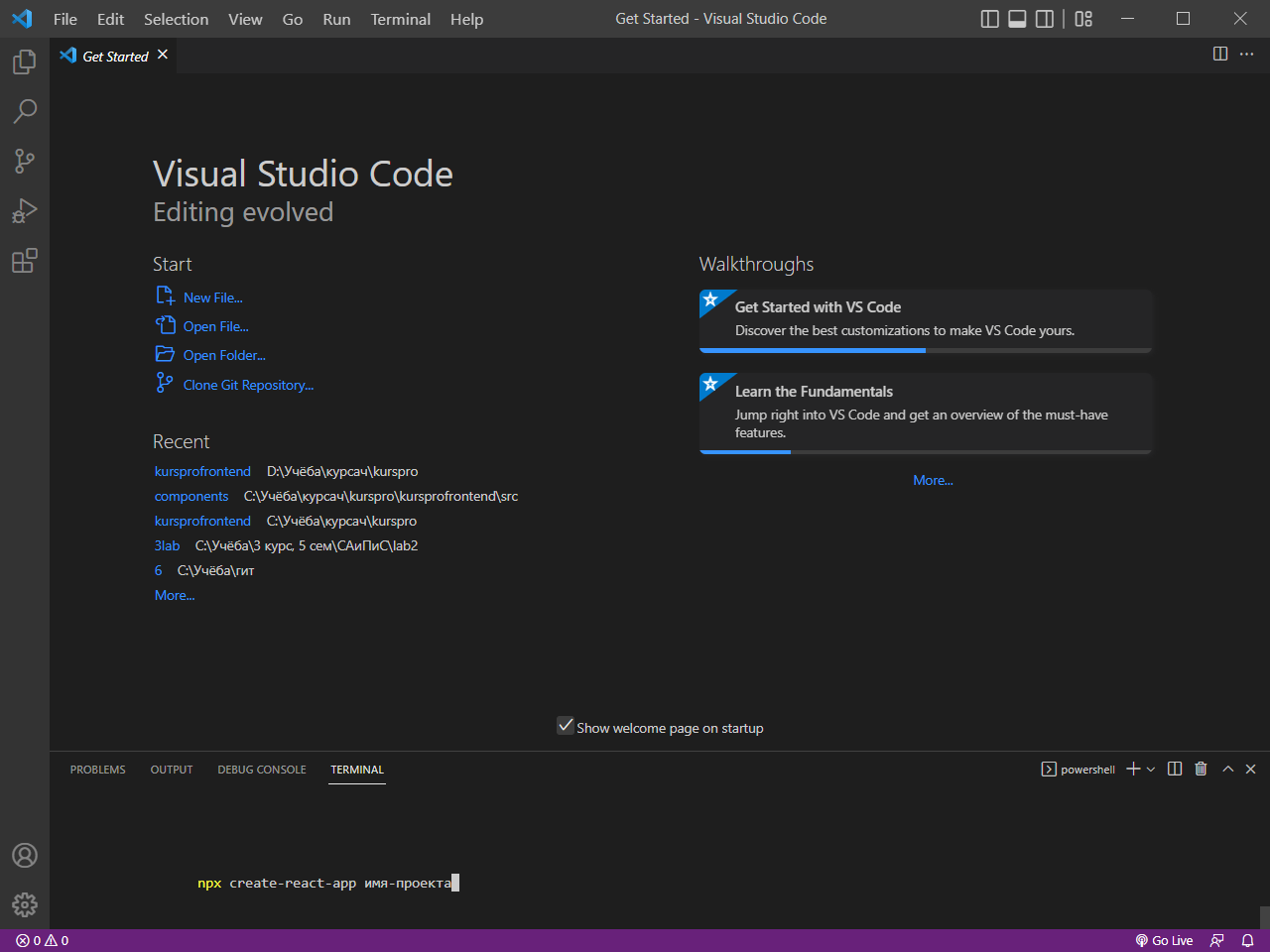


Рисунок 18.

Нажимаем ок. Для инициализации может потребоваться некоторое время. После инициализации переходим в директорию с проектом, название которой мы задавали при инициализации проекта, и пишем команду npm start. После успешного запуска должно открыться окно браузера следующего вида (рис. 19).

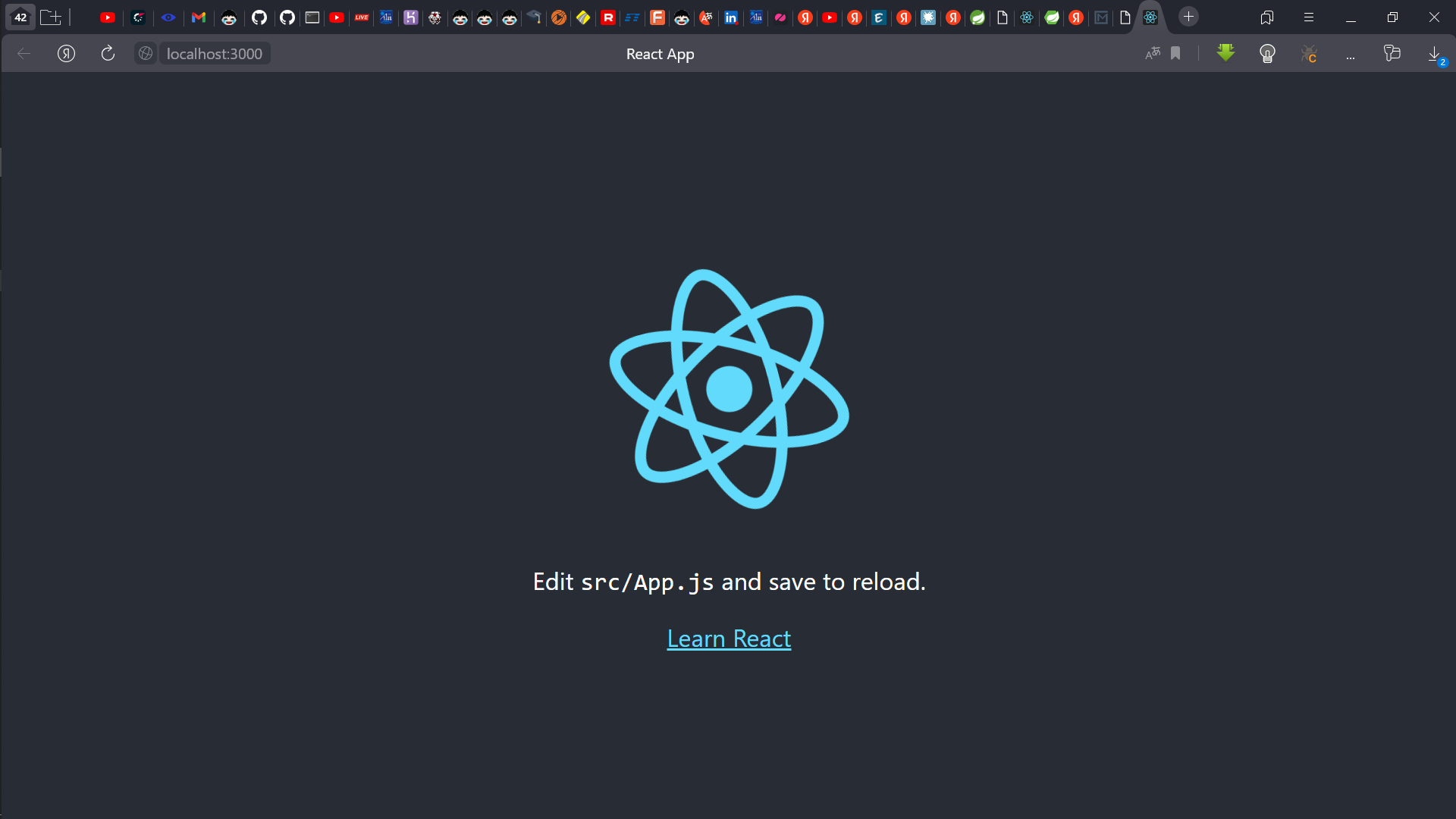


Рисунок 19.

Все работает, отлично. Добавим к проекту material-ui стилей, для этого в терминале пропишем соответствующую команду (рис. 20).

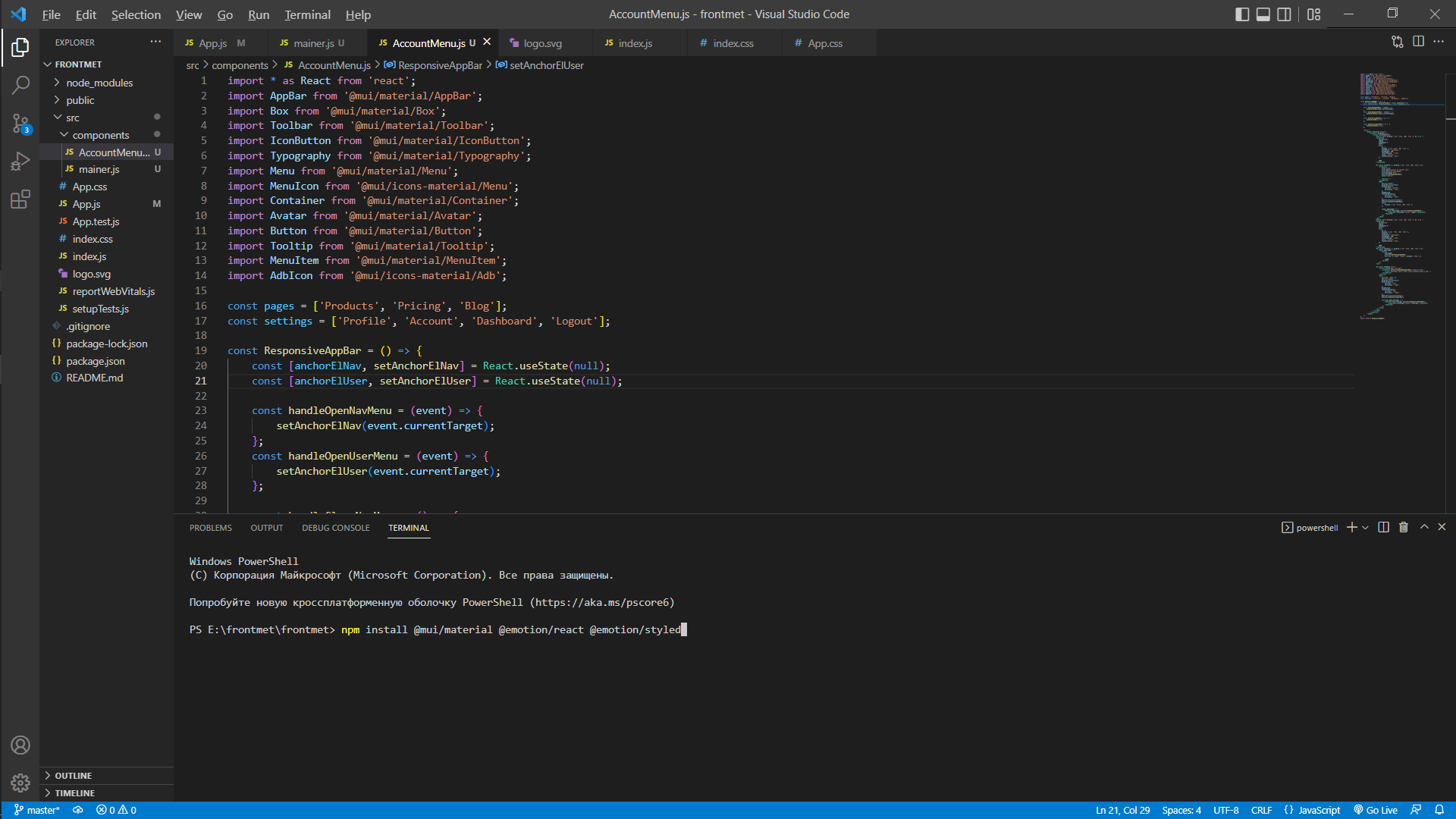


Рисунок 20.

Также подключим к проекту реакт-дом, для возможности работа с хуками. Все сторонние подключения выполняем через терминал (рис. 21).

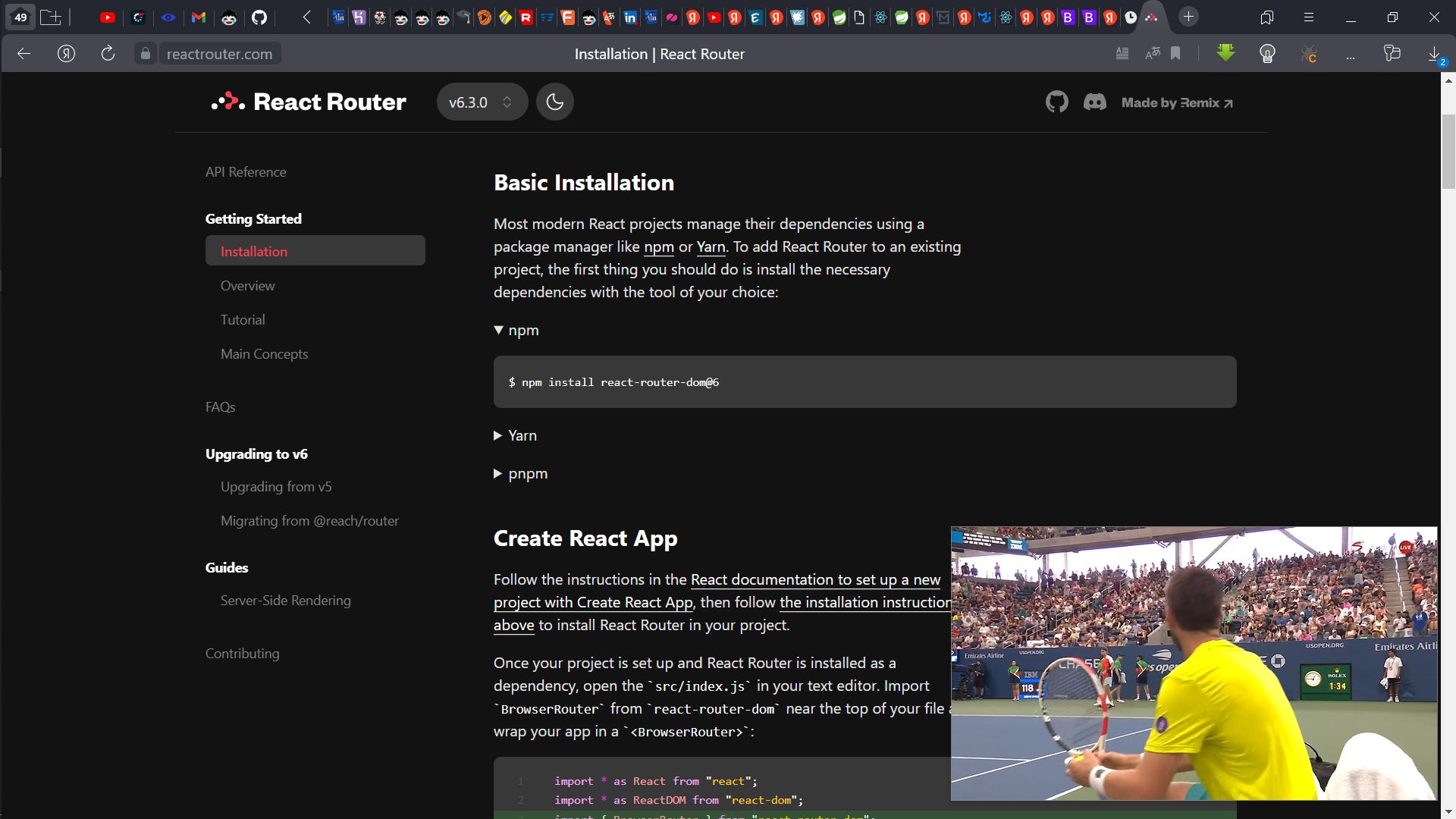


Рисунок 21.

Теперь откроем созданный проект в VCode и преобразуем содержимое файла Index.js. Создадим некую иерархию страниц для работы клиента с этой самой иерархией и блочное формирование страниц. Для этого пропишем следующий код в файле (рис. 22).

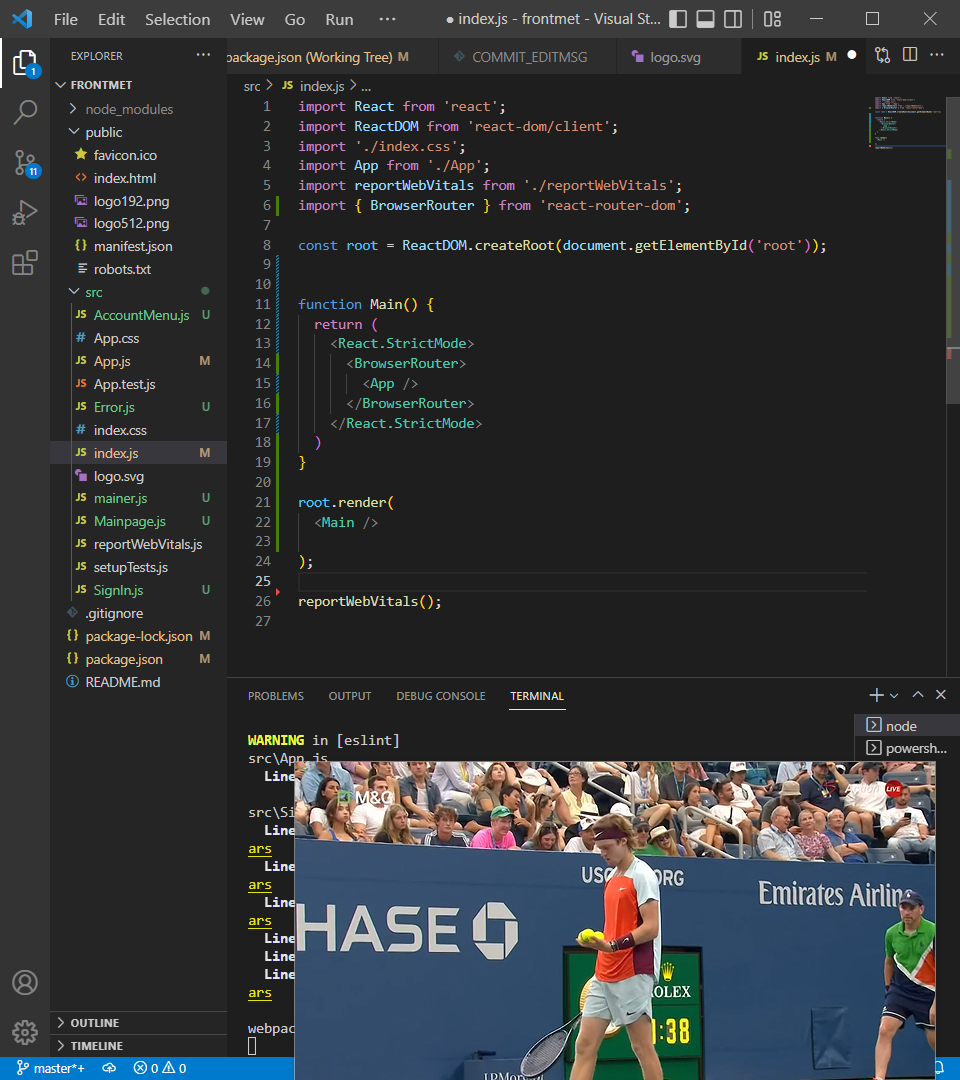


Рисунок 22.

В файле App.js пропишем следующий код (рис. 23). Теги Routs и Route позволяют создать иерархию страниц для клиента и то, по какому адресу какие javascript файлы будут подгружаться к странице. Отметим также присутствие адреса со звездочкой (\*), что означает открытие данного файла в случае ввода адреса, который отсутствует в списке роутов.

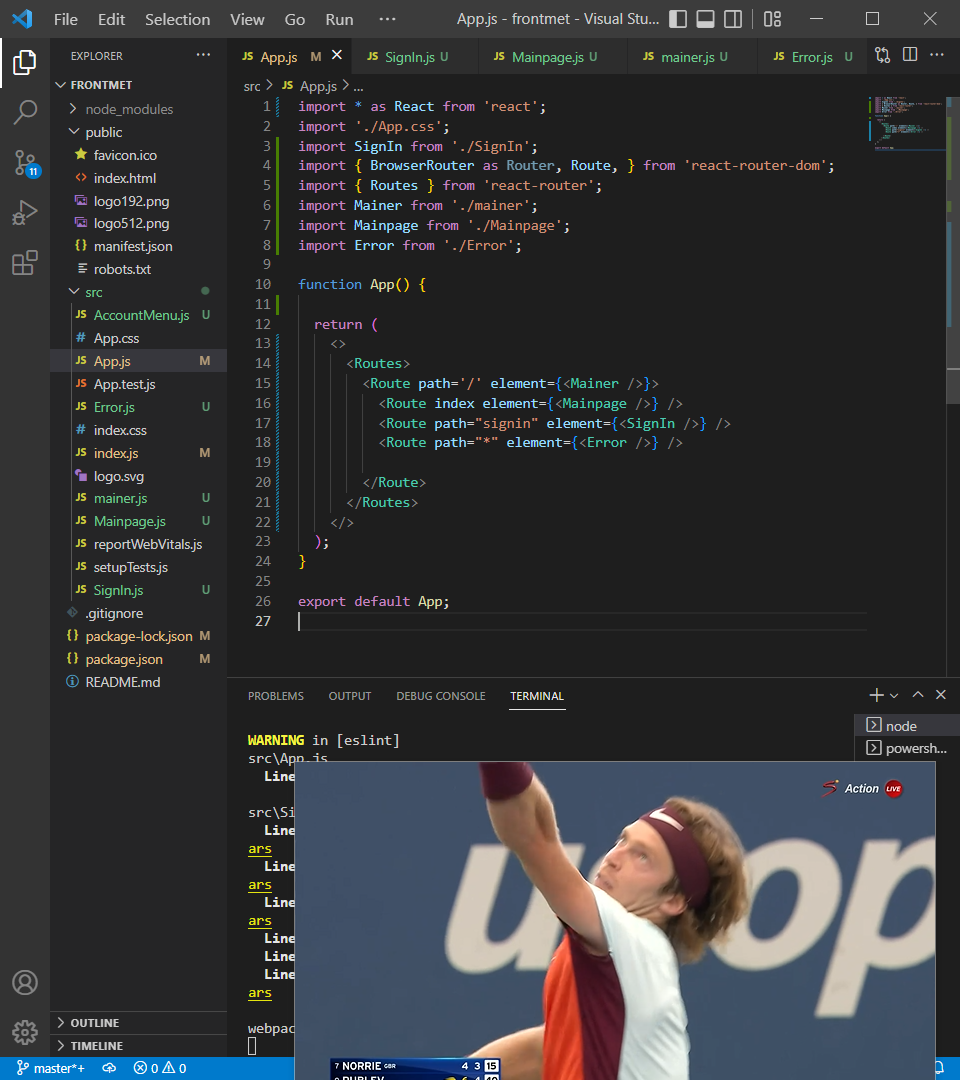


Рисунок 23.

Головным файлом согласно рисунка выше, будет открываться файл mainer.js. Структура этого файла и обозначает необходимую нам блочность. Для этого мы обозначаем шапку страницы, которая будет открываться всегда, и outlet-элемент, который и будет меняться, в зависимости от происходящего на странице (рис. 24).

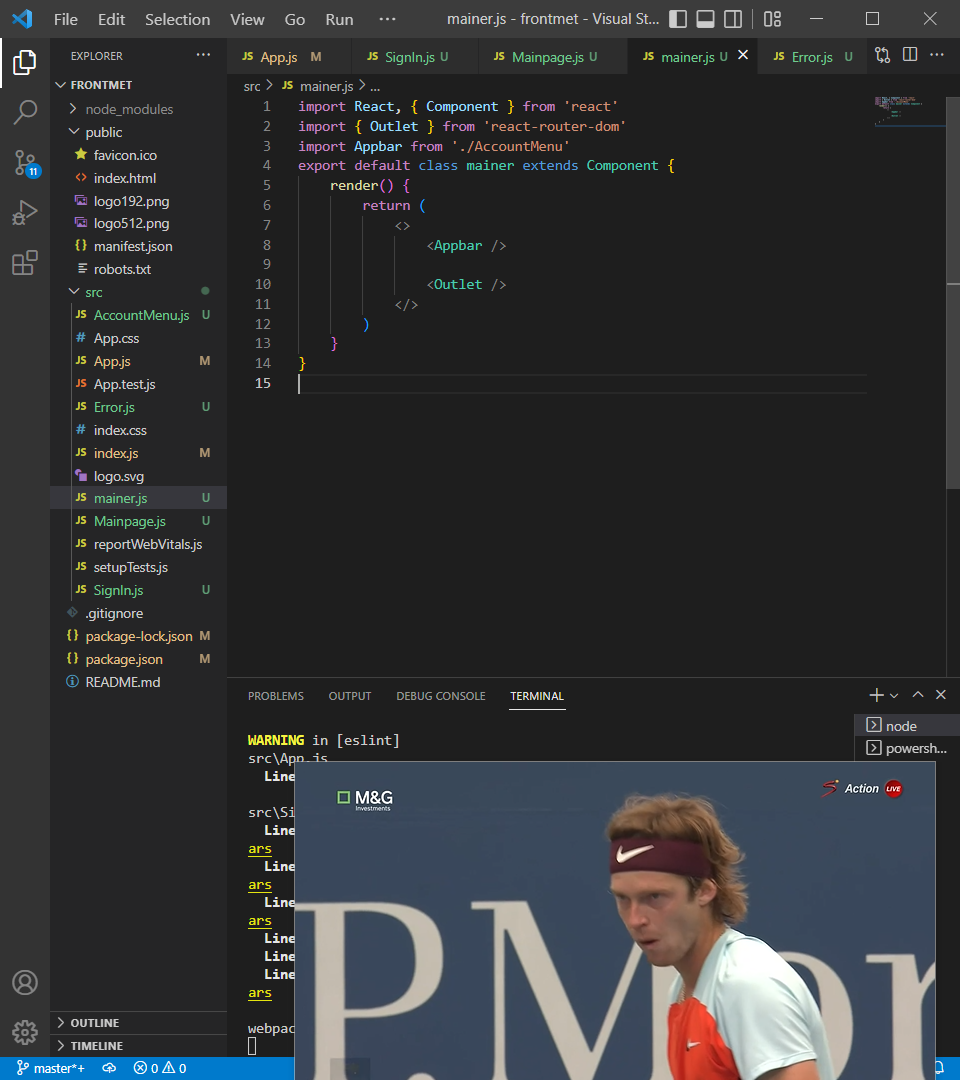


Рисунок 24.

Далее займемся созданием шапки страницы, подходящий шаблон находим на сайте mui.com в разделе AppBar. В дальнейшем по максимуму будем использовать примеры и шаблоны с этого сайта. Ничего дополнительно для работы с данными шаблонами подключать не нужно, потому как подключение мы выполнили сразу же после инициализации проекта. Для переноса кода шаблона просто копируем js код в наш файл проекта. В данном примере выбрали такой шаблон (рис. 25).

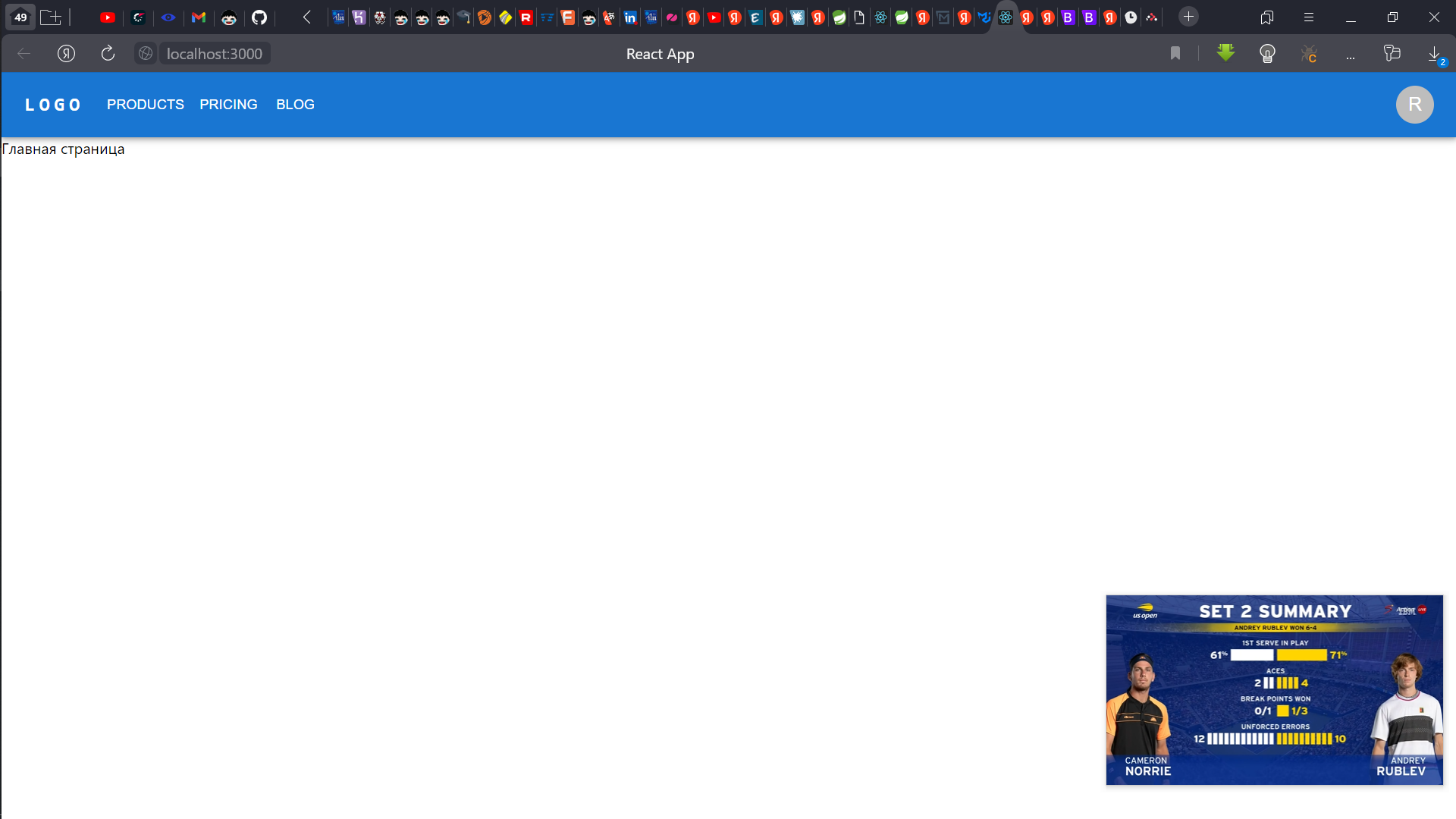


Рисунок 25.

При нажатии на каждую из кнопок мы сделаем проверку на авторизацию, если пользователь не авторизован, но нажал на какую-то из кнопок, то он будет переадресован на страницу авторизации. Страницу авторизации создадим также при помощи шаблонов, выглядеть она будет следующим образом (рис. 26).

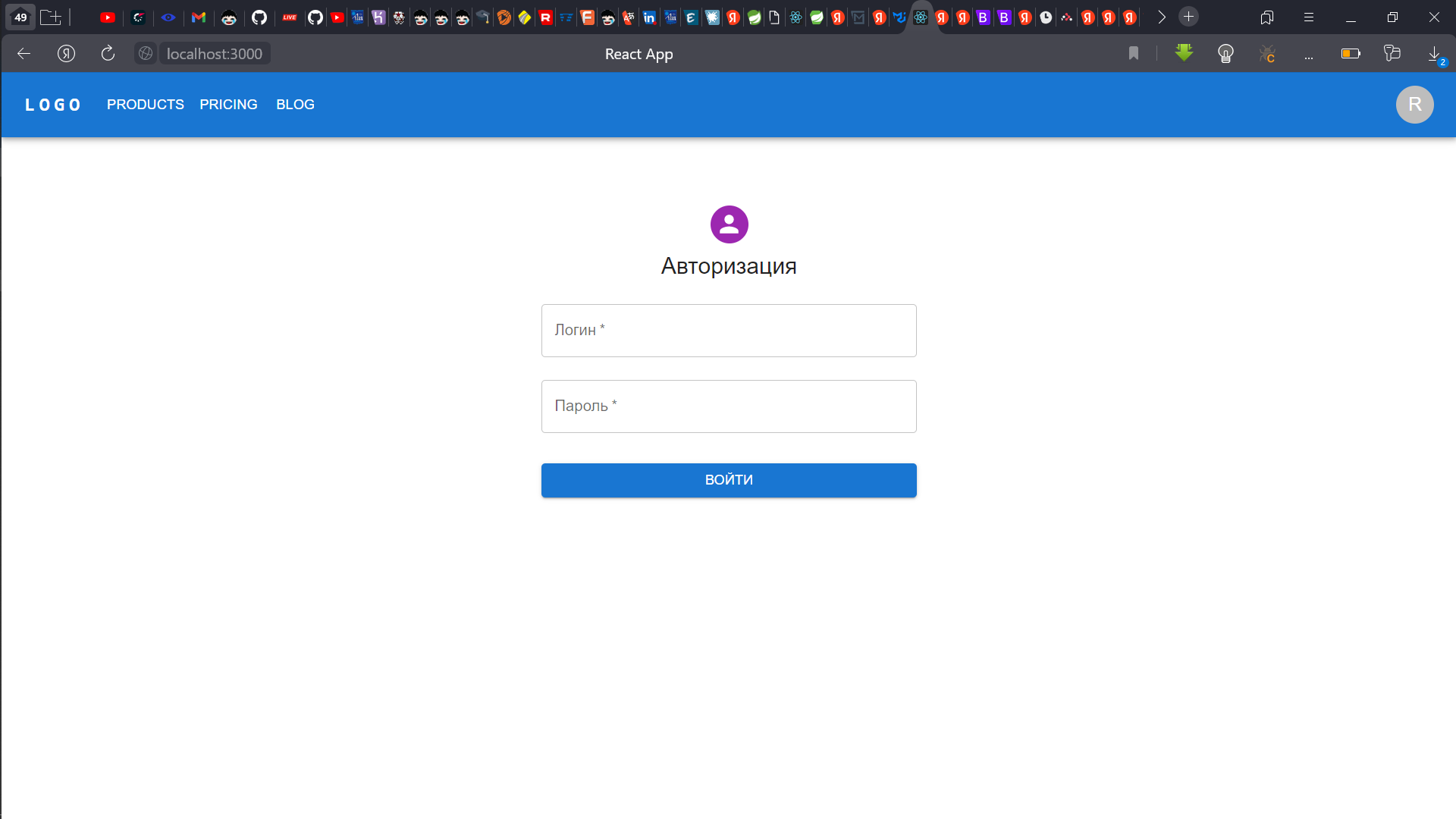


Рисунок 26.

Страница авторизации прописана в файле SignUp.js. Для того, чтобы корректно можно было проверить на стороне сервера введенную клиентом информацию, создаем объект с полями, названия которых идентичны названию полей в шаблоне объекта на стороне сервера, для того чтобы сервер корректно понимал, какую информацию ему нужно получить для проверки. В данном случае необходимо логин и пароль поместить в соответствующие поля login и password. Далее поместим эти поля в некий let объект и передадим его на сервер в формате JSON. Сервер прочитает эти поля и поместит их в объект класса User, причем в поля login и password соответственно. Остальные поля объекта User будут инициализированы как null. Именно по такому принципу и будет происходить связь между сервером и клиентом в нашем проекте.

Пропишем теперь всю логику нашего сервера. Важно! В проекте не используется Spring Security!

В папке контроллеров мы создали контроллер HomeController. Полный код данного класса, в котором, единственном, прописана вся логика работы сервера, представлен ниже:

package com.example.filmbase.Controllers;  
  
import com.example.filmbase.Entitys.Cinema;  
import com.example.filmbase.Entitys.CinemaUser;  
import com.example.filmbase.Entitys.FriendRelationship;  
import com.example.filmbase.Entitys.User;  
import com.example.filmbase.Repositories.CinameRepositorie;  
import com.example.filmbase.Repositories.CinemaUserRepositorie;  
import com.example.filmbase.Repositories.FriendRelationshipRepositorie;  
import com.example.filmbase.Repositories.UserRepositorie;  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/")  
@CrossOrigin  
public class HomeController {  
 @Autowired  
 private UserRepositorie userRepositorie;  
  
 @Autowired  
 private CinameRepositorie cinameRepositorie;  
  
 @Autowired  
 private CinemaUserRepositorie cinemaUserRepositorie;  
  
 @Autowired  
 private FriendRelationshipRepositorie friendRelationshipRepositorie;  
  
 private User processUser;  
  
 @PostMapping("/add")//добавление нового пользователя в базу данных  
 public String adder(@RequestBody User user){  
 if(userRepositorie.findByLogin(user.getLogin()) != null)  
 return "ERRORLOGIN";  
 if(userRepositorie.findByEmail(user.getEmail()) != null)  
 return "ERROREMAIL";  
 user.setShowforaddfr(false);  
 user.setShowotherfilms(false);  
 userRepositorie.save(user);  
 return "GOOD";  
 }  
  
 @PostMapping("/autorization")//авторизация пользователя  
 public String auth(@RequestBody User user){  
 if(userRepositorie.findByLogin(user.getLogin()).getLogin().equals(user.getLogin()) &&  
 userRepositorie.findByLogin(user.getLogin()).getPassword().equals(user.getPassword())) {  
 processUser = userRepositorie.findByLogin(user.getLogin());  
 return "AUTH";  
 }  
 else  
 return "NOAUTH";  
 }  
  
 @GetMapping("/allmyfriend")//передать клиенту всех друзей текущего авторизованного пользователя  
 public List<FriendRelationship> allmyfriend(){  
 return friendRelationshipRepositorie.findFriendRelationshipsByUsersone\_Id(processUser.getId());  
 }  
  
 @PostMapping("/allmyfriendsort")//сортировка друзей текущего авторизованного пользователя по логину  
 public List<FriendRelationship> allmyfriendsort(@RequestBody User user){  
 return friendRelationshipRepositorie.findFriendRelationshipsByUserstwo\_LoginContainingAndUsersone\_Id(user.getLogin(), processUser.getId());  
 }  
  
 @GetMapping("/allusersforfriend")//передать клиенту всех пользователей, у которых разрешено добавление в друзья другими пользователями  
 public List<User> allusersforfriend(){  
 List<User> userList = userRepositorie.findByShowforaddfr(true);  
 return userList;  
 }  
  
 @PostMapping("/allusersforfriendsort")//сортировка пользователей для добавление в друзья  
 public List<User> allusersforfriendsort(@RequestBody User user){  
 List<User> userList = userRepositorie.findByLoginContainingAndShowforaddfr(user.getLogin(), true);  
 return userList;  
 }  
  
  
  
 @PostMapping("/addmyfriend")//добавление выбранного пользователя в друзья к текущему авторизованному пользователю, производиться проверка  
 //если пользователь уже есть в друзьях у авторизованного, то ошибка  
 public String addmyfriend(@RequestBody User user){  
 if(user.getId() == processUser.getId())  
 return "ERROR";  
 if (friendRelationshipRepositorie.findFriendRelationshipsByUsersone\_IdAndUserstwo\_Id(processUser.getId(), user.getId()) != null)  
 return "ERROR1";  
 else {  
 FriendRelationship friendRelationship = new FriendRelationship();  
 friendRelationship.setUsersone(processUser);  
 friendRelationship.setUserstwo(userRepositorie.findById(user.getId()));  
 friendRelationshipRepositorie.save(friendRelationship);  
 return "GOOD";  
 }  
 }  
  
  
 @PostMapping("/deleteonmyfriend")//удаление пользователя из друзей текущего авторизованного пользователя  
 public String deleteonmyfriend(@RequestBody FriendRelationship friendRelationship){  
 if (friendRelationshipRepositorie.findFriendRelationshipsByUsersone\_IdAndUserstwo\_Id(processUser.getId(), friendRelationship.getUserstwo().getId()) != null) {  
 FriendRelationship friendRelationship1 = friendRelationshipRepositorie.findFriendRelationshipsByUsersone\_IdAndUserstwo\_Id(processUser.getId(), friendRelationship.getUserstwo().getId());  
 friendRelationshipRepositorie.delete(friendRelationship1);  
 return "GOOD";  
 }  
 else {  
 return "ERROR";  
 }  
 }  
  
 @GetMapping("/deauth")//в отсутствие использования spring security вручную производим логаут, в данном случае очищаем экземпляр класса User  
 public String deauth(){  
 processUser = null;  
 return "DEAUTH";  
 }  
  
  
  
 @PostMapping("/savechangemyaccount")//сохранение настроек приватности текущего авторизованного пользователя  
 public String savechangemyaccount(@RequestBody User user){  
 processUser.setShowotherfilms(user.isShowotherfilms());  
 processUser.setShowforaddfr(user.isShowforaddfr());  
 userRepositorie.save(processUser);  
 return "1. Настройки успешно изменены";  
 }  
  
 @PostMapping("/savemynewpass")//сохранение нового праоля, вместе с проверкой на валидность нового и старого  
 public String savemynewpass(@RequestBody User user){  
 if(!user.getLogin().equals(processUser.getPassword()))  
 return "2. Старый пароль введен неверно";  
 else{  
 if(!user.getPassword().equals(user.getEmail()))  
 return "3. Новый пароль не совпадает";  
 else {  
 processUser.setPassword(user.getPassword());  
 userRepositorie.save(processUser);  
 return "1. Пароль успешно изменен";  
 }  
 }  
 }  
  
 @GetMapping("/getprocessuser2")//проверка на авторизацию пользователя в системе  
 public String rs2(){  
 if(processUser == null)  
 return "NULL";  
 if(processUser != null && processUser.getLogin().equals("admin"))  
 return "ADMIN";  
 else  
 return "USER";  
  
 }  
  
 @GetMapping("/getAccount")//получение данных аккаунта  
 public User getAccount(){  
 User user1 = userRepositorie.findById(processUser.getId());  
 user1.setPassword(" ");  
 return user1;  
 }  
  
 @GetMapping("/allfilms")//передать на сторону клиента все фильмы, которые есть в базе  
 public List<Cinema> allfilms(){  
 return cinameRepositorie.findAll();  
 }  
  
 @PostMapping("/allfilmssort")//отсортировать фильмы либо по жанру, либо по названию, либо по режисеру, причем  
 //используем для этого один переданный на сервер параметр, отправленных в поле названия  
 public List<Cinema> allfilmssort(@RequestBody Cinema cinema){  
 List<Cinema> cinema1 = cinameRepositorie.findByHeadnameContainingOrZhanrContainingOrDirectorContaining(cinema.getHeadname(), cinema.getHeadname(), cinema.getHeadname());  
 return cinema1;  
 }  
  
 @GetMapping("/allmyfilms")//все фильмы текущего авторизованного пользователя  
 public List<CinemaUser> allmyfilms(){  
 List<CinemaUser> fer = cinemaUserRepositorie.findAllByUsers(processUser);  
 return fer;  
 }  
  
  
 @PostMapping("/showusercollection")//отобразить коллекцию выбранного из друзей пользователя  
 public List<CinemaUser> showusercollection(@RequestBody User user){  
 User user1 = userRepositorie.findById(user.getId());  
 List<CinemaUser> cinemaUser = cinemaUserRepositorie.findAllByUsers(user1);  
 return cinemaUser;  
 }  
  
 @PostMapping("/showusercollectionsort")//отсортировать фильмы из колекции пользователя-друга текущего авторизованного  
 public List<CinemaUser> showusercollectionsort(@RequestBody Cinema cinema){  
 User user1 = userRepositorie.findById(cinema.getId());  
 List<CinemaUser> cinemaUser = cinemaUserRepositorie.findCinemaUsersByCinemas\_HeadnameContainingAndUsers(cinema.getHeadname(), user1);  
 return cinemaUser;  
 }  
  
 @PostMapping("/allmyfilmssort")//сортировка фильмов коллекци текущего авторизованного пользователя  
 public List<CinemaUser> allmyfilmssort(@RequestBody Cinema cinema){  
 return cinemaUserRepositorie.findCinemaUsersByCinemas\_HeadnameContainingAndUsers(cinema.getHeadname(), processUser);  
 }  
  
 @PostMapping("/addmyfilm")//добавление фильма в коллекцию текущего авторизованного пользователя  
 public String addmyfilm(@RequestBody Cinema cinema){  
 if(cinemaUserRepositorie.findByCinemasAndUsers(cinema, processUser) != null) {  
 return "ERROR";  
 }  
 else {  
 CinemaUser cinemaUser = new CinemaUser();  
 cinemaUser.setCinemas(cinema);  
 cinemaUser.setUsers(processUser);  
 cinemaUser.setStatuscinema("Без статуса просмотра");  
 cinemaUser.setRatinguser(0);  
 cinemaUserRepositorie.save(cinemaUser);  
 return "GOOD";  
 }  
 }  
  
 @PostMapping("/addnewfilm")//добавление нового фильма в базу  
 public String addnewfilm(@RequestBody Cinema cinema){  
 if(cinameRepositorie.findByHeadnameAndYear(cinema.getHeadname(), cinema.getYear()).size() != 0) {  
 return "Этот фильм уже есть в базе";  
 }  
 else {  
 cinameRepositorie.save(cinema);  
 return "1. Фильм успешно добавлен в каталог";  
 }  
 }  
  
  
 @PostMapping("/editfilm")//редактирование выбранного фильма, данной функией обладает только администратор  
 public String editfilm(@RequestBody Cinema cinema){  
 Cinema cinema1 = cinameRepositorie.findById(cinema.getId());  
 cinema1.setHeadname(cinema.getHeadname());  
 cinema1.setAboutis(cinema.getAboutis());  
 cinema1.setDirector(cinema.getDirector());  
 cinema1.setUrlimage(cinema.getUrlimage());  
 cinema1.setYear(cinema.getYear());  
 cinema1.setZhanr(cinema.getZhanr());  
 cinameRepositorie.save(cinema1);  
 return "1. Изменения успешно сохранены";  
 }  
  
 @PostMapping("/deleteonemyfilm")//удаление фильма из коллекции фильмов текущего авторизованного пользователя  
 public String deleteonemyfilm(@RequestBody Cinema cinema){  
 if(cinemaUserRepositorie.findByCinemasAndUsers(cinema, processUser) != null) {  
 CinemaUser cinemaUser = cinemaUserRepositorie.findByCinemasAndUsers(cinema, processUser);  
 cinemaUserRepositorie.delete(cinemaUser);  
 return "1. Фильм успешно удален из вашего каталога";  
 }  
 else {  
 return "Произошла ошибка, повторите попытку позже";  
 }  
 }  
  
  
 @PostMapping("/savechangemyfilm")//сохранение оценки к фильму от пользователя, у которого этот фильм находиться в коллекции  
 public String savechangemyfilm(@RequestBody CinemaUser cinemaUser){  
 CinemaUser cinemaUser1 = cinemaUserRepositorie.findById(cinemaUser.getId());  
 if(cinemaUser1 != null) {//проверка есть ли этот фильм в базе  
 if(cinemaUser.getRatinguser() != 0){//проверка, оценивал ли этот фильм пользователь ранее  
 Cinema cinema = cinemaUser1.getCinemas();  
 int kolvoocenok = cinema.getMarks();  
 float ocenka = cinema.getRating(), promocenka = cinemaUser1.getRatinguser();  
 if (kolvoocenok == 0) {//если количество оценок равно нулю, то есть фильм не имеет оценок на портале  
 cinema.setRating(cinemaUser.getRatinguser());  
 cinema.setMarks(1);  
 }  
 if(kolvoocenok > 0 && promocenka != 0){//если оценки к фильму уже есть и есть оценка от пользователя  
 cinema.setRating((kolvoocenok \* ocenka + cinemaUser.getRatinguser() - promocenka)/ (kolvoocenok));  
 cinema.setMarks(cinema.getMarks());  
 }  
 if(kolvoocenok > 0 && promocenka == 0){  
 cinema.setRating((kolvoocenok \* ocenka + cinemaUser.getRatinguser())/ (kolvoocenok + 1));  
 cinema.setMarks(cinema.getMarks() + 1);  
 }  
 cinemaUser1.setCinemas(cinema);  
 cinemaUser1.setRatinguser(cinemaUser.getRatinguser());  
 }  
  
 cinemaUser1.setStatuscinema(cinemaUser.getStatuscinema());  
 cinemaUserRepositorie.save(cinemaUser1);  
 return "1. Изменения успешно сохранены";  
 }  
 else {  
 return "Произошла ошибка, повторите попытку позже";  
 }  
 }  
}

Отметим, что внутри этого класса, при авторизации, запоминаем данные в созданном постоянном экземпляре класса User. Делаем это для более упрощенной сверки данных в связанных с другими пользователями данными, в частности, для сущности дружественных взаимоотношений между пользователями.

В остальном, каждая отдельная функция выполняет свои роли в связи между сервером и клиентом, каждая из них прописана в комментариях к функциям.

Как уже было сказано выше, в проекте используются четыре сущности (таблицы), к каждой из которых можно обращаться либо через сервисы, либо через репозитории (как в нашем случае).

Репозиторий сущностей:

UserRepositorie:

package com.example.filmbase.Repositories;  
  
import com.example.filmbase.Entitys.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
  
@Repository  
public interface UserRepositorie extends JpaRepository<User, Integer> {  
 User findByLogin(String a);  
 User findById(int i);  
 List<User> findByShowforaddfr(boolean v);  
  
 List<User> findByLoginContainingAndShowforaddfr(String a, boolean v);  
 User findByEmail(String a);  
}

CinameRepositorie:

@Repository  
public interface CinameRepositorie extends JpaRepository<Cinema, Integer> {  
 Cinema findById(int t);  
 List<Cinema> findByHeadnameContainingOrZhanrContainingOrDirectorContaining(String a, String b, String c);  
 List<Cinema> findByHeadnameAndYear(String a, int b);

CinemaUserRepositorie:

@Repository  
public interface CinemaUserRepositorie extends JpaRepository<CinemaUser, Integer> {  
 CinemaUser findByCinemasAndUsers(Cinema a, User b);  
 List<CinemaUser> findCinemaUsersByCinemas\_HeadnameContainingAndUsers(String a, User b);  
 CinemaUser findById(int id);  
 List<CinemaUser> findAllByUsers(User user);

FrindRelationshipRepositorie:

@Repository  
public interface FriendRelationshipRepositorie extends JpaRepository<FriendRelationship, Integer> {  
 FriendRelationship findFriendRelationshipsByUsersone\_IdAndUserstwo\_Id(int a, int b);  
 List<FriendRelationship> findFriendRelationshipsByUsersone\_Id(int r);  
 List<FriendRelationship> findFriendRelationshipsByUserstwo\_LoginContainingAndUsersone\_Id(String a, int b);

Во всех случаях наследуемся от JpaRepository. Так же следует заметить, что все названия функции должны быть прописаны четко, в соответствии с запрашиваемыми стандартами спринга. “Idea” в частности подсказывает, как необходимо называть функции для их корректной генерации, в противном случае, проект просто будет крошиться.

Это все, что касается серверной части, теперь рассмотрим клиентскую часть

import \* as React from 'react';

import './App.css';

import SignIn from './SignIn';

import SignUp from './SignUp';

import { Route } from 'react-router-dom';

import { Routes } from 'react-router';

import Mainer from './mainer';

import Mainpage from './Mainpage';

import Allfilms from './Allfilms';

import Myfilms from './Myfilms';

import Searchfriend from './Searchfriend';

import Error from './Error';

import Addnewfilm from './Addnewfilm';

import Myonefilm from './Myonefilm';

import Logout from './Logout';

import Settingsaccount from './Settingsaccount';

import Myfriends from './Myfriends';

import Showusercollection from './Showusercollection';

import Editfilm from './Editfilm';

function App() {

  return (

    <>

      <Routes>

        <Route path='/' element={<Mainer />}>

          <Route index element={<Mainpage />} />

          <Route path="signin" element={<SignIn />} />

          <Route path="logout" element={<Logout />} />

          <Route path="myfriends" element={<Myfriends />} />

          <Route path="settingsaccount" element={<Settingsaccount />} />

          <Route path="allfilms" element={<Allfilms />} />

          <Route path="editfilm" element={<Editfilm />} />

          <Route path="showusercollection" element={<Showusercollection />} />

          <Route path="myfilms" element={<Myfilms />} />

          <Route path="addnewfilm" element={<Addnewfilm />} />

          <Route path="searchfriend" element={<Searchfriend />} />

          <Route path="myonefilm" element={<Myonefilm />} />

          <Route path="signup" element={<SignUp />} />

          <Route path="\*" element={<Error />} />

        </Route>

      </Routes>

    </>

  );

}

export default App;

Выше представлен файл App.js , в котором, как было сказано выше, прописывается весь роутинг по клиентской части.

Ниже представлен код шапки сайта:

import \* as React from 'react';

import AppBar from '@mui/material/AppBar';

import Box from '@mui/material/Box';

import Toolbar from '@mui/material/Toolbar';

import IconButton from '@mui/material/IconButton';

import Typography from '@mui/material/Typography';

import Menu from '@mui/material/Menu';

import MenuIcon from '@mui/material/Menu';

import Container from '@mui/material/Container';

import Avatar from '@mui/material/Avatar';

import Button from '@mui/material/Button';

import Tooltip from '@mui/material/Tooltip';

import MenuItem from '@mui/material/MenuItem';

import AdbIcon from '@mui/material/Menu';

export default function AccountMenu() {

    const [whouser, setWho] = React.useState();

    React.useState(() => {

        proverka();

        console.log(whouser);

    })

    const pages = [{ name: 'ФИЛЬМЫ', hreff: '/allfilms' }, { name: 'МОИ ФИЛЬМЫ', hreff: '/myfilms' }, { name: 'ПОИСК ДРУЗЕЙ', hreff: '/searchfriend' }];

    const settings = [{ name: 'ДОБАВИТЬ ФИЛЬМ', hreff: '/addnewfilm' },

    { name: 'НАСТРОЙКИ ВИДИМОСТИ', hreff: '/settingsaccount' },

    { name: 'МОИ ДРУЗЬЯ', hreff: '/myfriends' },

    { name: 'ВЫХОД', hreff: '/logout' }];

    const [anchorElNav, setAnchorElNav] = React.useState(null);

    const [anchorElUser, setAnchorElUser] = React.useState(null);

    async function proverka() {

        let status;

        const response = await fetch("http://localhost:8080/getprocessuser2");

        await response.text()

            .then((json) => {

                status = json;

                if (status === 'ADMIN') {

                    setWho("ADMIN");

                }

                if (status === 'USER') {

                    setWho("USER");

                }

                if (status === 'NULL') {

                    setWho("NULL");

                }

            });

    }

    const handleOpenNavMenu = (event) => {

        setAnchorElNav(event.currentTarget);

    };

    const handleOpenUserMenu = (event) => {

        setAnchorElUser(event.currentTarget);

    };

    const handleCloseNavMenu = (event) => {

        setAnchorElNav(event.currentTarget.page);

        console.log(event.currentTarget);

    };

    const handleCloseUserMenu = () => {

        setAnchorElUser(null);

    };

    return (

        <AppBar position="static">

            <Container maxWidth="xl">

                <Toolbar disableGutters>

                    <AdbIcon sx={{ display: { xs: 'none', md: 'flex' }, mr: 1 }} />

                    <Typography

                        variant="h6"

                        noWrap

                        component="a"

                        href="/"

                        sx={{

                            mr: 2,

                            display: { xs: 'none', md: 'flex' },

                            fontFamily: 'monospace',

                            fontWeight: 700,

                            letterSpacing: '.3rem',

                            color: 'inherit',

                            textDecoration: 'none',

                        }}

                    >

                        FILMBASE

                    </Typography>

                    <Box sx={{ flexGrow: 1, display: { xs: 'flex', md: 'none' } }}>

                        <IconButton

                            size="large"

                            aria-label="account of current user"

                            aria-controls="menu-appbar"

                            aria-haspopup="true"

                            onClick={handleOpenNavMenu}

                            color="inherit"

                        >

                            <MenuIcon />

                        </IconButton>

                        <Menu

                            id="menu-appbar"

                            anchorEl={anchorElNav}

                            anchorOrigin={{

                                vertical: 'bottom',

                                horizontal: 'left',

                            }}

                            keepMounted

                            transformOrigin={{

                                vertical: 'top',

                                horizontal: 'left',

                            }}

                            open={Boolean(anchorElNav)}

                            onClose={handleCloseNavMenu}

                            sx={{

                                display: { xs: 'block', md: 'none' },

                            }}

                        >

                            {pages.map((page) => (

                                <MenuItem key={page.name} onClick={handleCloseNavMenu}

                                >

                                    < Typography textAlign="center">

                                        {page.name}

                                    </Typography>

                                </MenuItem>

                            ))}

                        </Menu>

                    </Box>

                    <AdbIcon sx={{ display: { xs: 'flex', md: 'none' }, mr: 1 }} />

                    <Typography

                        variant="h5"

                        noWrap

                        component="a"

                        href=""

                        sx={{

                            mr: 2,

                            display: { xs: 'flex', md: 'none' },

                            flexGrow: 1,

                            fontFamily: 'monospace',

                            fontWeight: 700,

                            letterSpacing: '.3rem',

                            color: 'inherit',

                            textDecoration: 'none',

                        }}

                    >

                        LOGO

                    </Typography>

                    <Box sx={{ flexGrow: 1, display: { xs: 'none', md: 'flex' } }}>

                        {pages.map((page) => (

                            <Button

                                key={page.name}

                                onClick={handleCloseNavMenu}

                                href={page.hreff}

                                sx={{ my: 2, color: 'white', display: 'block' }}

                            >

                                {page.name}

                            </Button>

                        ))}

                    </Box>

                    {whouser !== "NULL" &&

                        <Box sx={{ flexGrow: 0 }}>

                            <Tooltip title="Open settings">

                                <IconButton onClick={handleOpenUserMenu} sx={{ p: 0 }}>

                                    <Avatar alt="Remy Sharp" src="/static/images/avatar/2.jpg" />

                                </IconButton>

                            </Tooltip>

                            <Menu

                                sx={{ mt: '45px' }}

                                id="menu-appbar"

                                anchorEl={anchorElUser}

                                anchorOrigin={{

                                    vertical: 'top',

                                    horizontal: 'right',

                                }}

                                keepMounted

                                transformOrigin={{

                                    vertical: 'top',

                                    horizontal: 'right',

                                }}

                                open={Boolean(anchorElUser)}

                                onClose={handleCloseUserMenu}

                            >

                                {settings.map((setting) => (

                                    <MenuItem key={setting.name} onClick={handleCloseUserMenu}

                                        href={setting.hreff}>

                                        <Button href={setting.hreff}>

                                            <Typography textAlign="center"

                                            >{setting.name}</Typography>

                                        </Button>

                                    </MenuItem>

                                ))}

                            </Menu>

                        </Box>}

                </Toolbar>

            </Container>

        </AppBar >

    );

}

Разберем одну из функции связи между клиентом и сервером:

async function proverka() {

        let status;

        const response = await fetch("http://localhost:8080/getprocessuser2");

        await response.text()

            .then((json) => {

                status = json;

                if (status === 'ADMIN') {

                    setWho("ADMIN");

                }

                if (status === 'USER') {

                    setWho("USER");

                }

                if (status === 'NULL') {

                    setWho("NULL");

                }

            });

    }

В данном случае, она является acync, что позволяет использовать await, который позволяет не выполнять дальнейший код, без получения полного ответа от сервера. Функция fetch имеет два параметра, адресс и параметры передачи. В данном случае мы ничего не передаем, а просто обращаемся по адресу, чтобы получить ответ. В этом примере мы запрашиваем у сервера роль пользователя, авторизованного сейчас в системе. Для получения ответа, мы снова прибегаем к response, с помощью которого отправляли запрос по адресу, и декодируем ответ в текст с помощью функции .text(). Далее, полученный ответ мы проверяем в соответствии с надобностью. Либо Администратор, либо обычный пользователь, либо еще не авторизованный в системе пользователь.

В React обычная функция не запускается при рендеринге страницы. В данном примере нужно было решить задачу не отображения страницы не авторизованному пользователю. Для этого в React существуют хуки. В проекте используются два вида хуков. И вот один из них:

React.useState(() => {

        proverka();

        console.log(whouser);

    })

useState – позволяет исполнить код содержимого до начала рендеринга самой страницы. Здесь мы внутри хука прописали стрелочную функцию, в которой вызвали обычную именную функцию, в которой получили данные о текущем авторизованном пользователе.

Далее рассмотрим страницу добавления нового фильма в базу:

import \* as React from 'react';

import Avatar from '@mui/material/Avatar';

import Button from '@mui/material/Button';

import CssBaseline from '@mui/material/CssBaseline';

import TextField from '@mui/material/TextField';

import Grid from '@mui/material/Grid';

import Box from '@mui/material/Box';

import Typography from '@mui/material/Typography';

import Container from '@mui/material/Container';

import AddBoxIcon from '@mui/icons-material/AddBox';

import { createTheme, ThemeProvider } from '@mui/material/styles';

const theme = createTheme();

export default function Addnewfilm() {

    const [whouser, setWho] = React.useState();

    React.useState(() => {

        proverka();

        console.log(whouser);

    })

    async function proverka() {

        let status;

        const response = await fetch("http://localhost:8080/getprocessuser2");

        await response.text()

            .then((json) => {

                status = json;

                if (status === 'NULL') {

                    setWho("NULL");

                    window.location.replace("/");

                }

            });

    }

    const handleSubmit = (event) => {

        event.preventDefault();

        const data = new FormData(event.currentTarget);

        if (data.get('headname').length < 1 || data.get('urlimage').length < 1

            || data.get('director').length < 1 || data.get('zhanr').length < 1

            || data.get('year').length < 1 || data.get('aboutis').length < 1) {

            alert("Заполните все поля!");

            return;

        }

        const models = setfrom(data);

        utopia(models);

    };

    async function utopia(models) {

        let sms;

        console.log(models);

        const response = await fetch("http://localhost:8080/addnewfilm", {

            method: "POST",

            headers: { "Content-Type": "application/json" },

            body: JSON.stringify(models)

        });

        await response.text()

            .then((json) => {

                sms = json;

            });

        alert(sms);

        var s = sms.split('');

        console.log(s);

        if (s[0] === '1') {

            window.location.replace('/allfilms');

        }

    };

    function setfrom(data) {

        let headname = data.get('headname');

        let urlimage = data.get('urlimage');

        let director = data.get('director');

        let year = data.get('year');

        let zhanr = data.get('zhanr');

        let aboutis = data.get('aboutis');

        let models = {

            headname, urlimage, director,

            year, zhanr, aboutis

        };

        return models;

    }

    return (

        <ThemeProvider theme={theme}>

            <Container component="main" maxWidth="xs">

                <CssBaseline />

                <Box

                    sx={{

                        marginTop: 8,

                        display: 'flex',

                        flexDirection: 'column',

                        alignItems: 'center',

                    }}

                >

                    <Avatar sx={{ m: 1, bgcolor: '#FF0000' }}>

                        <AddBoxIcon />

                    </Avatar>

                    <Typography component="h1" variant="h5">

                        Добавление Фильма

                    </Typography>

                    <Box component="form" noValidate onSubmit={handleSubmit} sx={{ mt: 3 }}>

                        <Grid container spacing={2}>

                            <Grid item xs={12}>

                                <TextField

                                    name="headname"

                                    required

                                    fullWidth

                                    id="headname"

                                    label="Название фильма"

                                    autoFocus

                                />

                            </Grid>

                            <Grid item xs={12}>

                                <TextField

                                    required

                                    fullWidth

                                    id="urlimage"

                                    label="Постер фильма"

                                    name="urlimage"

                                />

                            </Grid>

                            <Grid item xs={12}>

                                <TextField

                                    required

                                    fullWidth

                                    id="director"

                                    label="Режисер"

                                    name="director"

                                    autoComplete="director"

                                />

                            </Grid>

                            <Grid item xs={12}>

                                <TextField

                                    required

                                    fullWidth

                                    id="zhanr"

                                    label="Жанр"

                                    name="zhanr"

                                    autoComplete="zhanr"

                                />

                            </Grid>

                            <Grid item xs={12}>

                                <TextField

                                    required

                                    fullWidth

                                    id="aboutis"

                                    label="Описание"

                                    name="aboutis"

                                    autoComplete="aboutis"

                                />

                            </Grid>

                            <Grid item xs={12}>

                                <TextField

                                    required

                                    fullWidth

                                    id="year"

                                    label="Год"

                                    name="year"

                                    autoComplete="year"

                                />

                            </Grid>

                        </Grid>

                        <Button

                            type="submit"

                            fullWidth

                            variant="contained"

                            sx={{ mt: 3, mb: 2 }}

                        >

                            Добавить

                        </Button>

                    </Box>

                </Box>

            </Container>

        </ThemeProvider>

    );

}

На этой странице требуется как раз отправлять данные на сервер:

const response = await fetch("http://localhost:8080/addnewfilm", {

            method: "POST",

            headers: { "Content-Type": "application/json" },

            body: JSON.stringify(models)

        });

В параметрах отправки указывается стратегия запроса, у нас это POST, соответственно и на стороне сервера у нас Post. В заголовках указываем, что переданная информация будет в формате json. В параметре body отправляем объект на сервер в формате JSON. Важно, чтобы названия полей, в которых передается информация были такими же, как в объекте, который принимает их на стороне сервера, иначе данный просто исчезнут.

await response.text()

            .then((json) => {

                sms = json;

            });

        alert(sms);

        var s = sms.split('');

        console.log(s);

        if (s[0] === '1') {

            window.location.replace('/allfilms');

        }

В примере выше получаем ответ от сервера, выводим полученный текст на экран, после чего разбираем его на символы, в которых считываем первый, и если он соответствует условию, то мы перенаправляемся на страницу со всеми фильмами.

И здесь очень важное отличие: мы используем перерелокацию в браузере, что полностью перерендерит странницу. Есть также второй способ перенаправления на другой адрес, рассмотрим его на примере кода страницы авторизации:

import React from 'react'

import Button from '@mui/material/Button';

import { useNavigate } from 'react-router-dom';

export default function Error() {

    const [whouser, setWho] = React.useState();

    React.useState(() => {

        proverka();

        console.log(whouser);

    })

    async function proverka() {

        let status;

        const response = await fetch("http://localhost:8080/getprocessuser2");

        await response.text()

            .then((json) => {

                status = json;

                if (status === 'ADMIN') {

                    setWho("ADMIN");

                }

                if (status === 'USER') {

                    setWho("USER");

                }

                if (status !== 'NULL') {

                    setWho("NULL");

                    window.location.replace("/allfilms");

                }

            });

    }

    let navigate = useNavigate();

    const handleClickReg = async (e) => {

        e.preventDefault()

        navigate("/signin");

    }

    const handleClickEnter = async (e) => {

        e.preventDefault()

        navigate("/signup");

    }

    return (

        <div >

            <Button

                variant="contained"

                onClick={handleClickReg}

                color="success"

                sx={{ mt: 3, mb: 2, ml: 10 }}>

                РЕГИСТРАЦИЯ

            </Button>

            <Button

                variant="outlined"

                onClick={handleClickEnter}

                color="error"

                sx={{ mt: 3, mb: 2, ml: 10 }}>

                ВХОД

            </Button>

        </div>

    )

}

Здесь используются два вида перехода на другую страницу: через window, здесь используется после проверки авторизации пользователя. Если пользователь уже авторизован в системе, то находиться на этой странице ему нет смысла и мы переходим на страницу со всеми фильмами с перерендерингом.

const handleClickEnter = async (e) => {

        e.preventDefault()

        navigate("/signup");

    }

А здесь мы используем хук useNavigate. В частности, создан ее экземпляр. Так как мы используем роутинг, это те теги, которые мы прописывали в файле App.js , то мы можем переходить по страницам сайта, меняя только Outlet область, а шапка будет оставаться неизменной. Данная зависимость прописана в следующем файле:

import React, { Component } from 'react'

import { Outlet } from 'react-router-dom'

import Appbar from './AccountMenu'

export default class mainer extends Component {

    render() {

        return (

            <>

                <Appbar />

                <Outlet />

            </>

        )

    }

}

В роутинге файл mainer занимает самую высокую приоритетную позицию, и имеет адресс обращения “/”. Данный класс наследуются от Component, тем самым позволяет указать элементы которые будут рендериться, и которые не будут рендериться. Чтобы указать перерендериваемую часть, помечаем ее как <Outlet/>. <Appbar/> в котором располагается шапка сайта, изменятся вслед за измененями оутлета не будет.

Последний рассматриваемый клиентской части файл будет страница отображения всех фильм проекта-сервиса.

import \* as React from 'react';

import Card from '@mui/material/Card';

import CardContent from '@mui/material/CardContent';

import CardMedia from '@mui/material/CardMedia';

import Typography from '@mui/material/Typography';

import { CardActionArea } from '@mui/material';

import Container from '@mui/material/Container';

import Grid from '@mui/material/Grid';

import './Comp.css';

import Buttonn from '@mui/material/Button';

import EditIcon from '@mui/icons-material/Edit';

import FormControl from '@mui/material/FormControl';

import Input from '@mui/material/Input';

import Rating from '@mui/material/Rating';

import InputLabel from '@mui/material/InputLabel';

import { useNavigate } from 'react-router-dom';

export default function Allfilms() {

    const navigate = useNavigate();

    const [whouser, setWho] = React.useState();

    async function addonmyfilms(myfilm) {

        console.log(myfilm)

        let accounts;

        const response = await fetch("http://localhost:8080/addmyfilm/", {

            mode: 'cors',

            method: "POST",

            headers: { "Content-Type": "application/json" },

            body: JSON.stringify(myfilm)

        })

        response.text()

            .then((json) => {

                accounts = json;

                if (accounts === 'ERROR') {

                    alert("Данный фильм уже есть в вашей коллекции");

                }

                if (accounts === 'GOOD') {

                    alert("Фильм добавлен в вашу коллекцию");

                }

            });

    }

    const [item, setItem] = React.useState([]);

    React.useState(() => {

        fetch("http://localhost:8080/allfilms")

            .then(res => res.json())

            .then((result) => {

                setItem(result);

                console.log(result);

            })

        console.log(typeof (item));

        proverka();

        console.log(whouser);

    })

    async function editfilm(myfilm) {

        navigate("/editfilm", { state: myfilm });

    }

    async function proverka() {

        let status;

        const response = await fetch("http://localhost:8080/getprocessuser2");

        await response.text()

            .then((json) => {

                status = json;

                if (status === 'ADMIN') {

                    setWho("ADMIN");

                }

                if (status === 'USER') {

                    setWho("USER");

                }

                if (status === 'NULL') {

                    setWho("NULL");

                }

            });

    }

    const handleChange = (event) => {

        let headname = event.target.value;

        const idd = { headname }

        console.log(idd);

        fetch("http://localhost:8080/allfilmssort", {

            method: "POST",

            headers: { "Content-Type": "application/json" },

            body: JSON.stringify(idd)

        })

            .then(res => res.json())

            .then((result) => {

                setItem(result);

            })

        console.log(typeof (item));

    };

    return (

        <>

            <div className='forsearcharea'>

                <FormControl variant="standard">

                    <InputLabel htmlFor="component-simple">ПОИСК...</InputLabel>

                    <Input id="idmarka" onChange={handleChange} />

                </FormControl>

            </div>

            <Container component="main" maxWidth="100%">

                <Grid container spacing={1}>

                    {item.map(item => (

                        <Card sx={{ minWidth: 330, maxWidth: 330, mt: 3 }} xs={3} className='Cardstyle' key={item.idcinema}

                        >

                            <CardActionArea>

                                <CardMedia

                                    component="img"

                                    height="250"

                                    width="250"

                                    image={item.urlimage}

                                    alt="green iguana"

                                />

                                <CardContent >

                                    <Typography gutterBottom variant="h5" component="div">

                                        {item.headname}

                                    </Typography>

                                    {whouser === "ADMIN" &&

                                        <Buttonn sx={{ mt: 0, mb: 1, width: 150, height: 25 }} variant="contained"

                                            onClick={() => { editfilm(item) }}  >

                                            Редактировать

                                        </Buttonn>

                                    }

                                    <Typography variant="body2" color="text.secondary">

                                        {item.director} - {item.year}

                                    </Typography>

                                    <Typography variant="body2" color="text.secondary">

                                        Жанр :  {item.zhanr}

                                    </Typography>

                                    <Typography variant="body2" color="text.secondary">

                                        Оценок фильма на портале :  {item.marks}

                                    </Typography>

                                    <Rating name="half-rating-read" defaultValue={item.rating} precision={0.5} readOnly />

                                    {whouser !== "NULL" &&

                                        <Buttonn sx={{ mt: 1, width: 300, height: 35 }} variant="contained"

                                            onClick={() => { addonmyfilms(item) }} endIcon={<EditIcon />} >

                                            Добавить в мои фильмы

                                        </Buttonn>

                                    }

                                </CardContent>

                            </CardActionArea>

                        </Card>

                    ))}

                </Grid>

            </Container>

        </>

    )

}

Еще до рендеринга страницы получаем данные о всех фильмах в базе:

React.useState(() => {

        fetch("http://localhost:8080/allfilms")

            .then(res => res.json())

            .then((result) => {

                setItem(result);

                console.log(result);

            })

        console.log(typeof (item));

        proverka();

        console.log(whouser);

    })

Полученный ответ записывает в переменную Item.

Далее, при рендеринге, мы записываем каждый отдельный элемент из всей полученной коллекции в map:

<Grid container spacing={1}>

                    {item.map(item => (

                        <Card sx={{ minWidth: 330, maxWidth: 330, mt: 3 }} xs={3} className='Cardstyle' key={item.idcinema}

Далее изымаем информацию из каждого отдельного фильма, и отображаем ее. Также можно создавать условия отображения внутри функции render():

{whouser === "ADMIN" &&

 <Buttonn sx={{ mt: 0, mb: 1, width: 150, height: 25 }}

variant="contained"

  onClick={() => { editfilm(item) }}  >

  Редактировать

 </Buttonn>

}

В данном случае, если роль авторизованного пользователя админовская, то отображаем кнопку редактирования фильма.

Таким образом создан сервис по коллекции фильмов. В котором Гость может зарегистрироваться, добавить новый фильм в базу, добавить его в свою коллекцию.

Создано 4 связанных между собой сущности:

Отличительной особенностью администратора является возможность изменять добавленную информацию о фильмах пользователями. Из иных функций отметим следующие возможности:

1. Регистрация пользователя
2. Авторизация пользователя
3. Просмотр коллекции фильмов сервиса
4. Фильтрация фильмов сервиса
5. Добавление фильма в свою собственную коллекцию
6. Добавление нового фильма в базу
7. Редактирование фильма из базы данных
8. Изменение настроек приватности пользователя
9. Изменение пароля пользователя
10. Просмотр фильмов из собственной коллекции пользователя
11. Просмотр детальной информации о фильме из своей коллекции
12. Фильтрация фильмов из своей коллекции
13. Выставление оценки фильму из своей коллекции
14. Выставление статуса просмотра фильму из своей коллекции
15. Удаление фильма из своей коллекции
16. Поиск новых друзей-пользователей
17. Фильтрация в поиске друзей
18. Просмотр своих друзей
19. Фильтрация своих друзей
20. Просмотр коллекции фильмов своих друзей
21. Фильтрация коллекций фильмов друзей
22. Удаление из друзей

Полный код проекта по ссылке:

<https://github.com/vadyaooo/FullstackAppFilmBase>

Скриншоты страниц на клиентской части:

