|  |  |
| --- | --- |
| https://lh7-us.googleusercontent.com/QuBaagabzZYLr7U3ZbYOtMusd5FbWWxIvMUTN8jrLHzSLMg534z9gXRTIG1Us4i_lOwmWlaBxKedNt-SQ26dm4WmyqwjGDmEO6z8GE3QrZosqvHM88J2EFeVf1u0GzyCZQlhWmp1Zeo85tKo4LJVXQ | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» |

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

Отчёт по курсовой работе  
по учебной дисциплине «Web-технологии»

на тему «Веб-приложение на стеке PERN»

Выполнил  
студент группы М3О-119БВ-24

Нарзиев А.Т.

Приняла

Иванова П.М.

Москва  
2024

# Содержание

[Содержание 2](#_Toc197971593)

[Задание 3](#_Toc197971594)

[Введение 4](#_Toc197971595)

[Разработка приложения 5](#_Toc197971596)

[1. Описание используемых технологий 5](#_Toc197971597)

[1.1. Клиентская часть 5](#_Toc197971598)

[1.2. Серверная часть 5](#_Toc197971599)

[1.3. Инструменты 5](#_Toc197971600)

[2. Разработка архитектуры приложения 6](#_Toc197971601)

[3.1. Архитектура клиента 6](#_Toc197971604)

[3.2. Архитектура сервера 7](#_Toc197971605)

[3.3. Описание структуры базы данных 8](#_Toc197971606)

[3. Реализация приложения 11](#_Toc197971607)

[4. Тестирование приложения 11](#_Toc197971608)

[4.1. План тестирования 11](#_Toc197971613)

[4.2. Процесс и результаты тестирования (приложить скрины процесса тестирование) 11](#_Toc197971614)

[Заключение 11](#_Toc197971615)

[Список источников 11](#_Toc197971616)

[Приложение 11](#_Toc197971617)

# Задание

Используя концепцию CRUD (Create, Read, Update, Delete), разработать single-page application, представляющее собой список / витрину уникальных сущностей.

В приложении должен быть реализован функционал:

добавления и удаления сущностей,

просмотра сущностей,

редактирования сущностей.

Под «сущностями» могут пониматься, например, видеоигровые персонажи, преподаватели, музеи, книги, памятные фотографии и так далее.

По желанию:

реализовать систему «ролей». Основные роли:

пользователь (доступно только чтение)

администратор (создание, редактирование, удаление).

Для ограничения доступа к функционалу администратора необходимо реализовать систему авторизации и аутентификации.

предусмотреть механизм валидации входящих данных на сторонах сервера и клиента.

При реализации валидации заполнения форм на клиенте предусмотреть выделение полей ввода с ошибкой, а также автоматическую прокрутку страницы к первому полю с ошибкой. (для реализации валидации можно использовать Redux-toolkit, Final-form и т.д.)

Поиск сущностей, а также их сортировка по разным критериям (поиск и сортировка должны быть оптимальными по времени)

В процессе разработки необходимо использовать стек технологий PERN:

PostgreSQL,

Express,

React,

Node.js.

Разработка ведётся индивидуально. Тема должна быть заранее представлена студентами для согласования.

Критерии успешной защиты:

самостоятельный выбор темы,

индивидуальная разработка без использования готовых решений,

освоение принципов CRUD и стека технологий PERN.

Допустимо использование фреймворков (Django, Flask и др.), но это противоречит критериям успешной защиты → такие работы будут оцениваться ниже.

Срок сдачи: 17 мая 2025 года.

Базовые материалы:

Что такое CRUD? — <https://wiki.merionet.ru/articles/chto-takoe-crud?ysclid=m67jxktsis524763230>

Про CRUD на Wikipedia — <https://ru.wikipedia.org/wiki/CRUD>

What is PERN Stack? — <https://www.geeksforgeeks.org/what-is-pern-stack/?ysclid=m67k0kc7z6103820203>

Learn the PERN Stack by building a web app — <https://youtu.be/ldYcgPKEZC8?si=L4JasetsP7RFoe45>

# Введение

Современный цифровой ландшафт немыслим без веб-приложений, которые стали основным инструментом для решения бизнес-задач, взаимодействия с пользователями и автоматизации процессов. Разработка таких приложений требует не только глубоких технических знаний, но и грамотного выбора технологического стека, обеспечивающего масштабируемость, производительность и удобство поддержки. В этой связи стек PERN (PostgreSQL, Express.js, React, Node.js) занимает особое место, объединяя проверенные решения для back-end и современные подходы к front-end, что делает его одним из наиболее востребованных инструментов в веб-разработке.

Актуальность темы обусловлена растущим спросом на Fullstack-разработчиков, способных создавать полноценные приложения «под ключ», а также преимуществами PERN-стека:

- Использование **JavaScript** на всех этапах (от клиента до сервера) упрощает разработку и снижает порог вхождения.

- **React** обеспечивает динамический и отзывчивый интерфейс за счёт компонентного подхода.

- **Node.js** и **Express.js** позволяют строить высокопроизводительный back-end с минимальными затратами ресурсов.

- **PostgreSQL** гарантирует надёжное хранение данных и гибкость в работе со сложными запросами.

Цель:

разработка веб-приложения на стеке PERN, демонстрирующего взаимодействие между клиентской и серверной частями, а также эффективное управление данными. Для её достижения поставлены следующие задачи:

1. Исследовать особенности технологий PERN-стека и их интеграции.

2. Спроектировать архитектуру приложения, включая схемы баз данных и API.

3. Реализовать клиентскую часть на React с использованием современных hooks и state-менеджмента.

4. Разработать серверную логику на Node.js и Express.js с RESTful API.

5. Органировать взаимодействие с базой данных PostgreSQL через ORM-инструменты.

6. Провести тестирование системы, оценив её производительность и безопасность.

Практическая значимость работы заключается в приобретении навыков Fullstack-разработки, которые соответствуют текущим требованиям IT-рынка. Результатом проекта станет готовое веб-приложение, сопровождаемое документацией, а также выводы о преимуществах и ограничениях выбранного стека.

Структура отчета отражает этапы разработки: анализ технологий, проектирование, реализация, тестирование и заключение. Во введении обоснован выбор темы, сформулированы цели и задачи, а также обозначена ценность проведённой работы для дальнейшего профессионального развития.

# Разработка приложения

## Описание используемых технологий

### Клиентская часть:

**React** — библиотека для построения пользовательского интерфейса.

**React Router** — маршрутизация между страницами (Login, Profile, Movies).

**React Final Form** — управление формами (регистрация, вход, редактирование фильмов).

**Context API** — глобальное состояние (аутентификация, уведомления).

**Axios** — HTTP-клиент для взаимодействия с API.

**Jest** — фреймворк для тестирования JavaScript, используется для написания и запуска тестов.

**React Testing Library** — библиотека для тестирования компонентов React, упрощает взаимодействие с компонентами в тестах.

### Серверная часть:

**Express.js** — фреймворк для создания RESTful API.

**PostgreSQL** — реляционная база данных.

**JWT (JSON Web Tokens)** — аутентификация через токены.

**CORS** — безопасное межресурсное взаимодействие.

**Bcrypt** — хеширование паролей.

### Инструменты:

**PlantUML** — визуализация схемы базы данных.

**PostgreSQL Indexes** — оптимизация запросов.

**Jest Mocking** — возможность создания моков и шпионов для тестирования зависимостей и функций.

## Разработка архитектуры приложения

В этой части описана структура разрабатываемого приложения.

### Архитектура клиента

#### Описание файловой структуры

Client/

├── src/

│ ├── api/ # Настройка Axios

│ ├── app/ # Redux store (если используется)

│ ├── components/ # Переиспользуемые компоненты (Notification, MovieCard)

│ ├── context/ # Глобальные контексты (Auth, Notification)

│ ├── pages/ # Страницы приложения:

│ │ ├── Auth/ # Формы входа/регистрации

│ │ ├── MovieForm/ # Форма создания/редактирования фильмов

│ │ ├── Profile/ # Страница профиля пользователя

│ │ └── ...

│ ├── services/ # Сервисы для работы с API (MovieService)

│ ├── slices/ # Redux-слайсы (AuthSlice)

│ ├── App.js # Главный компонент приложения

│ └── index.js # Главный файл приложения

└── package-lock.json # Файл описания версий компонентов

#### Описание функционала страниц

1. Login/Registration

**Аутентификация через JWT:**

Отправка данных на /api/auth/login и /api/auth/register.

Сохранение токенов в cookies.

**Валидация полей:** Проверка на пустые значения, формат email.

2. Profile

**Управление данными:**

Редактирование никнейма и email через /api/users/me.

**Просмотр билетов:** Запрос к /api/users/me/tickets (не реализовано в текущем коде).

3. Movies

**Фильтрация:** По жанру, году через /api/movies/search?genre=1&yearFrom=2000.

**CRUD для админов:**

**Создание:** POST /api/movies.

**Редактирование:** PUT /api/movies/:id.

#### Описание связей между страницами

!

### Архитектура сервера

#### Описание структуры файлов сервера

Server/

├── config/

│ └── db.js # Подключение к PostgreSQL

├── controllers/

│ ├── authController.js # Логика аутентификации

│ └── movieController.js # Управление фильмами

├── middlewares/

│ ├── authMiddleware.js # Проверка JWT

│ └── isAdmin.js # Проверка роли администратора

├── routes/

│ └── api/

│ ├── auth.js # Маршруты аутентификации

│ └── movies.js # Маршруты для фильмов

└── ...

#### Описание API

Аутентификация

**POST /api/auth/register** — регистрация пользователя.

**POST /api/auth/login** — вход, установка cookies с токенами.

**POST /api/auth/logout** — выход, удаление cookies.

Пользователи

**GET /api/users/me** — данные текущего пользователя.

**PUT /api/users/me** — обновление профиля.

**DELETE /api/users/me** — удаление аккаунта.

Фильмы

**GET /api/movies** — список всех фильмов.

GET /api/movies/search — поиск с фильтрами.

**POST /api/movies** — создание фильма (админ).

**PUT /api/movies/:id** — обновление фильма (админ).

**DELETE /api/movies/:id** — удаление фильма (админ).

Жанры

**GET /api/genres** — список всех жанров.

### Описание структуры базы данных

1. Основные таблицы

| Таблица | Описание | Ключевые поля |
| --- | --- | --- |
| user\_levels | Уровни пользователей (VIP, Киномагнат и т.д.) | id (PK), name, description |
| users | Данные пользователей | id (PK), level\_id (FK → user\_levels), credits (баланс) |
| movies | Информация о фильмах | id (PK), title, release\_year, description, trailerid |
| genres | Жанры фильмов | id (PK), name (UNIQUE) |
| halls | Кинозалы | id (PK), name, seat\_layout (JSON) |
| screenings | Сеансы показов | id (PK), movie\_id (FK → movies), hall\_id (FK → halls), start\_time |
| seats | Места в залах | id (PK), hall\_id (FK → halls), row, number |
| tickets | Билеты пользователей | id (PK, UUID), user\_id (FK → users), screening\_id (FK → screenings) |
| ratings | Рейтинги фильмов | id (PK), movie\_id (FK → movies), rating (0-10), created\_at |

2. Типы связей

2.1. Один-ко-многим (1:N)

| Связь | Описание | Реализация |
| --- | --- | --- |
| user\_levels ↔ users | Один уровень может быть у многих пользователей | users.level\_id → user\_levels.id |
| halls ↔ screenings | В одном зале проходит много сеансов | screenings.hall\_id → halls.id |
| movies ↔ screenings | Один фильм показывается в нескольких сеансах | screenings.movie\_id → movies.id |
| halls ↔ seats | В одном зале много мест | seats.hall\_id → halls.id |
| users ↔ tickets | Один пользователь покупает много билетов | tickets.user\_id → users.id |

2.2. Многие-ко-многим (N:M)

| Связь | Описание | Реализация |
| --- | --- | --- |
| movies ↔ genres | Фильм может принадлежать к нескольким жанрам, жанр — к нескольким фильмам | Через таблицу movie\_genres: movie\_genres.movie\_id → movies.id movie\_genres.genre\_id → genres.id |

3. Промежуточные таблицы

| Таблица | Назначение | Пример использования |
| --- | --- | --- |
| movie\_genres | Связывает фильмы с их жанрами | Фильм "Интерстеллар" связан с жанрами "Фантастика" и "Драма" |
| tickets | Связывает пользователей, сеансы и места | Билет пользователя на конкретный сеанс и место |

4. Оптимизация

Индексы

**-- sql**

-- Для быстрого поиска фильмов:

CREATE INDEX idx\_movies\_title ON movies (title text\_pattern\_ops);

CREATE INDEX idx\_movies\_year ON movies (release\_year);

**-- sql**

-- Для связи фильмов и жанров:

CREATE INDEX idx\_movie\_genres ON movie\_genres (genre\_id);

**-- sql**

-- Для работы с билетами:

CREATE INDEX idx\_tickets\_user ON tickets (user\_id);

5. Ограничения

**UNIQUE** для genres.name (жанры не повторяются).

**CHECK** для ratings.rating (значение от 0 до 10).

**Каскадное удаление** для зависимых записей (например, при удалении фильма удаляются все связанные сеансы).

6. Группировка таблиц по модулям

|  |  |
| --- | --- |
| **Модуль** | **Таблицы** |
| **User System** | user\_levels, users |
| **Movie System** | movies, genres, movie\_genres, ratings |
| **Cinema System** | halls, screenings, seats, tickets |

3. Схема базы данных

!

## Реализация приложения

Описание процесса реализации приложения

## Тестирование приложения

### План тестирования

#### Тестирование серверной части

**Тестируемые компоненты:**

* Маршруты: /api/auth, /api/movies
* Middleware: authMiddleware, isAdmin
* Контроллеры: authController, movieController

**Тестовые сценарии:**

1. **Регистрация:**
   * Запрос с корректными данными → 201 Created.
   * Повторная регистрация → 409 Conflict.
   * Регистрация без соответствующих обязательных полей → 400 Bad Request.
2. **Авторизация:**
   * Неверный пароль или Логин → 401 Unauthorized.
   * Успешный вход → JWT в ответе.
3. **Доступ к /api/movies:**
   * Без JWT → 403 Forbidden.
   * С ролью "user" → 200 OK (только GET).
   * С ролью "admin" → Доступ к POST/PUT/DELETE.
4. **CRUD для фильмов:**
   * Создание фильма без соответствующих обязательных полей → 400 Bad Request.
   * Удаление несуществующего фильма → 404 Not Found.

#### Тестирование клиентской части

**Тестируемые компоненты:**

* Страницы: Auth, MovieForm, Profile
* Компоненты: Notification, MovieCard
* Контексты: Auth, Notification
* Redux: AuthSlice, MovieService

#### Тестирование компонентов с помощью Jest

1. Тестирование компонента карточки фильма

Сначала рассмотрим компонент MovieFormPage. Он использует react-final-form для управления формой, есть валидация полей. Нужно проверить, что форма правильно отображает поля, показывает ошибки валидации и корректно отправляет данные.

Для тестирования формы, нам нужно:

* Смоделировать рендер компонента.
* Проверить наличие всех необходимых полей: название, описание, год выпуска, идентификатор трейлера, выбор жанров.
* Проверить валидацию: при пустых полях должны появляться сообщения об ошибках.
* Проверить, что при отправке формы вызывается правильный API-метод (createMovie или updateMovie).

**Порядок действий**

Для теста валидации:

1. Рендерим форму.
2. Нажимаем кнопку отправки без заполнения полей.
3. Проверяем, что отображаются сообщения об ошибках для каждого обязательного поля.

Для теста успешного создания фильма:

1. Заполняем все поля корректными данными.
2. Симулируем отправку формы.
3. Проверяем, что вызвался createMovie с правильными данными.
4. Проверяем, что произошел переход на страницу /movies.

2. Тестирование формы редактирования фильма

Подробнее о компоненте карточки фильма (MovieListPage). Каждая карточка отображает информацию о фильме, постер, кнопки для админа. Нужно проверить:

* Корректное отображение данных фильма: название, описание, изображение.
* Отображение кнопок редактирования и удаления только для админа.
* Проверку обработки кликов: переход на страницу фильма, открытие трейлера, вызов функций удаления и редактирования.

**Порядок действий**

1. Мокируем API-вызовы (fetch, getMovieById, createMovie и т.д.).
2. Рендерим компонент с переданными пропсами фильма.
3. Проверяем отображение названия, описания, изображения.
4. Эмулируем авторизованного пользователя (админа и обычного).
5. Если пользователь админ, проверяем наличие кнопок редактирования и удаления.
6. Симулируем клик на кнопке удаления и проверяем вызов соответствующей функции с правильным ID.

Также учитываем различные состояния: загрузка, ошибки, отсутствие изображения у фильма.

3. Итог

Итоговая структура включает:

* Описание тестовых случаев.
* Настройку моков перед каждым тестом.
* Рендер компонента.
* Взаимодействие с элементами (ввод текста, клики).
* Проверки (утверждения) на наличие элементов, вызовы функций, изменения состояния.

#### Тестовые сценарии

**Тестовые сценарии:**

| **Сценарий** | **Роль** | **Ожидаемый результат** | **Скриншоты** |
| --- | --- | --- | --- |
| 1. Регистрация нового пользователя | Гость | Уведомление об успехе, редирект на вход | Форма регистрации, уведомление |
| 2. Вход с неверными данными | Гость | Ошибка в Notification | Форма входа с ошибкой |
| 3. Просмотр списка фильмов | Все | Отображение MovieCard для каждого фильма | Главная страница с карточками |
| 4. Попытка создать фильм (не админ) | Пользователь | Ошибка "Доступ запрещен" | Уведомление, скрытая форма |
| 5. Создание фильма (админ) | Администратор | Фильм появляется в списке | Форма MovieForm, успешное уведомление |
| 6. Редактирование профиля | Пользователь | Обновленные данные в Profile | Страница Profile до/после |
| 7. Выход из системы | Авторизованный | Редирект на страницу входа | Главная страница после выхода |

### Процесс и результаты тестирования

#### Тестирование серверной части

**!**

#### Тестирование клиентской части

1. Тестирование компонента карточки фильма

**UniTest для компонента карточки фильма (MovieCard.test.js):  
!**

2. Тестирование формы  создания/редактирования фильма

**UniTest для формы создания/редактирования фильма (MovieFormPage.test.js):**

import React from 'react';

import { render, screen, fireEvent, waitFor } from '@testing-library/react';

import { Form } from 'react-final-form';

import MovieFormPage from './MovieFormPage';

import { useAuth } from '../../context/AuthContext';

import { useNotification } from '../../context/NotificationContext';

// Мокируем зависимости

jest.mock('../../context/AuthContext');

jest.mock('../../context/NotificationContext');

jest.mock('../../services/movieService');

const mockNavigate = jest.fn();

jest.mock('react-router-dom', () => ({

  ...jest.requireActual('react-router-dom'),

  useNavigate: () => mockNavigate,

  useParams: () => ({ id: null }),

}));

describe('MovieFormPage Component', () => {

    const mockUser = { role: 'admin' };

    const mockShowNotification = jest.fn();

    beforeEach(() => {

        useAuth.mockReturnValue({ user: mockUser });

        useNotification.mockReturnValue({ showNotification: mockShowNotification });

    });

    test('renders form with all fields', async () => {

        render(<MovieFormPage />);

        expect(screen.getByLabelText('Название')).toBeInTheDocument();

        expect(screen.getByLabelText('Описание')).toBeInTheDocument();

        expect(screen.getByLabelText('Год выпуска')).toBeInTheDocument();

        expect(screen.getByLabelText('Идентификатор трейлера')).toBeInTheDocument();

        expect(screen.getByText('Жанры')).toBeInTheDocument();

    });

    test('shows validation errors', async () => {

        render(<MovieFormPage />);

        fireEvent.click(screen.getByText('Создать'));

        await waitFor(() => {

            expect(screen.getByText('Название обязательно')).toBeInTheDocument();

            expect(screen.getByText('Описание обязательно')).toBeInTheDocument();

            expect(screen.getByText('Идентификатор трейлера обязателен')).toBeInTheDocument();

            expect(screen.getByText('Выберите хотя бы один жанр')).toBeInTheDocument();

        });

    });

    test('submits valid form', async () => {

        const { getByLabelText } = render(<MovieFormPage />);

        // Заполняем форму

        fireEvent.change(getByLabelText('Название'), { target: { value: 'Test Movie' } });

        fireEvent.change(getByLabelText('Описание'), { target: { value: 'Test Description' } });

        fireEvent.change(getByLabelText('Год выпуска'), { target: { value: '2020' } });

        fireEvent.change(getByLabelText('Идентификатор трейлера'), { target: { value: '123' } });

        // Выбираем жанр

        const firstCheckbox = screen.getAllByRole('checkbox')[0];

        fireEvent.click(firstCheckbox);

        fireEvent.click(screen.getByText('Создать'));

        await waitFor(() => {

          expect(mockNavigate).toHaveBeenCalledWith('/movies');

        });

    });

});

**Скриншоты результата:**

**!**

# Заключение

**!**

# Список источников

**!**

# Приложение

**!**