# Министерство образования Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**КУРСОВОЙ ПРОЕКТ**

по дисциплине **«Объектно-ориентированное программирование»**

на тему **«**Website for a cinema with ticket booking subsystem (Сайт-визитка для кинотеатра с подсистемой резервирования билетов)**»**

Выполнил:

студент группы 10702217 Д.Д. Аристова

Руководитель: В.В. Иванченко

Минск 2019

Содержание

[Министерство образования Республики Беларусь 1](#_Toc8420325)

[Введение 3](#_Toc8420326)

[1. Техническое задание на предмет разработки 4](#_Toc8420327)

[2. Обоснование используемых языков и технологий 7](#_Toc8420328)

[3. Моделирование и проектирование программного обеспечения 9](#_Toc8420329)

[4. Реализация программного обеспечения 12](#_Toc8420330)

[5. Развёртывание и тестирование программного обеспечения 13](#_Toc8420331)

[Заключение 14](#_Toc8420332)

[Список использованных источников 15](#_Toc8420333)

[Приложение 16](#_Toc8420334)

[Листинг: 21](#_Toc8420335)

# **Введение**

Интернет является крупнейшей глобальной информационной и коммуникационной сетью мира, число пользователей которой стремительно растет. Созданная в начале 90-х годов XX века служба World Wide Web (WWW) стала основным толчком к расширению глобальной компьютерной сети. Именно благодаря WWW в ней участвуют не только ученые, специалисты в области программирования, но и огромное число обычных пользователей. Интернет стал мощным средством ведения бизнеса, и большинство коммерческих фирм уже не может в своей деятельности обходиться без использования его возможностей.

В Европе, например, сложно найти даже небольшую компанию, у которой не было бы собственного сайта.

В данном курсовом проекте будет реализованн сайт-визитка кинотеатра. Но что же такое сайт-визитка?

Это как правило, сайты, которые имеют простую структуру, содержат до 10 страниц, предоставляющих общую информацию о фирме и ее продукции. Обычно ссылки на страницах такого сайта позволяют с каждой страницы попасть на любую другую. Такие сайты могут быть выполнены целиком на flash и являться по сути презентацией фирмы в Интернете. Сайт-визитка вполне себя оправдывает в том случае, если продукция фирмы не нуждается в полном и серьезном представлении в Интернете, основная же задача такого сайта заключается в предоставлении пользователю краткого описания фирмы и ее контактной информации.

# **Техническое задание на предмет разработки**

Клиентская часть приложения:

Клиент:

* Регистрация пользователя – кнопка Sign Up
  + После нажатия кнопки пользователь видит модальное диалоговое окно с заданными полями для регистрации
  + Поля для регистрации:
    - Username
    - Name
    - Password.
  + Все поля для регистрации должны быть заполнены;
  + Пользователь должен ввести более 4 символов.
  + При успешной регистрации вывести message box, что пользователь зарегистрировался.
* Авторизация пользователя – кнопка Sign In.
  + После нажатия кнопки пользователь видит модальное диалоговое окно с заданными полями для авторизации.
  + Поля для авторизации: username, password.
  + Все поля должны быть заполнены
  + При успешной авторизации вывести message box, что пользователь зарегистрировался.
* Выбор фильма.
  + Пользователь видит список всех доступных фильмов
  + Каждый элемент списка, фильм, является раскрывающимся списком
  + При нажатии на элемент списка, фильм, можно увидеть описание фильма, список сеансов.
  + если пользователь сделал заказ на этот сеанс, то появляется поле «Price of your order» и указывается цена заказнного билета.
* Выбор сеанса.
  + Информация о сеансе:
    - день/ месяц/ год показа фильма
    - Начало сеанса
    - Конец сеанса
    - Количество мест
    - Цена билета
  + Кнопка заказа билета на сеанс «Reserve»
  + Если пользователь уже сделал заказ на данный сеанс, то вместо кнопки заказа билета, появляется кнопка «Change reserved places», «Cancel Reservation» и дополнительное поле показывающее сумму заказа
* Бронирование билета– кнопка Reservation session или « Change reserved places ».
  + При нажатии кнопки появляется Reservation/ Change reserved places модальное диалоговое окно в котором пользователь видит:
    - Поле для ввода ряда
    - Поле ввода места
    - Графическое представление зала
    - Кнопка для добавления мест (в виде «+»)
    - Кнопка добавления резервировании выбранных мест - Reserve
    - Кнопка отмены Cancel
    - Список мест, забронированные пользователем
      * Можно удалить выбранное место, нажав на крестик
* Отмена бронирования билета.
  + Кнопка «Cancel Reservation»
* Просмотр списка доступных фильмов и сеансов.

Администратор:

* Авторизация (такая же как и у клиента).
* Добавление новых фильмов.
  + Кнопка – Add film
  + При нажатии на копку появляется модальное окно в котором есть два поля для ввода данных( имя фильма/ описание) и две кноки (добавить фильм и отмена)
* Добавлении сеанса
  + Кнопка – Add session
  + При нажатии на копку появляется модальное окно в котором есть пять полей для ввода данных( время начала сенса, продолжительность сеанса, количество рядов, количество мест, цена билета ) и две кноки (добавить сеанс и отмена)
* Удаление существующего фильма.
  + Кнопка – Remove film
* Выбор фильма (как и у клиента).
* Выбор сеанса (как у клиента).
* Бронирование сеанса (как у клиента).
* Отмена бронирования сеанса (как у клиента).
* Просмотр списка созданных фильмов и сеансов .
* Поле прибыли от фильма
* Поле прибыли от сеанса

Серверная часть приложения:

* Хранение фильмов, сеансов, размер зала кинотеатра в базе данных.
  + Использовать MongoDB
* Добавление новых фильмов, сеансов, размер зала кинотеатра в базу данных.
* Хранение зарегистрированных пользователей в базе данных.
* Добавление новых пользователей в базу данных.
* Авторизация пользователя.
* Если пользователь является администратором, то он может пользоваться дополнительными функциями (добавление/ удаление фильмов или сеансов).

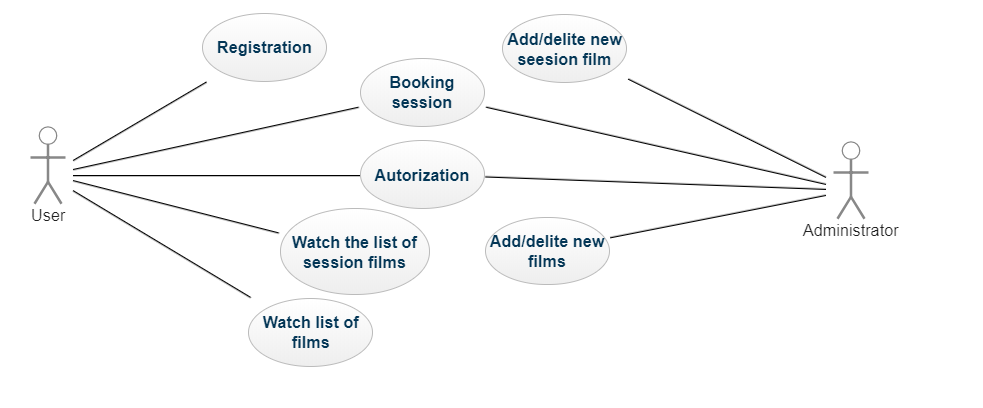


Диаграмма 1- use-case диаграмма

# **Обоснование используемых языков и технологий**

В данном курсовом проекте backend реализован на TypeScript и фреймворка Koa, а frontend с использованием фреймворка Angular.

TypeScript один из языков программирования, который используется в веб-программировании.

Идея фреймворка Koa идеально ложиться на парадигму микросервисов. Ядро фреймворка минималистично, код middleware для koa читается и понимается очень легко, что очень важно для командной разработки.

Среди основных ***особенностей языка TypeScript*** выделим следующие:

1. язык TypeScript строго типизирован;
2. код на TypeScript можно компилировать в JavaScript;
3. язык обратно совместим с JavaScript и даже, если разработчик будет писать смешанный код, то это также будет валидно.

Почему стоит выбрать ***TypeScript*** ?

* Возможность жестко описывать каждый элемент приложения. Веб-специалисту нужно четко продумывать логику приложения вплоть до момента реализации. Нет возможности изменить одну часть проекта, при этом сломав другую часть.
* Есть возможность описать область видимости свойств класса.
* Нужно писать меньше тестов из-за жесткой архитектуры.
* Любой некомпилируемый код нельзя будет закоммитить с помощью специальной настройки.
* Некоторые из ошибок форматирования кода исключены из-за жесткого формата конструкций языка программирования.

TypeScript — это расширение JavaScript, дающее пользователю возможность работать со статической типизацией и надежным инструментарием. При помощи TypeScript можно исполнять множество функций: использовать уже написанный код JavaScript, включать популярные библиотеки, вызывать TypeScript-код, который сгенерирован из других JavaScript.

В последнее врямя данный язык набрал популярность и стал часто использоваться из-за того, что он полностью совмести с Angular 7, который, кстати, написан на TypeScript.

Angular предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация и так далее.

Преимущества Angular:

* Статическая типизация
* Использование ООП в связи с тем, что использует TypeScript
* Использование Angular material в качестве стилизованных компонентов.

С исходным кодом TypeScript все желающие могут ознакомиться на ресурсе github. Данный язык имеет большие перспективы развития в ближайшем будущем, и предоставляет уже много возможностей для работы с Javascript. Важно также отметить, что TypeScrip будет легче выучить веб-специалистам, которые ранее работали с C#, Java и другими языками с классическим ООП синтаксисом.

Для модульного тестирования использовался Jest. Jest — это программа для запуска тестов, разработанная Facebook, направленная на предоставление функционального решения для модульного тестирования. Jest работает с проектами написанные на: JavaScript, Babel, TypeScrip, Node.js, React, Angular и Vue.js.

Для хранения данных была выбранна база данных: MongoDB.  
MongoDB — документоориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++.

Преимущества MongoDB:

* Отсутствие схемы
* Данная БД основана на коллекциях различных документов. Количество полей, содержание и размер этих документов может отличаться. Т.е. различные сущности не должны быть идентичны по структуре.
* Крайне понятная структура каждого объекта.
* Легко масштабируется
* Для хранения используемых в данный момент данных используется внутренняя память, что позволяет получать более быстрый доступ.
* Данные хранятся в виде JSON документов
* MongoDB поддерживает динамические запросы документов (document-based query)
* Отсутствие сложных JOIN запросов
* Нет необходимости маппинга объектов приложения в объекты БД

Для работы с MongoDB используется Mongoode.

Mongoose представляет специальную ODM-библиотеку (Object Data Modelling) для работы с MongoDB, которая позволяет сопоставлять объекты классов и документы коллекций из базы данных.Mongoose предоставляет простое решение на основе схем для моделирования данных вашего приложения. Он включает встроенное приведение типов, проверку, построение запросов, хуки бизнес-логики и многое другое.

# Моделирование и проектирование программного обеспечения

Для хранения данных используется база данных: MongoDB.

Mongoose позволяет определять объекты со строго-типизированной схемой, соответствующей документу MongoDB

В базе данных хранятся такие схемы как User, Session, Film, Reservation, Place.  
Перейдем к более детальному описанию.

В User есть свойства: username, name, password, role. Нужна для хранения информации зарегистрировавшихся пользователей.

В Session есть свойства: filmName, startTime, endTime, rows, columns, reservation, ticketPrice. Реализована для хранения сеансов на фильм.

В Film есть свойства: name, description. Нужна для хранения фильмов

В Reservation есть свойства: userId, sessionId, places.Нужна для хранения информации о зарезервированных пользователем на сеанс мест.

В Place есть свойства:row, column. Нужна для хранения резервации конкретного места.



Диаграмма 1 - ER– диаграмма

Backend сайта реализован при помощи языка программирования TypeScript и фреймворка Koa. Основнми сущностями являются: User, Session, Reservation, Place, Film. Для реализации функционала данных моделек реализованы сервисы UserService, SessionService, ReservationService, FilmService.



Диаграмма 2 - UML – диаграмма классов (model и service)

Для обработки и выполнения запросов пользователя реализован роутинг при помощи koa-router. Самым главным является файл index.ts в нем прописаны все роуты, которые может использовать пользователь. Для моего проекта реализовано 4 роута: ‘/auth’, ‘/film’, ‘/session’, ‘/reservation’. И у каждого роута определены свои post и get методы.

Для роута ‘/auth’, отвечающий за регистрацию и авторизацию пользователей определенно 2 post метода: ‘/signIn’, ‘/signUp’.

Для роута ‘/film’, отвечающий за добавление, удаление и взятие из базы данных список фильмов, определенно 2 post метода: ‘/add’, ‘/remove’ и get метод ‘/all’.

Для роута ‘/session’, отвечающий за добавление, удаление, взятие из базы данных список сеансов на нужный фильм и взятие одного конкретного сеанса на фильм, определенно 2 post метода: ‘/add’, ‘/remove’ и 2 get метода ‘/all/:filmName’, ‘/one/:sessionId’.

Для роута ‘/reservation’, отвечающий за добавление, удаление, взятие из базы данных одно зарезервированное место на сеанс фильма, определенно 3 post метода: ‘/add’, ‘/remove’, ‘/one’.



Диаграмма 3 - UML – диаграмма классов (routes)

Для того, чтобы пользователь получал ответ на свои запросы в той форме, которая нужна программисту, созданы классы:

RestResponce - в нём есть поле success(по умолчанию false) и message, чтобы «общаться» с пользователем и является родителем для следующих описанных классов.

SignInResponce - в нём есть поле token(дается при авторизации пользователя) и user.

FilmsResponce -содержит одно поле films, чтобы получать список фильмов

SessionListResponce - содержит одно поле sessions, чтобы получать список сенансов.

SessionResponce - содержит одно поле session.

ReservationResponce - содержит одно поле reservation.



Диаграмма 4 - UML – диаграмма классов (responce)

# Реализация программного обеспечения

Клиентская часть курсового проекта (frontend) реализована при помощи фреймворка Angular. Клиентска часть предоставленна самым главным компонентом Application В котором есть компонент Navigation. В Navigation реализован роутинг для администратора(‘/admin’) и простого клиента (‘/home’). При чём, если пользователь являющийся адмимнстратором имеет доступ и к ‘/home’. На каждой страничке мы можем наблюдать список существующих фильмов. Функционал на страничке администратора: добавление/ удаление фильмов или сеанса. Функционал на страничке пользователя: резервирование/отмена резервирования мест на сеанс. Весь функционал реализован при помощи компонентов: Session- create-dialog, film-create-dialog, reservation-dialog в виде диалогового окна. Авторизация и регистрация осуществлены при помощи компонентов: SignIn-dialog, signUp-dialog, так же в виде диалогового окна.



Диаграмма 5 – диаграмма компонентов клиентской части

Серверная часть веб-сайта(backend) реализована при помощи фреймворка Koa и языка программирования TypeScript. Доступ к нужному функционалу происходит при прохождении запроса через middleware и rout. Они обрабатывают http запрос и выдают нужный для пользователя ответ.



Диаграмма 6 – диаграмма компонентов серверной части

# Развёртывание и тестирование программного обеспечения

Angular, как и MongoDB является кроссплатформенной программой, поэтому операционна система не является главным аспектом для развертывания приложения.



Диаграмма 6 - Структура программного развёртывания системы

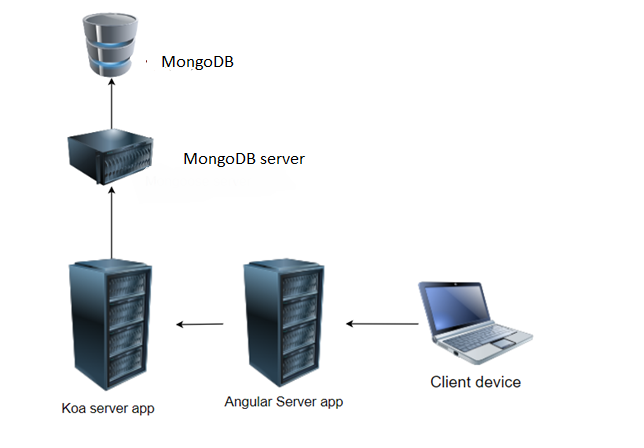


Диаграмма 7 - Структура аппаратного развёртывания приложения

Тестирование реализовано с использованием jest.

Тестирование сервиса и валидатора для фильмов. (файлы film.service.ts, film.validators.ts)

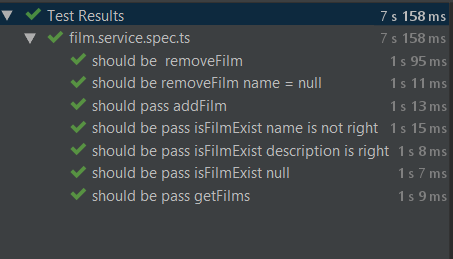


Рис 1- результат тестирования film.service.spec.ts

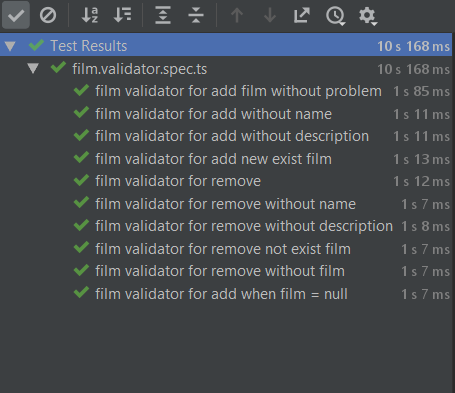


Рис 2- результат тестирования film.validators.spec.ts

Тестирование сервиса и валидатора для сенасов. (файлы session.service.ts, session.validators.ts

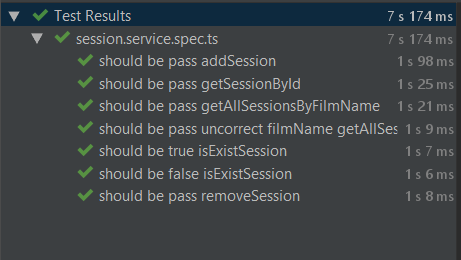


Рис 3- результат тестирования session.service.spec.ts

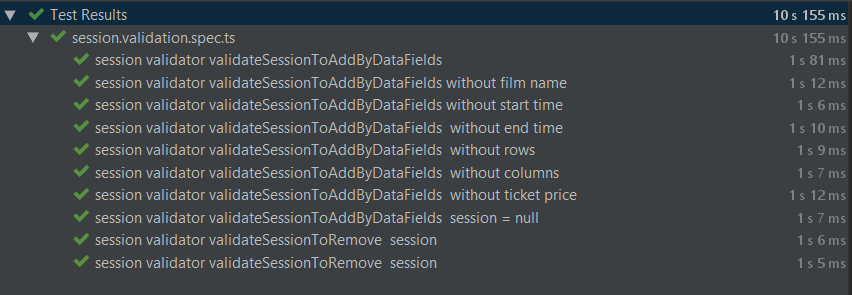


Рис 4- результат тестирования session.validators.spec.ts

Тестирование сервиса и валидатора для резервации (файлы reservation.service.ts, reservation.validators.ts

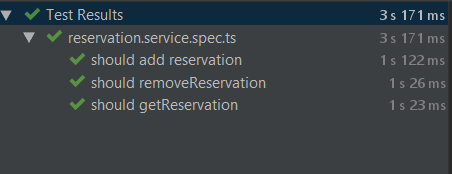


Рис 5- результат тестирования reservation.service.spec.ts

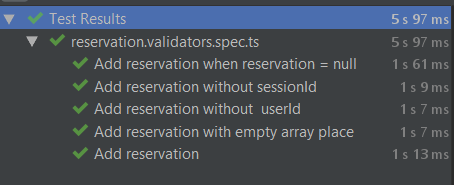


Рис 6- результат тестирования reservation.validators.spec.ts

Тестирование сервиса и валидатора для пользователей. (файлы user.service.ts, user.validators.ts

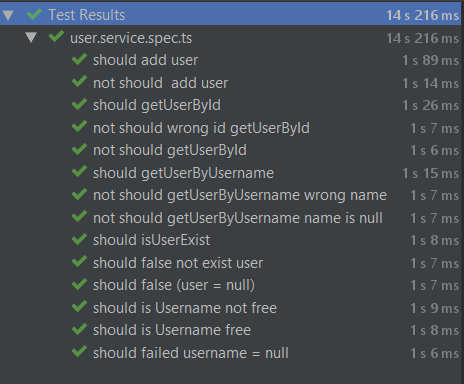


Рис 7- результат тестирования user.service.spec.ts

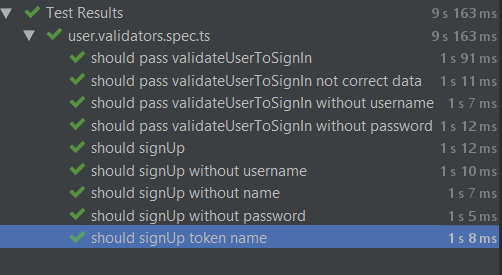


Рис 8- результат тестирования user.validators.spec.ts

Тестирование доступа к роутам:

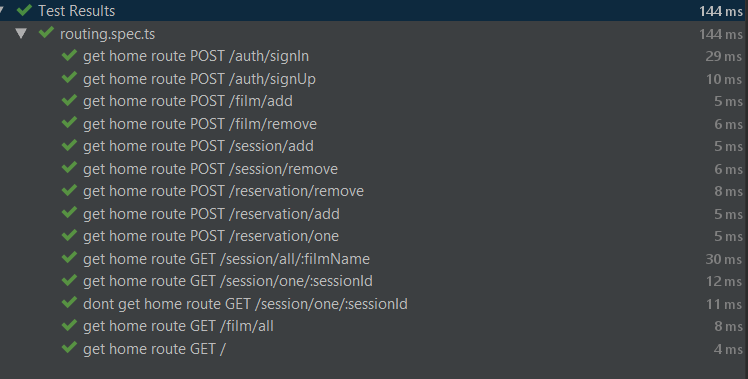


Рис 9- результат тестирования rout.spec.ts

# Заключение

В данном курсовом проекте был реализован сайт-визитка с подсистемой бронирования мест. Backend реализован на языке программирования TypeScript и фреймворка Koa.Frontend реализован на фреймворке Angular. Для хранения данных и занесения информации использовалась база данных: MongoDB. Журналирование реализовано при помощи стандартной библиотеки typescript-logging в стиле Log4j. Тестирование реализованно на Jest.

# Список использованных источников

1. Angular – необходимый инструмент для профессионального веб-разработчика [Электронный ресурс]. – Электронные данные.- Режим доступа: <https://webformyself.com/angular-neobxodimyj-instrument-dlya-professionalnogo-veb-razrabotchika/>
2. Введение в Angular [Электронный ресурс]. – Электронные данные.- Режим доступа: <https://metanit.com/web/angular2/1.1.php>
3. Koajs 2.0: новое поколение фреймворка нового поколения [Электронный ресурс]. – Электронные данные.- Режим доступа: <https://habr.com/ru/post/301126/>
4. Jest Methods [Электронный ресурс]. – Электронные данные.- Режим доступа: <https://jestjs.io/>
5. Выборка из БД [Электронный ресурс]. – Электронные данные.- Режим доступа: <https://metanit.com/nosql/mongodb/2.4.php>
6. Работа с типами данных TypeScript [Электронный ресурс]. – Электронные данные.- Режим доступа: <https://metanit.com/web/typescript/2.4.php>
7. Объектно-ориентированное программирование TypeScript [Электронный ресурс]. – Электронные данные.- Режим доступа: https://metanit.com/web/typescript/3.4.php

# Приложение

Скриншоты работы веб-сайта:

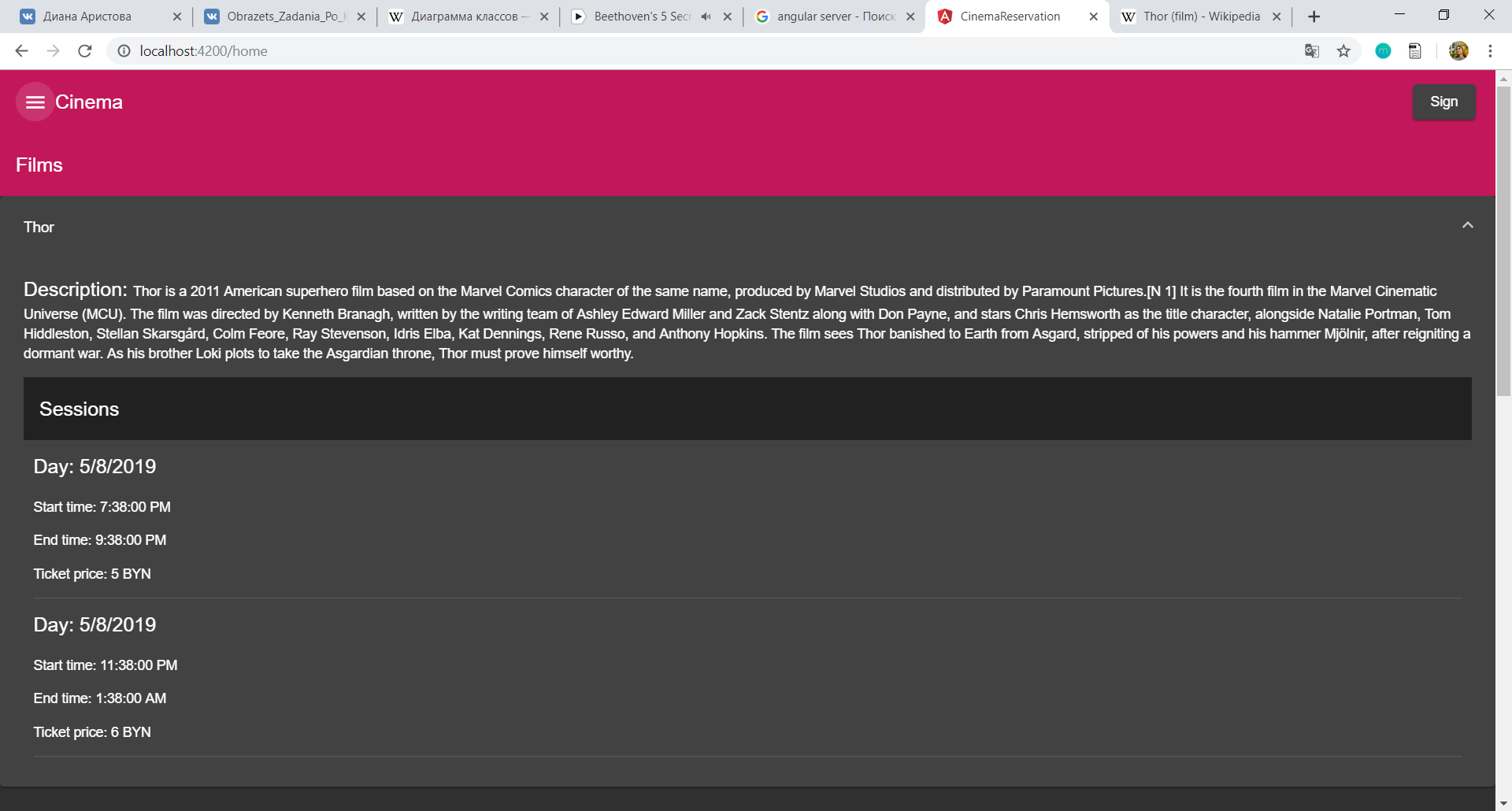


Рис 1 – вид сайта, если пользователь не авторизован

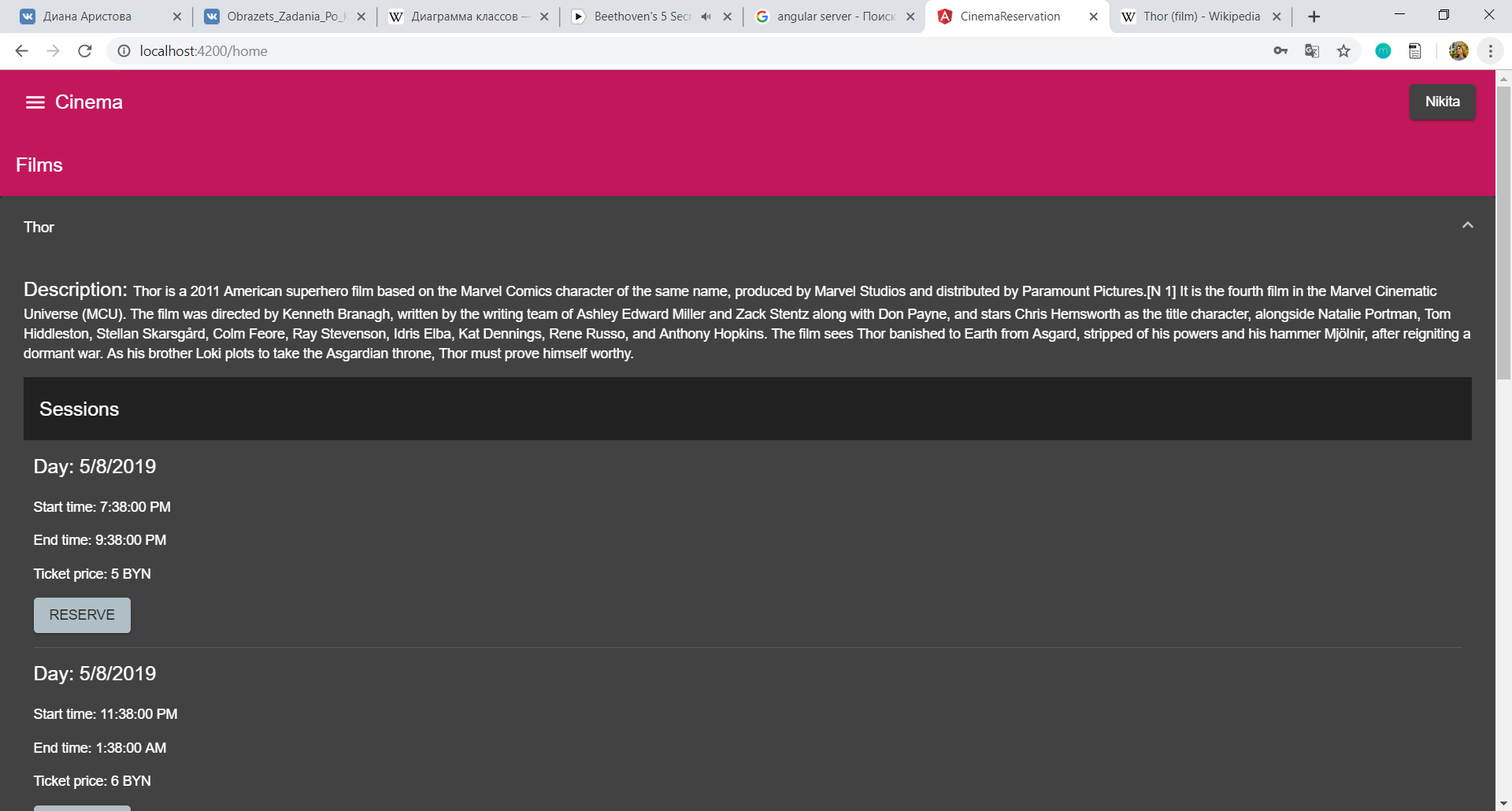


Рис 2 – вид сайта, если пользователь авторизован и не является администратором

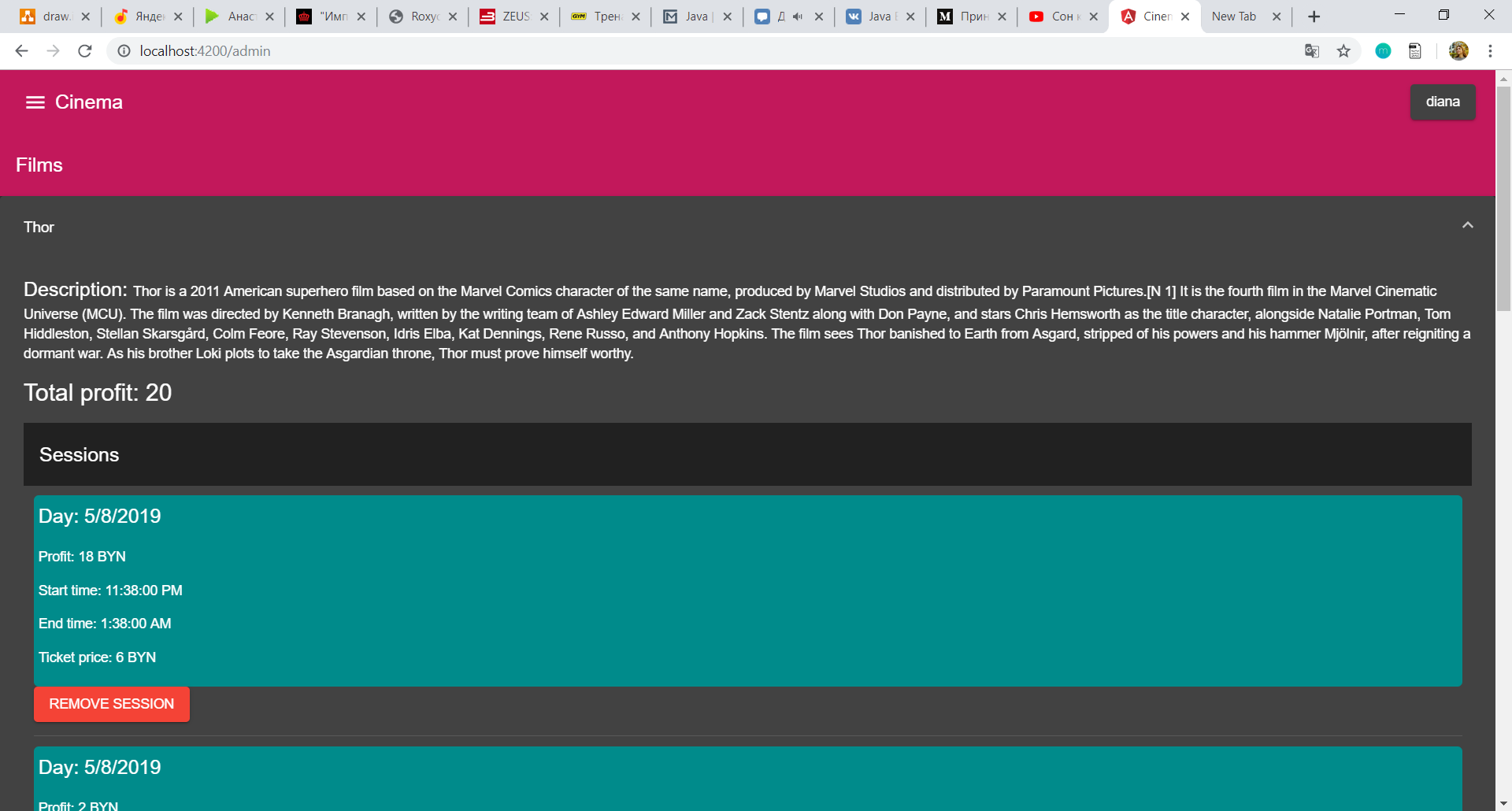


Рис 3 – дополнительная страница доступная только администратору

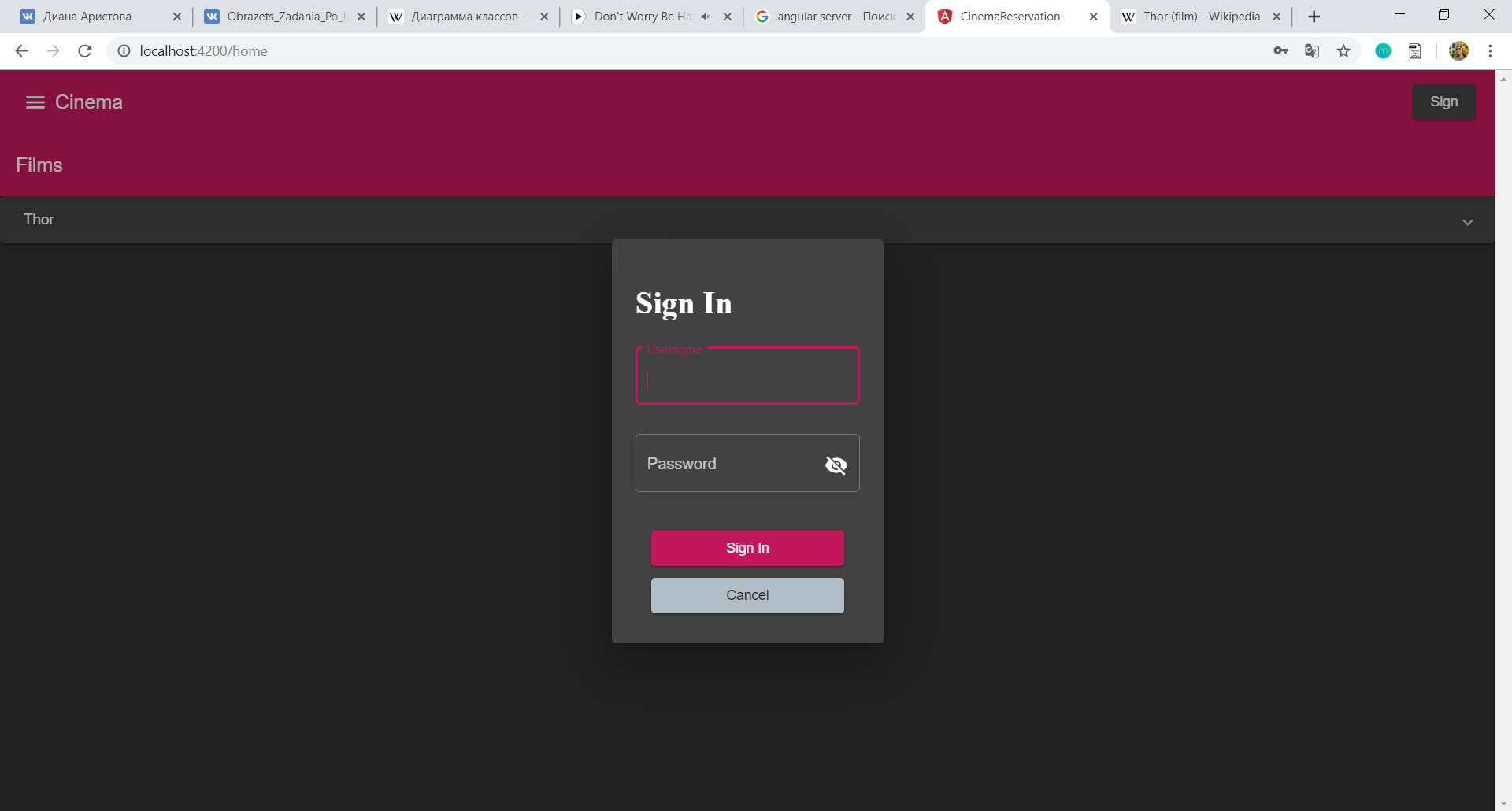


Рис 4 – диалоговое окно для авторизации пользователя

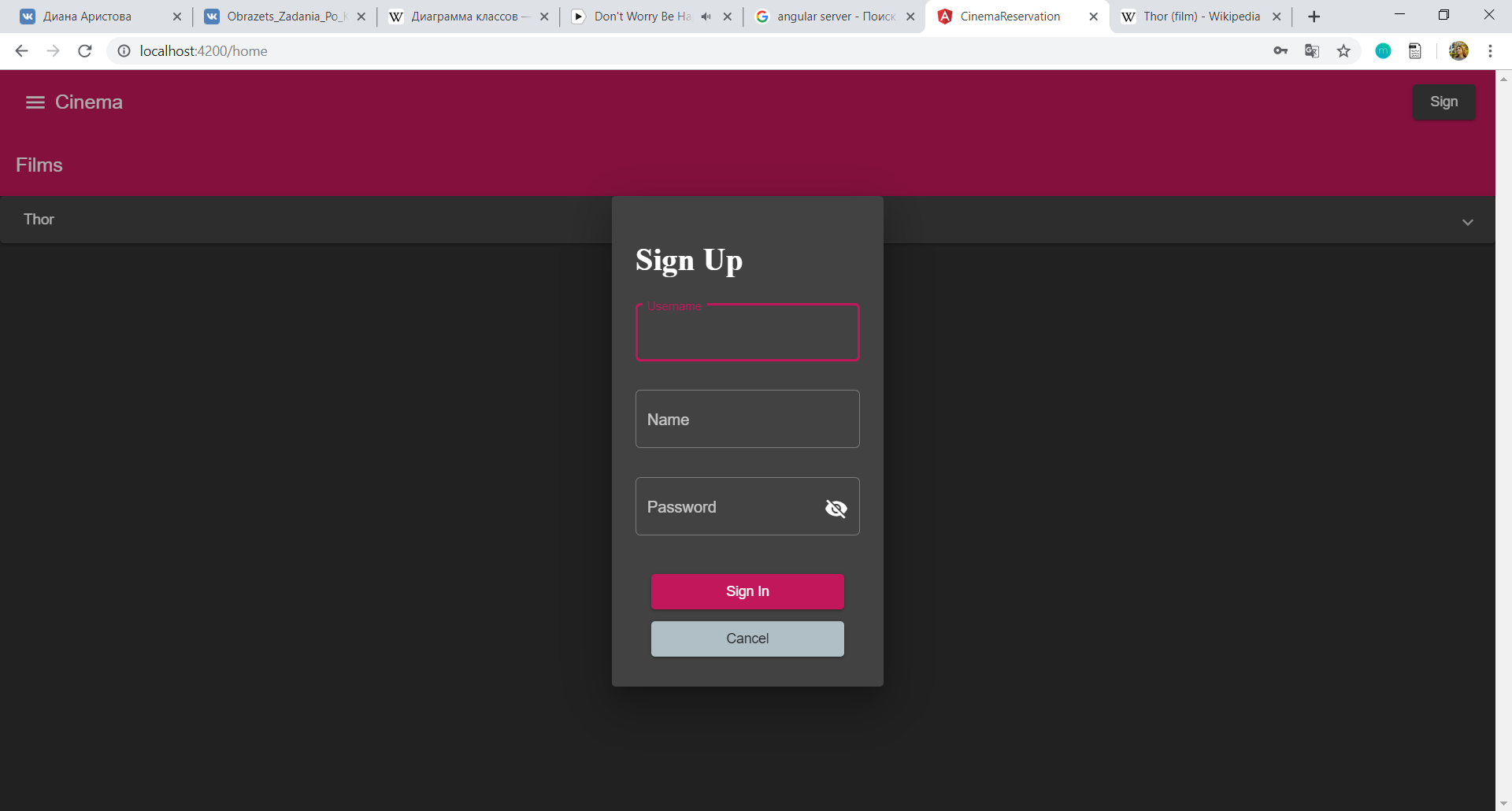


Рис 5 – диалоговое окно для регистрации пользователя

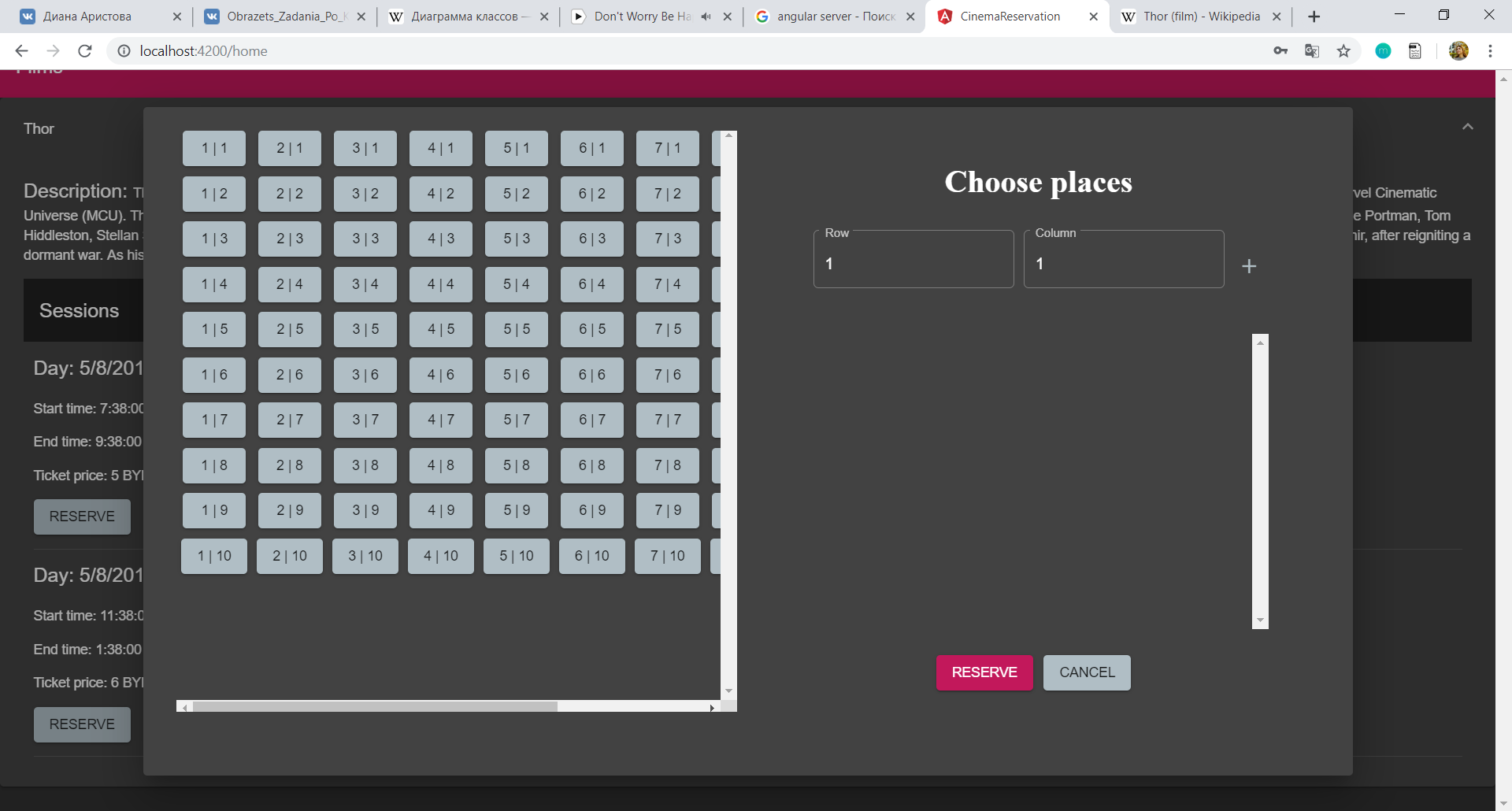


Рис 6 – диалоговое окно для резервирования мест

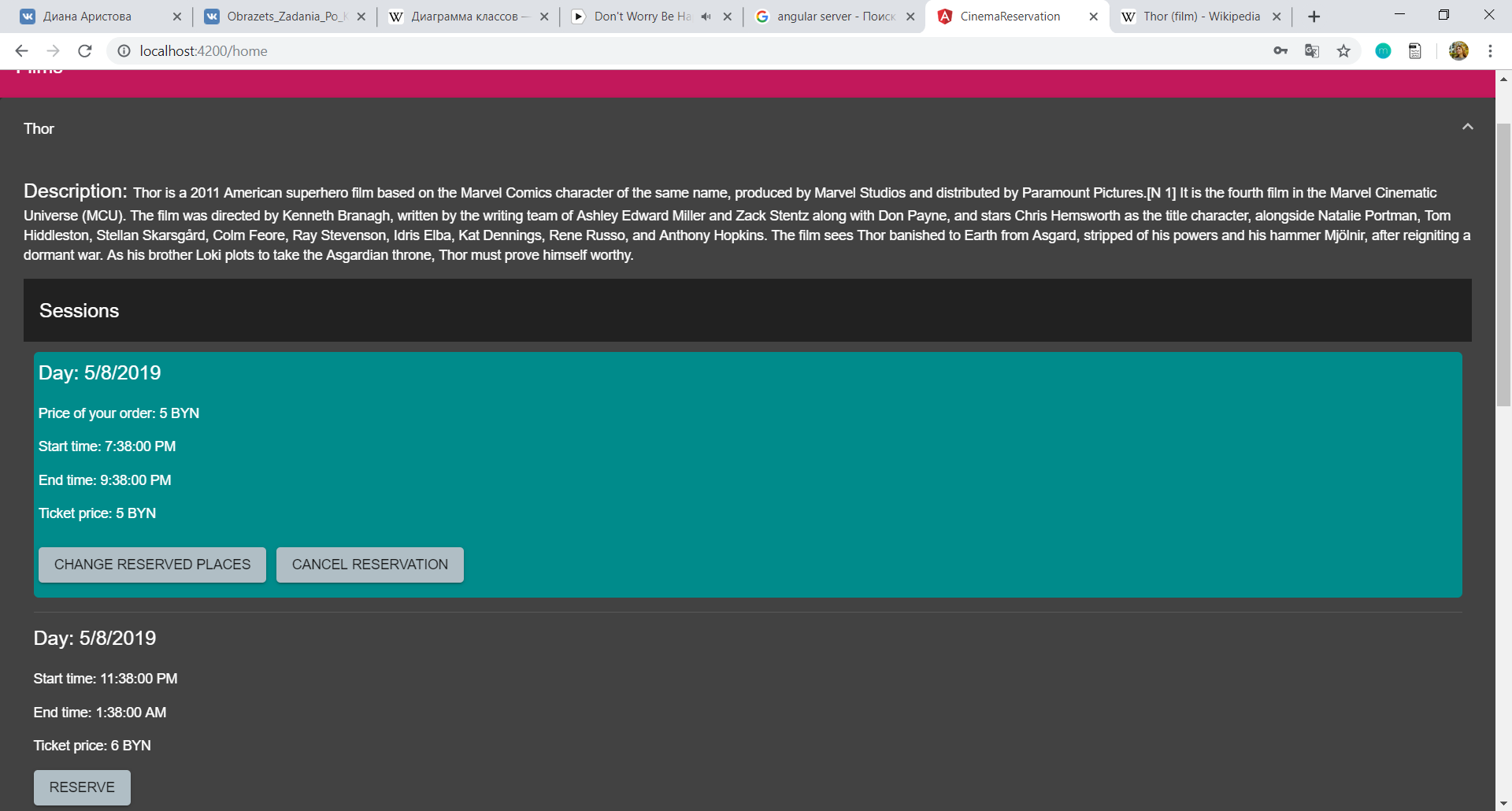


Рис 7 – изменение внешнего вида после резервирования места

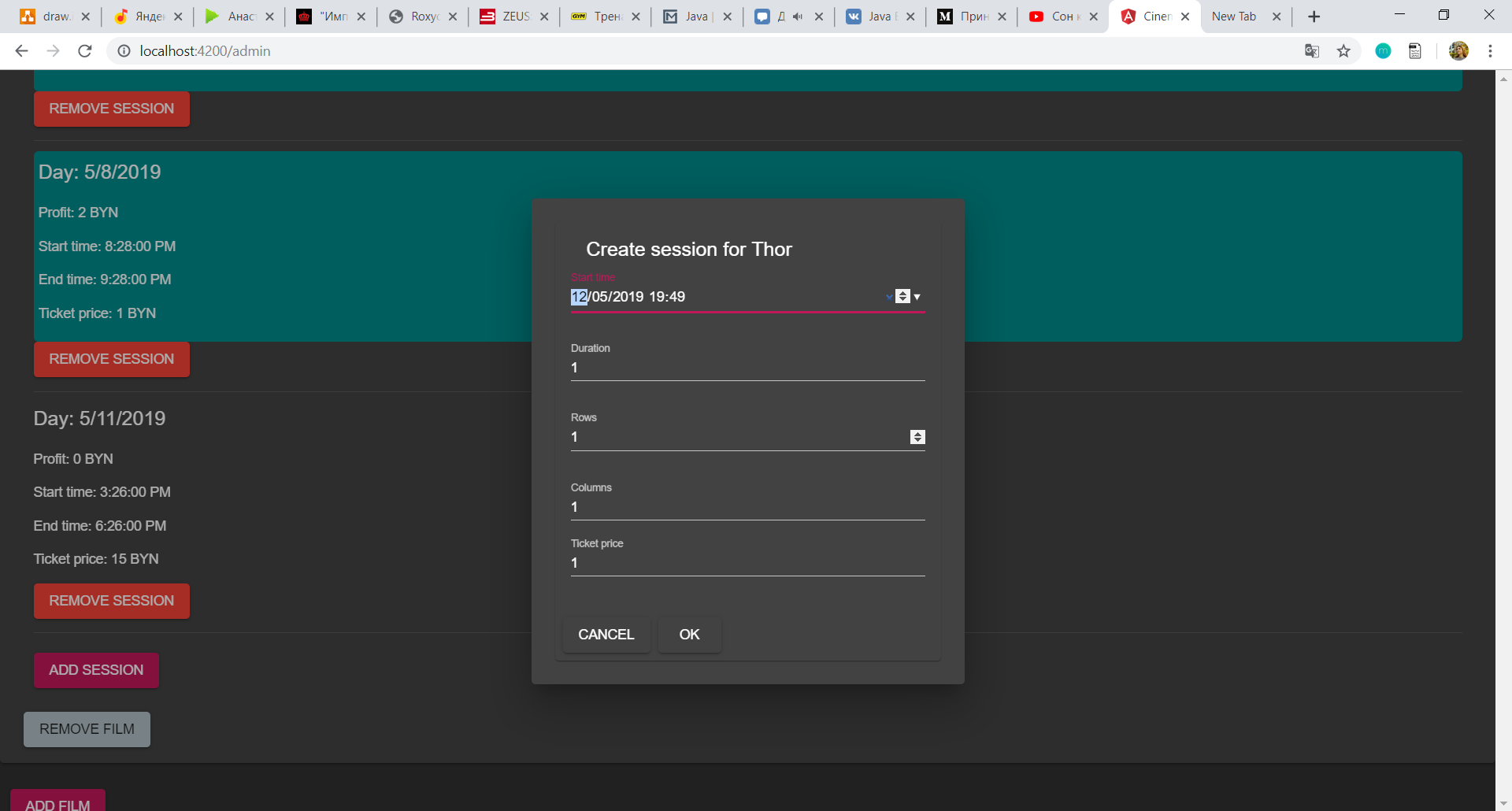


Рис 8 – диалоговое окно добавления сеансов на странице администратора

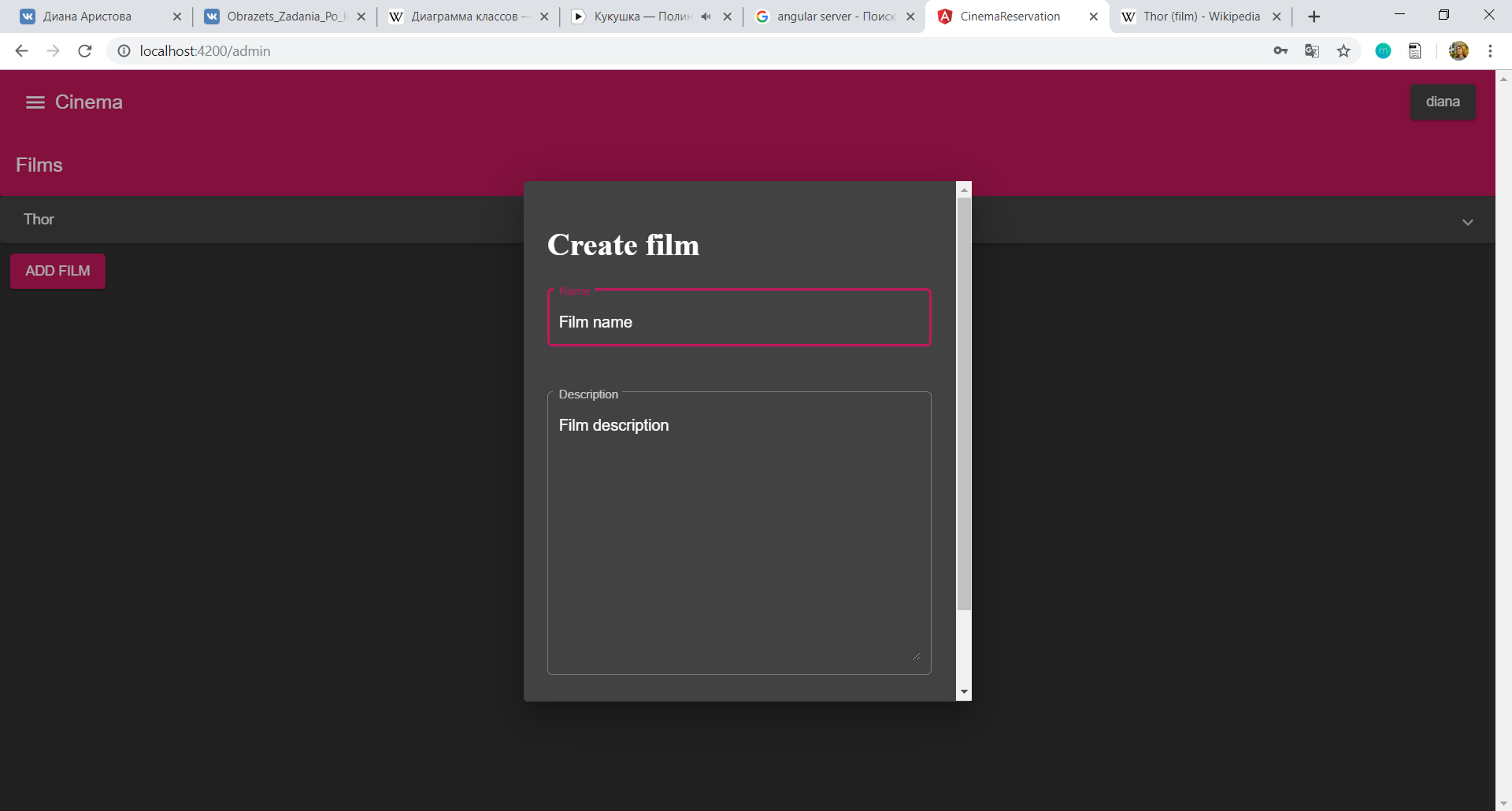


Рис 9 – диалоговое окно добавления фильмов

# Листинг:

Папка database:

Файл mongo-connect.database.ts

import Mongoose from 'mongoose';

export default async (tbName) =>

Mongoose.connect(`mongodb://localhost:27017/${tbName}`, {useNewUrlParser: true});

Папка enums:

Файл user.role.enum.ts:

export enum UserRoleEnum {

user = 'user',

admin = 'admin'

}

Папка guards:

Файл user-role.guard.ts:

import {UserRoleEnum} from '../enums/user-role.enum';

export default (roles: UserRoleEnum[]) => {

return async (ctx, next) => {

if (!ctx.user) {

ctx.statusCode = 401;

ctx.body = { success: false, messages: ['Unauthorized error'] };

return;

}

if (!roles.includes(ctx.user.role)) {

ctx.statusCode = 403;

ctx.body = { success: false, messages: ['Forbidden error'] };

return;

}

await next();

};

};

Папка logger:

Файл configLog4j.ts:

import {LoggerFactory, LoggerFactoryOptions, LFService, LogGroupRule, LogLevel} from "typescript-logging";

const options = new LoggerFactoryOptions()

.addLogGroupRule(new LogGroupRule(new RegExp("rout.+"), LogLevel.Info))

.addLogGroupRule(new LogGroupRule(new RegExp("service.+"), LogLevel.Info));

export const factory = LFService.createNamedLoggerFactory("LoggerFactory", options);

Папка middlewar:

Файл cors.middlewar.ts:

import Cors from 'koa2-cors';

export default () => Cors({

origin: () => '\*',

exposeHeaders: ['WWW-Authenticate', 'Server-Authorization'],

maxAge: 5,

credentials: true,

allowMethods: ['GET', 'POST', 'DELETE'],

allowHeaders: ['Content-Type', 'Authorization', 'Accept'],

}

);

Файл jwt.middlewar.ts:

import {Context} from 'koa';

import JWTService from '../services/jwt.service';

import UserService from '../services/user.service';

export default async (ctx: Context, next) => {

const token: string = ctx.request.headers.authorization;

if (token) {

const userId = await JWTService.getIdFromJwt(token);

if (userId) {

ctx.user = await UserService.getUserById(userId);

ctx.userId = userId;

}

}

await next();

};

Папка models:

Файл film.model.ts:

export class Film {

constructor(

public name: string,

public description: string

) { }

}

Файл film-mongoose.model.ts:

import \* as Mongoose from 'mongoose';

const filmSchema = new Mongoose.Schema({

name: {

type: String,

required: true

},

description: {

type: String,

required: true

}

});

const FilmMongooseModel = Mongoose.model('film', filmSchema);

export default FilmMongooseModel;

Файл place.model.ts:

export default class Place {

public row: number;

public column: number;

}

Файл place-mongoose.model.ts:

import \* as Mongoose from 'mongoose';

const placeSchema: Mongoose.Schema = new Mongoose.Schema({

row: {

type: Number,

required: true

},

column: {

type: Number,

required: true

}

});

export default placeSchema;

import Place from './place.model';

export default class Reservation {

constructor(

public userId: string,

public sessionId: string,

public places: Array<Place>

) { }

}

Файл reservation-mongoose.model.ts:

import \* as Mongoose from "mongoose";

import placeSchema from "./place-mongoose.model";

const reservationSchema: Mongoose.Schema = new Mongoose.Schema({

userId: {

type: String,

required: true

},

sessionId: {

type: String,

required: true

},

places: {

type: [placeSchema],

required: true

}

});

export default reservationSchema;

Файл rest,responce.model.ts:

import User from './user.model';

import Session from './session.model';

import {Film} from './film.model';

import Reservation from './reservation.model';

export class RestResponse {

public success = false;

public messages: string[] = [];

}

export class SignInResponse extends RestResponse {

public token: string;

public user: User;

}

export class SessionResponse extends RestResponse {

session: Session;

}

export class FilmsResponse extends RestResponse {

films: Array<Film>;

}

export class ReservationResponse extends RestResponse {

reservation: Reservation;

}

export class SessionListResponse extends RestResponse {

sessions: Array<Session>;

}

Файл session.model.ts:

import Reservation from './reservation.model';

export default class Session {

public id: string;

public reservations: Array<Reservation>;

constructor(

public filmName: string,

public startTime: Date,

public endTime: Date,

public rows: number,

public columns: number,

public ticketPrice: number

) {

this.reservations = [];

}

}

Файл reservation-mongoose.model.ts:

import \* as Mongoose from 'mongoose';

import reservationSchema from "./reservation-mongoose.model";

const sessionSchema: Mongoose.Schema = new Mongoose.Schema({

filmName: {

type: String,

required: true

},

startTime: {

type: Date,

required: true

},

endTime: {

type: Date,

required: true

},

rows: {

type: Number,

required: true,

},

columns: {

type: Number,

required: true

},

reservations: {

type: [reservationSchema],

default: []

},

ticketPrice: {

type: Number,

required: true

}

});

const SessionMongooseModel = Mongoose.model('cinema-session', sessionSchema);

export default SessionMongooseModel;

Файл user.model.ts:

import {UserRoleEnum} from '../enums/user-role.enum';

export default class User {

public id: string;

constructor(

public username: string,

public name: string,

public password: string,

public role: UserRoleEnum

) { }

}

Файл reservation-mongoose.model.ts:

import \* as Mongoose from "mongoose";

const userSchema = new Mongoose.Schema({

username: {type: String, required: true},

name: {type: String, required: true },

password: {type: String, required: true},

role: {type: String, required: true}

});

const UserMongooseModel = Mongoose.model('cinema-user', userSchema);

export default UserMongooseModel;

Папка routes:

Файл index.route.ts:

import Router from 'koa-router';

const router = new Router();

import authorizeRouter from './authorize.router';

import sessionRouter from './session.router';

import reservationRouter from './reservation.router';

import filmRouter from './film.router';

router.use('/auth', authorizeRouter.routes());

router.use('/film', filmRouter.routes());

router.use('/session', sessionRouter.routes());

router.use('/reservation', reservationRouter.routes());

export default router;

Файл autorize.route.ts:

import Router from 'koa-router';

import UserService from '../services/user.service';

import User from '../models/user.model';

import JWTService from '../services/jwt.service';

import {RestResponse, SignInResponse} from '../models/rest.response.model';

import {UserRoleEnum} from '../enums/user-role.enum';

import {UserValidators} from '../validators/user.validators';

import {factory} from "../logger/configLog4j";

const logSignIn = factory.getLogger('rout.SignIn');

const logSignUp = factory.getLogger('rout.SignUp');

const router = new Router();

router

.post('/signIn', async ctx => {

logSignIn.debug('Go to /signIn');

logSignIn.info('Go to /signIn');

const response: SignInResponse = new SignInResponse();

ctx.response.body = response;

const unsignedUser = ctx.request.body as User;

logSignIn.debug('Take send information from User');

const errors = await UserValidators.validateUserToSignIn(unsignedUser);

logSignIn.debug('Validation data');

logSignIn.info('Validation data');

if (errors.length) {

logSignIn.error(errors[0]);

response.messages = errors;

} else {

logSignIn.info('Authorization user!');

logSignIn.debug('Pass validation');

response.messages.push('Successful signing');

response.user = await UserService.getUserByUsername(unsignedUser.username);

response.success = true;

response.token = JWTService.createJwt(response.user.id);

}

})

.post('/signUp', async ctx => {

logSignUp.debug('Go to /auth/signUp');

logSignUp.info('Go to /auth/signUp');

const response: RestResponse = new RestResponse();

ctx.response.body = response;

const uncreatedUser = ctx.request.body as User;

logSignUp.debug('Take information from user');

const errors = await UserValidators.validateUserToSignUp(uncreatedUser);

logSignUp.debug('Validation data');

if (errors.length) {

logSignUp.error(errors[0]);

response.messages = errors;

} else {

logSignUp.debug('Pass Validation');

logSignUp.info('Create new user in DB');

const createdUser = new User(

uncreatedUser.username,

uncreatedUser.name,

uncreatedUser.password,

UserRoleEnum.user

);

response.messages.push('Successful joining');

response.success = true;

await UserService.saveUser(createdUser);

}

});

export default router;

Файл film.route.ts:

import Router from 'koa-router';

import {Film} from '../models/film.model';

import {FilmsResponse, RestResponse} from '../models/rest.response.model';

import userRoleGuardFactory from '../guards/user-role.guard';

import {UserRoleEnum} from '../enums/user-role.enum';

import FilmService from '../services/film.service';

import {FilmValidators} from '../validators/film.validators';

import {factory} from "../logger/configLog4j";

const logAdd = factory.getLogger('rout.FilmAdd');

const logRemove = factory.getLogger('rout.FilmRemove');

const logAll = factory.getLogger('rout.FilmAll');

const router = new Router();

router

.post('/add', userRoleGuardFactory([UserRoleEnum.admin]), async ctx => {

logAdd.info('/film/add');

logAdd.debug('Go to /film/add');

const film = ctx.request.body as Film;

const response = new RestResponse();

ctx.response.body = response;

logAdd.debug('Validation date');

const errors = await FilmValidators.validateFilmToAdd(film);

if (errors.length) {

logAdd.error('Not valid date: '+ errors[0]);

response.messages = errors;

} else {

logAdd.debug('Data pass validation');

logAdd.info('Film added');

await FilmService.addFilm(film);

response.success = true;

response.messages.push('Film added');

}

})

.post('/remove', userRoleGuardFactory([UserRoleEnum.admin]), async ctx => {

logRemove.info('/film/remove');

logRemove.debug('Go to /film/remove');

const film = ctx.request.body as Film;

const response = new RestResponse();

ctx.response.body = response;

logRemove.debug('GetInformation');

const errors = await FilmValidators.validateFilmToRemove(film);

logRemove.debug('Validation');

if (errors.length) {

logRemove.error(errors[0]);

response.messages = errors;

} else {

logRemove.debug('Pass validation and remove');

logRemove.info('Remove film');

await FilmService.removeFilm(film.name);

response.success = true;

response.messages.push('Film removed');

}

})

.get('/all', async ctx => {

logAll.debug('Go to /film/all');

const response = new FilmsResponse();

ctx.response.body = response;

response.films = await FilmService.getFilms();

logAll.info('get film list from DB');

response.success = true;

response.messages.push('All films');

});

export default router;

Файл reservation.route.ts:

import Router from 'koa-router';

import {ReservationResponse, RestResponse, SessionResponse} from '../models/rest.response.model';

import ReservationService from '../services/reservation.service';

import Reservation from '../models/reservation.model';

import {ReservationValidator} from '../validators/reservation.validator';

import {factory} from "../logger/configLog4j";

const logAdd = factory.getLogger('rout.AddReservation');

const logRemove = factory.getLogger('rout.RemoveReservation');

const logOne = factory.getLogger('rout.OneReservation');

const router: Router = new Router();

router

.post('/add', async ctx => {

logAdd.debug('Go to /reservation/add');

logAdd.info('/reservation/add');

const reservation: Reservation = ctx.request.body;

const response: RestResponse = new SessionResponse();

ctx.response.body = response;

logAdd.debug('get information');

const errors = ReservationValidator.validateReservationToAdd(reservation);

logAdd.debug('Validation');

if (errors.length) {

logAdd.error(errors[0]);

logAdd.debug('Error');

response.messages = errors;

} else {

logAdd.info('Add reservation');

logAdd.debug('Pass validation');

await ReservationService.addReservation(reservation);

logAdd.debug('Add reservation');

response.success = true;

response.messages.push('Successful place reservation');

}

})

.post('/remove', async ctx => {

logRemove.debug('Go to reservation/remove');

logRemove.info('reservation/remove');

const response = new RestResponse();

ctx.response.body = response;

logRemove.debug('Get data');

const sessionId: string = ctx.request.body.sessionId;

const userId: string = ctx.userId;

const reservation = {sessionId, userId} as Reservation;

const errors = await ReservationValidator.validateReservationToRemove(reservation);

if (errors.length) {

logRemove.error(errors[0]);

logRemove.debug('Error');

response.messages = errors;

} else {

logRemove.debug('Pass validation');

await ReservationService.removeReservation(reservation);

logRemove.info('Remove film');

response.success = true;

response.messages.push('Reservation removed');

}

})

.post('/one', async ctx => {

logOne.debug('Go to /reservation/one');

logOne.info('/reservation/one');

const response = new ReservationResponse();

ctx.response.body = response;

logOne.debug('Get data from body');

const sessionId = ctx.request.body.sessionId;

const userId = ctx.userId;

const reservation = {sessionId, userId} as Reservation;

const errors = await ReservationValidator.validateReservationToGet(reservation);

logOne.debug('Validation');

if (errors.length) {

logOne.error(errors[0]);

response.messages = errors;

} else {

logOne.debug('Pass Validation');

response.reservation = await ReservationService.getReservation(reservation);

logOne.info('Get one reservation');

response.success = true;

response.messages.push('Reservation found');

}

});

export default router;

Файл session.route.ts:

import Router from 'koa-router';

import Session from '../models/session.model';

import SessionService from '../services/session.service';

import {RestResponse, SessionListResponse, SessionResponse} from '../models/rest.response.model';

import userRoleGuardFactory from '../guards/user-role.guard';

import {UserRoleEnum} from '../enums/user-role.enum';

import {SessionValidators} from '../validators/session.validators';

import {factory} from "../logger/configLog4j";

const logAll = factory.getLogger('rout.AllSessionFilm');

const logOne = factory.getLogger('rout.SessionFilm');

const logAdd = factory.getLogger('rout.AddSession');

const logRemove = factory.getLogger('rout.RemoveSession');

const router = new Router();

router

.get('/all/:filmName', async ctx => {

logAll.debug('Go to session/all/:filmName');

const response = new SessionListResponse();

ctx.response.body = response;

logAll.debug('Get data');

const filmName: string = ctx.params.filmName;

logAll.info('session/all/'+ filmName);

if (!filmName) {

logAll.error('filmName required');

// response.messages.push('filmName required');

return;

}

const sessions = await SessionService.getAllSessionsByFilmName(filmName);

if (!sessions){

logAll.warn('Session not found' + sessions);

return ;

}

logAll.debug('Found session');

response.success = true;

response.sessions = sessions;

logAll.info('Session found');

// response.messages.push('Session found');

})

.get('/one/:sessionId', async ctx => {

const response: SessionResponse = new SessionResponse();

ctx.response.body = response;

const sessionId: string = ctx.params.sessionId;

logOne.debug('Go to session/one/'+ sessionId);

logOne.info('session/one/'+ sessionId);

if (!sessionId || sessionId === '') {

logOne.error('Incorrect session id');

// response.messages.push('Incorrect session id');

return;

}

const session: Session = await SessionService.getSessionById(sessionId);

logOne.debug('get user by ID');

if (!session ) {

logOne.error('Session not found');

// response.messages.push('Session not found');

return;

}

response.success = true;

logOne.info('Session found');

response.messages.push('Session founded');

response.session = session;

})

.post('/add', userRoleGuardFactory([UserRoleEnum.admin]), async ctx => {

logAdd.info('session/add');

logAdd.debug('Go to session/add');

const session = ctx.request.body as Session;

const response: RestResponse = new RestResponse();

ctx.response.body = response;

logAdd.debug('Get data from body');

const errors = SessionValidators.validateSessionToAddByDataFields(session);

logAdd.debug('Validation');

if (errors.length) {

logAdd.error(errors[0]);

response.messages = errors;

} else {

logAdd.debug('Pass validation');

await SessionService.addSession(session);

logAdd.info('Add new session');

response.success = true;

response.messages.push('Added new session');

}

})

.post('/remove', async ctx => {

logRemove.debug('Go to session/remove');

logRemove.info('session/remove');

const session = ctx.request.body as Session;

const response = new RestResponse();

logRemove.debug('Get data');

ctx.response.body = response;

logRemove.debug('Validation');

const errors = await SessionValidators.validateSessionToRemove(session);

if (errors.length) {

logRemove.error(errors[0]);

response.messages = errors;

} else {

logRemove.debug('Pass validation');

await SessionService.removeSession(session);

logRemove.info('Session removed');

response.messages.push('Session removed');

response.success = true;

}

});

export default router;

Папка service:

Файл film.service.ts:

import {Film} from '../models/film.model';

import FilmMongooseModel from '../models/film-mongoose.model';

import {MongooseDocument} from 'mongoose';

import {factory} from "../logger/configLog4j";

const log = factory.getLogger('service.Film');

export default class FilmService {

static async addFilm(film: Film): Promise<void> {

await new FilmMongooseModel(film).save();

}

static async removeFilm(filmName: string): Promise<MongooseDocument> {

return await FilmMongooseModel.findOneAndDelete({name: filmName});

}

static async isFilmExist(film: Film): Promise<boolean> {

if (film) {

return !!await FilmMongooseModel.findOne({name: film.name, description: film.description});

}

else {

return null;

}

}

static async getFilms(): Promise<Array<Film>> {

const docFilms = await FilmMongooseModel.find({});

return docFilms.map(doc => FilmService.transformDocToFilm(doc));

}

static transformDocToFilm(doc) {

return new Film(doc.name, doc.description);

}

}

Файл jwt.service.ts:

import \* as JSONWebToken from 'jsonwebtoken';

import config from '../../config.json';

import {factory} from "../logger/configLog4j";

const log = factory.getLogger('service.JWT');

export default class JWTService {

static async getIdFromJwt(token: string): Promise<string> {

try {

const decodedObject = await JSONWebToken.verify(token, config.jwt.secret);

log.info('decoding object');

} catch (e) {

log.fatal('Unknow exception ' + e);

return null;

}

// @ts-ignore

return decodedObject.id;

}

static createJwt(id: string): string {

return JSONWebToken.sign({id}, config.jwt.secret, {expiresIn: 60 \* 60 \* 24});

}

}

Файл reservation.service.ts:

import Reservation from '../models/reservation.model';

import SessionMongooseModel from '../models/session-mongoose.model';

import {factory} from "../logger/configLog4j";

const log = factory.getLogger('service.Reservation');

export default class ReservationService {

static async addReservation(reservation: Reservation): Promise<void> {

const sessionDoc: any = await SessionMongooseModel.findById(reservation.sessionId);

const reservations = sessionDoc.reservations as Reservation[];

const oldReservationIndex = reservations.findIndex(r => r.userId === reservation.userId);

if (oldReservationIndex !== -1) {

log.debug('Old reservation without change');

reservations[oldReservationIndex] = reservation;

} else {

log.debug('Add new reservation');

reservations.push(reservation);

}

await sessionDoc.save();

}

static async removeReservation(reservation: Reservation): Promise<boolean> {

const sessionDoc: any = await SessionMongooseModel.findById(reservation.sessionId);

const reservations = sessionDoc.reservations as Reservation[];

const oldReservationIndex = reservations.findIndex(r => r.userId === reservation.userId);

if (oldReservationIndex !== -1) {

log.debug('Remove this reservation');

reservations.splice(oldReservationIndex, 1);

await sessionDoc.save();

return true;

}

return false;

}

static async getReservation(reservation: Reservation): Promise<Reservation> {

log.debug('get reservation');

const sessionDoc: any = await SessionMongooseModel.findById(reservation.sessionId);

const reservations = sessionDoc.reservations.filter(r => r.userId === reservation.userId);

return reservations[0];

}

}

Файл sesion.service.ts:

import Session from '../models/session.model';

import SessionMongooseModel from '../models/session-mongoose.model';

import {MongooseDocument} from 'mongoose';

import {factory} from "../logger/configLog4j";

const log = factory.getLogger('service.Session');

export default class SessionService {

static async addSession(session: Session): Promise<void> {

log.debug('add Session');

await new SessionMongooseModel(session).save();

}

static async getSessionById(sessionId: string): Promise<Session> {

if (!await SessionService.isExistSessionById){

log.warn("Session not found!")

}

else {

log.debug('get session bu id');

const sessionDoc = await SessionMongooseModel.findById(sessionId);

return this.transformToSession(sessionDoc);

}

}

private static transformToSession(sessionDoc): Session {

const session = new Session(

sessionDoc.filmName,

sessionDoc.startTime,

sessionDoc.endTime,

sessionDoc.rows,

sessionDoc.columns,

sessionDoc.ticketPrice

);

session.id = sessionDoc.id;

session.reservations = sessionDoc.reservations;

return session;

}

static async getAllSessionsByFilmName(filmName: string): Promise<Session[]> {

log.debug('getAllSessionsByFilmName');

const sessionDocs = await SessionMongooseModel.find(

{filmName: filmName}

);

const sessions = [];

sessionDocs.forEach(

sessionDoc => sessions.push(

SessionService.transformToSession(sessionDoc)

));

return sessions;

}

static async isExistSession(session: Session): Promise<boolean> {

log.debug('is exist session');

return !!await SessionMongooseModel.findById(session.id);

}

private static async isExistSessionById(sessionId: string): Promise<boolean> {

log.debug('isExistSessionById');

return !!await SessionMongooseModel.findById({\_id:sessionId});

}

static async removeSession(session: Session): Promise<MongooseDocument> {

log.debug('removeSession');

return await SessionMongooseModel.findOneAndDelete(session.id);

}

}

Файл user.service.ts:

import User from '../models/user.model';

import UserMongooseModel from '../models/user-mongoose.model'

import {factory} from "../logger/configLog4j";

const log = factory.getLogger('service.User');

export default class UserService {

static async saveUser(user: User): Promise<void> {

if (!user) {

log.warn('User is null')

} else {

await new UserMongooseModel(user).save();

}

}

static async getUserById(id: string): Promise<User> {

if (!id) {

return null

}

const user = await UserMongooseModel.findById(id);

if (!user) {

return null;

} else {

return this.transformToUserModel(user);

}

}

static async getUserByUsername(username: string): Promise<User> {

const user = await UserMongooseModel.findOne({username});

if (!user) {

log.warn('User = null');

return null;

} else {

return UserService.transformToUserModel(user);

}

}

private static transformToUserModel(userDoc): User {

const user: User = new User(userDoc.username, userDoc.name, userDoc.password, userDoc.role);

user.id = userDoc.id;

return user;

}

static async isUserExist(username: string, password: string): Promise<boolean> {

return !!await UserMongooseModel.findOne({username, password});

}

static async isUsernameFree(username: string): Promise<boolean> {

if (!username) {

return false

} else {

return !await UserMongooseModel.findOne({username});

}

}

}

Папка validators:

Файл film.validators.ts:

import {Film} from '../models/film.model';

import FilmService from '../services/film.service';

export class FilmValidators {

private static validateFilmToAddByDataFields(film: Film): string[] {

const errors = [];

if (!film) {

errors.push('Film value required');

}

else if (!film.name) {

errors.push('Film name required');

}

else if (!film.description) {

errors.push('Film description required');

}

return errors;

}

private static validateFilmByName(film: Film): string[] {

const errors = [];

if (film == null){

errors.push('Film is null');

return errors;

}

if (!film.name) {

errors.push('Film name required');

}

return errors;

}

private static async validateFilmToAddByUniqueness(film: Film): Promise<string[]> {

const errors = [];

if (await FilmService.isFilmExist(film)) {

errors.push('Film name is already taken');

}

return errors;

}

private static async validateFilmToRemoveByExistence(film: Film): Promise<string[]> {

const errors = [];

if (!await FilmService.isFilmExist(film)) {

errors.push('Film not exists');

}

return errors;

}

static async validateFilmToRemove(film: Film): Promise<string[]> {

const errors = FilmValidators.validateFilmByName(film);

return errors.length

? errors

: await FilmValidators.validateFilmToRemoveByExistence(film);

}

static async validateFilmToAdd(film: Film): Promise<string[]> {

const errors = FilmValidators.validateFilmToAddByDataFields(film);

return errors.length

? errors

: FilmValidators.validateFilmToAddByUniqueness(film);

}

}

Файл reservation.validators.ts:

import Reservation from '../models/reservation.model';

import ReservationService from '../services/reservation.service';

import SessionMongooseModel from "../models/session-mongoose.model";

export class ReservationValidator {

static validateReservationToAdd(reservation: Reservation): string[] {

const nullErrors = ReservationValidator.validateReservationByNull(reservation);

return nullErrors.length

? nullErrors

: [

...ReservationValidator.validateReservationByIdFields(reservation),

...ReservationValidator.validateReservationByPlaces(reservation)

];

}

static async validateReservationToRemove(reservation: Reservation): Promise<string[]> {

const nullErrors = ReservationValidator.validateReservationByNull(reservation);

if (nullErrors.length) {

return nullErrors;

}

const idErrors = ReservationValidator.validateReservationByIdFields(reservation);

if (idErrors.length) {

return idErrors;

}

return ReservationValidator.validateReservationByExistence(reservation);

}

static async validateReservationToGet(reservation: Reservation): Promise<string[]> {

const errors = [];

const idErrors = ReservationValidator.validateReservationByIdFields(reservation);

if (idErrors.length) {

return idErrors;

}

return ReservationValidator.validateReservationByExistence(reservation);

}

private static validateReservationByIdFields(reservation: Reservation): string[] {

const errors = [];

if (!reservation.sessionId) {

errors.push('SessionId required');

}

if (!reservation.userId) {

errors.push('UserId is required');

}

return errors;

}

private static validateReservationByPlaces(reservation: Reservation): string[] {

const errors = [];

if (!reservation.places || !reservation.places.length) {

errors.push('Places are required');

}

return errors;

}

private static validateReservationByNull(reservation: Reservation): string[] {

const errors = [];

if (!reservation) {

errors.push('Reservation is required');

}

return errors;

}

private static async validateReservationByExistence(reservation: Reservation): Promise<string[]> {

return await ReservationService.getReservation(reservation)

? []

: ['Reservation not found'];

}

}

Файл session.validators.ts:

import Session from '../models/session.model';

import SessionService from '../services/session.service';

export class SessionValidators {

static validateSessionToAddByDataFields(session: Session): string[] {

const errors = [];

if (!session){

errors.push('Session is required');

}

else if (!session.filmName) {

errors.push('Film name is required');

}

else if (!session.startTime) {

errors.push('Start time is required');

}

else if (!session.endTime) {

errors.push('End time is required');

}

else if (!session.columns) {

errors.push('Columns value is required');

}

else if (!session.rows) {

errors.push('Rows value is required');

}

else if (!session.ticketPrice) {

errors.push('Ticket price is required');

}

return errors;

}

private static validateSessionIdToRemoveByDateField(session: Session): string[] {

const errors = [];

if (!session){

errors.push('Session is required');

}

else if (!session.id) {

errors.push('Id is required');

}

return errors;

}

private static async validateSessionToRemoveByExistence(session: Session): Promise<string[]> {

const errors = [];

if (!await SessionService.isExistSession(session)) {

errors.push('Session not found');

}

return errors;

}

static async validateSessionToRemove(session: Session): Promise<string[]> {

const errors = SessionValidators.validateSessionIdToRemoveByDateField(session);

return errors.length

? errors

: SessionValidators.validateSessionToRemoveByExistence(session);

}

}

Файл user.validators.ts:

import User from '../models/user.model';

import UserService from '../services/user.service';

export class UserValidators {

//signIn

private static validateUserToSignInByDataFields(user: User): string[] {

const errors = [];

if (!user.username) {

errors.push('Username is required');

}

if (!user.password) {

errors.push('Password is required');

}

return errors;

}

private static async validateUserToSignInByAccess(user: User): Promise<string[]> {

const errors = [];

if (!await UserService.isUserExist(user.username, user.password)) {

errors.push('Invalid username or password');

}

return errors;

}

private static validateUserToSignUpByDataFields(user: User): string[] {

const errors = [];

if (!user.name) {

errors.push('Name is required');

}

return [

...errors,

...UserValidators.validateUserToSignInByDataFields(user)

];

}

static async validateUserToSignIn(user: User): Promise<string[]> {

const errors = this.validateUserToSignInByDataFields(user);

return errors.length

? errors

: await UserValidators.validateUserToSignInByAccess(user);

}

//signUp

private static async validateUsernameToSignUpByUniqueness(username: string): Promise<string[]> {

const errors = [];

if (! await UserService.isUsernameFree(username)) {

errors.push('Username is already taken');

}

return errors;

}

static async validateUserToSignUp(user: User): Promise<string[]> {

const errors = this.validateUserToSignUpByDataFields(user);

return errors.length

? errors

: await UserValidators.validateUsernameToSignUpByUniqueness(user.username);

}

}

Папка src

Файл index.ts:

import Koa from 'koa';

import KoaBodyParser from 'koa-bodyparser';

import router from './routes/index';

import corsMiddleware from './middleware/cors.middleware';

import jwtMiddleware from './middleware/jwt.middleware';

const logger = require('koa-logger');

const app = new Koa();

app.use(logger());

app.use(corsMiddleware());

app.use(jwtMiddleware);

app.use(KoaBodyParser());

app.use(router.routes());

const server = app.listen(3000);

module.exports = server;

server.close();

export default app;

Папка test

Папка db

Файл mogodb.conect.spec.ts:

import mongoConnect from "../../src/database/mongo-connect.database";

it('work MongoConnect ', async () => {

await mongoConnect('cinema-test').then(tolk => console.log('MongoDB connect'))

});

Папка film-test

Файл film.service.spec.ts:

import mongoConnect from "../../src/database/mongo-connect.database";

import FilmService from "../../src/services/film.service";

import {Film} from "../../src/models/film.model";

beforeEach(async () =>{

await mongoConnect('cinema-test').then(() => console.log('MongoDB connected'));

});

//removeFilm

it('should be pass removeFilm', async () => {

const removeFilm = await FilmService.removeFilm("name");

const film = FilmService.transformDocToFilm(removeFilm);

expect(film.name).toEqual("name");

});

it('should be pass removeFilm name = null', async () => {

const removeFilm = await FilmService.removeFilm(null);

expect(removeFilm).toEqual(null);

});

//addFilm

it('should pass addFilm ', async () => {

const addFilm = await FilmService.addFilm(new Film("name", "hia"))

.then(tolk => console.log('Film save'));

});

//isFilmExist

it('should be pass isFilmExist name is not right', async () => {

const film = new Film("no name", "hiha");

const isExist = await FilmService.isFilmExist(film);

expect(isExist).toEqual(false);

});

it('should be pass isFilmExist description is right', async () => {

const film = new Film("name", "hiha");

const isExist = await FilmService.isFilmExist(film);

expect(isExist).toEqual(false);

});

it('should be pass isFilmExist description is right', async () => {

const film = new Film("name", "hia");

const isExist = await FilmService.isFilmExist(film);

expect(isExist).toEqual(true);

});

it('should be pass isFilmExist null', async () => {

const isExist = await FilmService.isFilmExist(null);

expect(isExist).toEqual(null);

});

//getFilms

it('should be pass getFilms', async () => {

const film = new Film("name", "hia");

const films = await FilmService.getFilms();

expect(films).toEqual([film]);

});

Файл film.validator.spec.ts

import mongoConnect from "../../src/database/mongo-connect.database";

import {FilmValidators} from "../../src/validators/film.validators";

import {Film} from "../../src/models/film.model";

beforeEach(async () =>{

await mongoConnect('cinema-test').then(() => console.log('MongoDB connected'));

});

it('film validator for add film without problem', async () => {

const newFilm = await FilmValidators.validateFilmToAdd( new Film('name', 'description'));

expect(newFilm).toEqual([])

});

it('film validator for add without name', async () =>{

const newFilm = await FilmValidators.validateFilmToAdd( new Film(null, 'description'));

expect(newFilm).toEqual(['Film name required'])

});

it('film validator for add without description', async () =>{

const newFilm = await FilmValidators.validateFilmToAdd( new Film('name', null));

expect(newFilm).toEqual(['Film description required'])

});

it('film validator for add new exist film ', async () => {

const film = await new Film('Tor', 'description');

const addfilm = await FilmValidators.validateFilmToAdd( film);

expect(addfilm).toEqual(['Film name is already taken']);

});

// remove

it('film validator for remove ', async () => {

const film = await new Film('Tor', 'des');

const addfilm = await FilmValidators.validateFilmToRemove( film);

expect(addfilm).toEqual([]);

});

it('film validator for remove without name', async () => {

const film = await new Film(null, 'des');

const addfilm = await FilmValidators.validateFilmToRemove( film);

expect(addfilm).toEqual(['Film name required']);

});

it('film validator for remove without description', async () => {

const film = await new Film('Tor', null);

const addfilm = await FilmValidators.validateFilmToRemove( film);

expect(addfilm).toEqual([]);

});

it('film validator for remove not exist film', async () => {

const film = await new Film('Film', 'some film');

const addfilm = await FilmValidators.validateFilmToRemove( film);

expect(addfilm).toEqual(['Film not exists']);

});

it('film validator for remove without film', async () => {

const addfilm = await FilmValidators.validateFilmToRemove(null);

expect(addfilm).toEqual(['Film is null']);

});

it('film validator for add when film = null ', async () => {

const newFilm = await FilmValidators.validateFilmToAdd(null);

expect(newFilm).toEqual(['Film value required']);

});

Папка reservation-test

Файл reservation.service.spec.ts:

import mongoConnect from "../../src/database/mongo-connect.database";

import Reservation from "../../src/models/reservation.model";

import Place from "../../src/models/place.model";

import ReservationService from "../../src/services/reservation.service";

beforeEach(async () =>{

// place.column = 2;

// place.row = 1;

await mongoConnect('cinema-test').then(() => console.log('MongoDB connected'));

});

it('should add reservation', async () => {

const place: Place = new Place();

place.row = 1;

place.column = 1;

await ReservationService.addReservation(new Reservation('5cd1a31ba3e7b7245815b279', '5cd18bbcfab47824182c5fd3', [place]))

.then(() => console.log('Save object reservation'));

});

// removeReservation

it('should removeReservation', async () => {

const place: Place = new Place();

place.row = 1;

place.column = 1;

const reserv = new Reservation('5cd1a31ba3e7b7245815b279', '5cd18bbcfab47824182c5fd3', [place]);

const remove = await ReservationService.removeReservation(reserv);

expect(remove).toEqual(true);

});

//getReservation

it('should getReservation', async () => {

const place: Place = new Place();

place.row = 1;

place.column = 1;

const reserv = new Reservation('5cd1a31ba3e7b7245815b279', '5cd18bbcfab47824182c5fd3', [place]);

await ReservationService.addReservation(new Reservation('5cd1a31ba3e7b7245815b279', '5cd18bbcfab47824182c5fd3', [place]))

.then(() => console.log('Save object reservation'));

const get = await ReservationService.getReservation(reserv);

expect(get.sessionId).toEqual(reserv.sessionId);

});

Файл reservation.validator.spec.ts

import mongoConnect from "../../src/database/mongo-connect.database";

import {ReservationValidator} from "../../src/validators/reservation.validator";

import Reservation from "../../src/models/reservation.model";

import Place from "../../src/models/place.model";

import ReservationService from "../../src/services/reservation.service";

const place: Place = new Place();

const reservedPlace = new Reservation('5cd1a31ba3e7b7245815b279', "5cd18bbcfab47824182c5fd3", [place] );

beforeEach(async () =>{

place.column = 2;

place.row = 1;

await mongoConnect('cinema-test').then(() => console.log('MongoDB connected'));

});

it('Add reservation when reservation = null', async () => {

const newRes = await ReservationValidator.validateReservationToAdd(null);

expect(newRes[0]).toEqual('Reservation is required');

});

it('Add reservation without sessionId', async () => {

const newRes = await ReservationValidator.validateReservationToAdd(new Reservation('123',null,[place]));

expect(newRes[0]).toEqual('SessionId required');

});

it('Add reservation without userId', async () => {

const newRes = await ReservationValidator.validateReservationToAdd(new Reservation(null, "123", [place] ));

expect(newRes[0]).toEqual('UserId is required')

});

it('Add reservation with empty array place', async () => {

const newRes = await ReservationValidator.validateReservationToAdd(new Reservation('123', "123", [] ));

expect(newRes[0]).toEqual('Places are required')

});

it('Add reservation', async () => {

const newRes = await ReservationValidator.validateReservationToAdd(new Reservation('123', "123", [place] ));

expect(newRes).toEqual([])

});

// it('Remove reservation', async () => {

// // const newRes = await ReservationValidator.validateReservationToRemove(reservedPlace);

// // expect(newRes).toEqual([])

// await ReservationService.addReservation(reservedPlace);

//

// });

Папка rout-test

Файл routing.spec.ts:

import mongoConnect from "../../src/database/mongo-connect.database";

const testFilm = { filmname: 'Tor'};

const request = require('supertest');

const server = require('../../src');

beforeAll(async () => {

await mongoConnect('cinema-test').then(() => console.log('MongoDB connected'));

// do something before anything else runs

console.log('Jest starting!');

});

// close the server after each test

afterAll(() => {

server.close();

console.log('server closed!');

});

it('get home route POST /auth/signIn', async () => {

const response = await request(server).post('/auth/signIn');

expect(response.status).toEqual(200);

});

it('get home route POST /auth/signUp', async () => {

const response = await request(server).post('/auth/signUp');

expect(response.status).toEqual(200);

});

it('get home route POST /film/add', async () => {

const response = await request(server).post('/film/add');

expect(response.status).toEqual(200);

});

it('get home route POST /film/remove', async () => {

const response = await request(server).post('/film/remove');

expect(response.status).toEqual(200);

});

it('get home route POST /session/add', async () => {

const response = await request(server).post('/session/add');

expect(response.status).toEqual(200);

});

it('get home route POST /session/remove', async () => {

const response = await request(server).post('/session/remove');

expect(response.status).toEqual(200);

});

it('get home route POST /reservation/remove', async () => {

const response = await request(server).post('/reservation/remove');

expect(response.status).toEqual(200);

});

it('get home route POST /reservation/add', async () => {

const response = await request(server).post('/reservation/add');

expect(response.status).toEqual(200);

});

it('get home route POST /reservation/one', async () => {

const response = await request(server).post('/reservation/one');

expect(response.status).toEqual(200);

});

it('get home route GET /session/all/:filmName', async () => {

const response = await request(server).get('/session/all/Tor');

expect(response.status).toEqual(200);

});

// it(' dont get route GET /session/all/tower', async () => {

// const response = await request(server).get('/session/all/tower');

// expect(response.status).toEqual(500);

// });

it('get home route GET /session/one/:sessionId', async () => {

const response = await request(server).get('/session/one/5cd1260d1129770934e14e93');

expect(response.status).toEqual(200);

});

it(' dont get home route GET /session/one/:sessionId', async () => {

const response = await request(server).get('/session/one/5cd186655e18b025a4d6d7dd');

expect(response.status).toEqual(500);

});

it('get home route GET /film/all', async () => {

const response = await request(server).get('/film/all');

expect(response.status).toEqual(200);

});

it('get home route GET /', async () => {

const response = await request(server).get('/');

expect(response.status).toEqual(404);

});

Папка session-test

Файл session.service.spec.ts:

import SessionService from "../../src/services/session.service";

import Session from "../../src/models/session.model";

import mongoConnect from "../../src/database/mongo-connect.database";

const startTime = new Date(2019, 5,6,18,22);

const endTime = new Date(2019, 5,6,19,22);

const sessionForAdd = new Session("Tor", startTime, endTime,10, 10, 2);

beforeEach(async () =>{

await mongoConnect('cinema-test').then(() => console.log('MongoDB connected'));

});

it('should be pass addSession', async () => {

const add = await SessionService.addSession(sessionForAdd).then(tolk => console.log('Session save'));

});

it('should be pass getSessionById', async () => {

const session = await SessionService.getSessionById('5cd1260d1129770934e14e93');

expect(session.filmName).toEqual(sessionForAdd.filmName);

});

it('should be pass getAllSessionsByFilmName', async () => {

const sessions = await SessionService.getAllSessionsByFilmName("Tor");

expect(sessions[0].filmName).toEqual(sessionForAdd.filmName);

});

it('should be pass getAllSessionsByFilmName', async () => {

const sessions = await SessionService.getAllSessionsByFilmName("Tor");

expect(sessions).toEqual(sessionForAdd.filmName);

});

it('should be pass uncorrect filmName getAllSessionsByFilmName', async () => {

const sessions = await SessionService.getAllSessionsByFilmName("For");

expect(sessions).toEqual([]);

});

it('should be true isExistSession', async () => {

sessionForAdd.id = '5cd186655e18b025a4d6d7de';

const sessions = await SessionService.isExistSession(sessionForAdd);

expect(sessions).toBe(true);

});

it('should be false isExistSession', async () => {

sessionForAdd.id = '5cd186655e18b025a4d6d7df';

const sessions = await SessionService.isExistSession(sessionForAdd);

expect(sessions).toBe(false);

});

// removeSession

it('should be pass removeSession', async () => {

sessionForAdd.id = '5cd186655e18b025a4d6d7de';

const sessions = await SessionService. removeSession(sessionForAdd);

expect(sessions).toBe(null);

});

Файл session.validator.spec.ts

import mongoConnect from "../../src/database/mongo-connect.database";

import {SessionValidators} from "../../src/validators/session.validators";

import Session from "../../src/models/session.model";

const startTime = new Date(2019, 5,6,18,22);

const endTime = new Date(2019, 5,6,19,22);

const sessionForAdd = new Session("Tor", startTime, endTime,10, 10, 2);

beforeEach(async () =>{

// await SessionService.addSession(sessionForAdd);

await mongoConnect('cinema-test').then(() => console.log('MongoDB connected'));

});

it('session validator validateSessionToAddByDataFields', async () => {

const session = new Session("Tor", startTime, endTime,10, 10, 2);

const add = await SessionValidators.validateSessionToAddByDataFields(session);

expect(add).toEqual([]);

});

it('session validator validateSessionToAddByDataFields without film name', async () => {

const session = new Session(null, startTime, endTime,10, 10, 2);

const add = await SessionValidators.validateSessionToAddByDataFields(session);

expect(add).toEqual(['Film name is required']);

});

it('session validator validateSessionToAddByDataFields without start time', async () => {

const session = new Session('Tor', null, endTime,10, 10, 2);

const add = await SessionValidators.validateSessionToAddByDataFields(session);

expect(add).toEqual(['Start time is required']);

});

it('session validator validateSessionToAddByDataFields without end time', async () => {

const session = new Session("Tor", startTime, null,10, 10, 2);

const add = await SessionValidators.validateSessionToAddByDataFields(session);

expect(add).toEqual(['End time is required']);

});

it('session validator validateSessionToAddByDataFields without rows', async () => {

const session = new Session("Tor", startTime, endTime,null, 10, 2);

const add = await SessionValidators.validateSessionToAddByDataFields(session);

expect(add).toEqual(['Rows value is required']);

});

it('session validator validateSessionToAddByDataFields without columns', async () => {

const session = new Session("Tor", startTime, endTime,10, null, 2);

const add = await SessionValidators.validateSessionToAddByDataFields(session);

expect(add).toEqual(['Columns value is required']);

});

it('session validator validateSessionToAddByDataFields without ticket price', async () => {

const session = new Session("Tor", startTime, endTime,10, 10, null);

const add = await SessionValidators.validateSessionToAddByDataFields(session);

expect(add).toEqual(['Ticket price is required']);

});

it('session validator validateSessionToAddByDataFields session = null', async () => {

const add = await SessionValidators.validateSessionToAddByDataFields(null);

expect(add).toEqual(['Session is required']);

});

it('session validator validateSessionToRemove session ', async () => {

const add = await SessionValidators.validateSessionToRemove(sessionForAdd);

expect(add).toEqual(['Id is required']);

});

it('session validator validateSessionToRemove session ', async () => {

const add = await SessionValidators.validateSessionToRemove(null);

expect(add).toEqual(['Session is required']);

});

Папка user-test

Файл user.service.spec.ts:

import mongoConnect from "../../src/database/mongo-connect.database";

import UserService from "../../src/services/user.service";

import User from "../../src/models/user.model";

import {UserRoleEnum} from "../../src/enums/user-role.enum";

const user = new User('diana','Diana', '12345', UserRoleEnum.admin);

beforeEach(async () =>{

await mongoConnect('cinema-test').then(() => console.log('MongoDB connected'));

});

//add user

it('should add user', async () => {

await UserService.saveUser(user).then(() => console.log('Save user'));

});

it('not should add user', async () => {

const save = await UserService.saveUser(null).then(() => console.log("User is null"));

});

// getUserById

it('should getUserById', async () => {

const get = await UserService.getUserById('5cd1a31ba3e7b7245815b279');

expect(get.name).toEqual('Diana');

});

it('not should wrong id getUserById ', async () => {

const get = await UserService.getUserById('5cd1a31ba3e7b7245815b179');

expect(get).toEqual(null);

});

it('not should getUserById', async () => {

const get = await UserService.getUserById(null);

expect(get).toEqual(null);

});

// getUserByUsername

it('should getUserByUsername',async () => {

const get = await UserService.getUserByUsername('diana');

expect(get.name).toEqual('Diana');

});

it('not should getUserByUsername wrong name',async () => {

const get = await UserService.getUserByUsername('dia');

expect(get).toEqual(null);

});

it('not should getUserByUsername name is null',async () => {

const get = await UserService.getUserByUsername(null);

expect(get).toEqual(null);

});

//isUserExist

it('should isUserExist',async () => {

const isExist = await UserService.isUserExist('diana', '12345');

expect(isExist).toBe(true);

});

it('should false not exist user', async () => {

const isExist = await UserService.isUserExist('diana', '12345523');

expect(isExist).toBe(false);

});

it('should false (user = null) ', async () => {

const isExist = await UserService.isUserExist(null, null);

expect(isExist).toEqual(false);

});

// isUsernameFree

it('should is Username not free', async () => {

const isFree = await UserService.isUsernameFree('diana');

expect(isFree).toBe(false);

});

it('should is Username free', async () => {

const isFree = await UserService.isUsernameFree('dia');

expect(isFree).toBe(true);

});

it('should failed username = null', async () => {

const isFree = await UserService.isUsernameFree(null);

expect(isFree).toBe(false);

});

Файл user.validator.spec.ts

import mongoConnect from "../../src/database/mongo-connect.database";

import User from "../../src/models/user.model";

import {UserRoleEnum} from "../../src/enums/user-role.enum";

import {UserValidators} from "../../src/validators/user.validators";

const user = new User('diana','Diana', '12345', UserRoleEnum.admin);

beforeEach(async () =>{

await mongoConnect('cinema-test').then(() => console.log('MongoDB connected'));

});

it('should pass validateUserToSignIn ', async () => {

const people = new User('diana', 'Diana', '12345', UserRoleEnum.admin);

const user = await UserValidators.validateUserToSignIn(people);

expect(user).toEqual([]);

});

it('should pass validateUserToSignIn not correct data', async () => {

const people = new User("dianaka", "dianaka", "12345", UserRoleEnum.user);

const user = await UserValidators.validateUserToSignIn(people);

expect(user).toEqual(['Invalid username or password']);

});

it('should pass validateUserToSignIn without username', async () => {

const people = new User(null, "dianaka", "12345", UserRoleEnum.user);

const user = await UserValidators.validateUserToSignIn(people);

expect(user).toEqual(['Username is required']);

});

it('should pass validateUserToSignIn without password', async () => {

const people = new User("dianaka", "dianaka", null, UserRoleEnum.user);

const user = await UserValidators.validateUserToSignIn(people);

expect(user).toEqual(['Password is required']);

});

it('should signUp', async () => {

const people = new User("dianaka", "dianaka", "12345", UserRoleEnum.user);

const user = await UserValidators.validateUserToSignUp(people);

expect(user).toEqual([]);

});

it('should signUp without username', async () => {

const people = new User(null, "dianaka", "12345", UserRoleEnum.user);

const user = await UserValidators.validateUserToSignUp(people);

expect(user).toEqual(['Username is required']);

});

it('should signUp without name', async () => {

const people = new User('dianka', null, "12345", UserRoleEnum.user);

const user = await UserValidators.validateUserToSignUp(people);

expect(user).toEqual(['Name is required']);

});

it('should signUp without password', async () => {

const people = new User('dianka', "di", null, UserRoleEnum.user);

const user = await UserValidators.validateUserToSignUp(people);

expect(user).toEqual(['Password is required']);

});

it('should signUp token name', async () => {

const people = new User('diana', "di", '1234', UserRoleEnum.user);

const user = await UserValidators.validateUserToSignUp(people);

expect(user).toEqual(['Username is already taken']);

});