МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА API-ЦЕНТРИЧНОГО ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ПОТОКОВОГО ПРОСМОТРА ФИЛЬМОВ С ИСПОЛЬЗОВАНИЕМ ПРИНЦИПОВ РЕАКТИВНОГО ПРОГРАММИРОВАНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Асербекова Куанышгали Олеговича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2023

Введение

В настоящее время потребность в просмотре фильмов и сериалов онлайн становится все более актуальной. Существует множество сайтов и приложений для просмотра фильмов, но не все они имеют удобный и интуитивно понятный интерфейс, а также хорошо оптимизированы под различные устройства. Выбор темы обусловлен несколькими факторами.

— Интернет-сервисы для просмотра фильмов набирают популярность среди пользователей во всем мире. Интернет-кинотеатры и стриминговые платформы предоставляют возможность смотреть фильмы и сериалы в любое удобное время и место, что делает их более привлекательными, чем традиционные кинотеатры.

— Создание собственного веб-сервиса для просмотра фильмов является актуальной задачей для разработчиков. Такой проект может быть полезен как для личного использования, так и для коммерческих целей.

— Использование фреймворка Django и библиотек React, TypeScript, Material UI и MobX позволяет создать высококачественное и удобное в использовании веб-приложение. Django обладает богатым набором функций и удобным API, что упрощает разработку серверной части приложения. React, TypeScript, Material UI и MobX, в свою очередь, позволяют создать красивый и удобный пользовательский интерфейс, а также облегчить управление состоянием компонентов.

В целом, создание сайта просмотра фильмов на базе фреймворка Django и библиотек React, TypeScript, Material UI и MobX является актуальной темой для дипломной работы, которая позволит студенту показать свои навыки в разработке веб-приложений, а также применить современные технологии и инструменты.

Цели работы:

1. Создать полнофункциональный сайт для просмотра фильмов, который обеспечивает удобный пользовательский интерфейс и привлекательный дизайн.
2. Разработать сайт, который имеет масштабируемую архитектуру и может обрабатывать большое количество данных, без потери производительности.
3. Реализовать сайт с использованием современных технологий, таких как

Django, React, TypeScript, Material UI и MobX.

Задачи работы:

1. Изучить требования пользователя и разработать дизайн интерфейса сайта, который удовлетворяет их потребности.
2. Разработать бэкенд сайта на Django, который обеспечивает безопасность, надежность и масштабируемость.
3. Создать фронтенд сайта на React с использованием TypeScript, Material UI и MobX, который обеспечивает удобный интерфейс и высокую производительность.
4. Реализовать возможность регистрации и авторизации пользователей, чтобы они могли сохранять свои настройки и просмотренные фильмы.
5. Разработать систему поиска фильмов, которая позволяет пользователям легко находить интересующие их фильмы.
6. Обеспечить интеграцию сайта с базами данных фильмов, такими как IMDb и Кинопоиск, чтобы пользователи могли получить достоверную информацию о фильмах.
7. Обеспечить возможность добавления фильмов в избранное и просмотра списка избранных фильмов.
8. Протестировать и оптимизировать сайт для обеспечения максимальной производительности и безопасности.

Выпускная квалификационная работа состоит из введения, 2 глав, заключения, списка использованных источников, включающего 20 наименований, работа изложена на 59 листах текста, с приведенным кодом программного продукта. Далее приведены наименования глав:

1. Описание теоретической области
2. Описание предметной области

Основное содержание работы

В первой главе рассматриваются технологии, инструменты и подходы, которые использовались в процессе разработки приложения.

1. Django — в этой главе описывается фреймворк Django и его преимущества при разработке веб-приложений. Django предоставляет мощный функционал "из коробки ускоряющий разработку и упрощающий проект. Фреймворк обладает модульной архитектурой, широкой поддержкой сторонних библиотек и интеграцией с различными системами управления базами данных. В конкретном проекте создания сайта просмотра фильмов используется Django REST Framework (DRF) для обеспечения связи между фронтендом (React) и бэкендом (Django). DRF предоставляет API для работы с данными фильмов, пользовательских профилей и других сущностей системы. Он облегчает разработку клиент-серверного взаимодействия, упрощая обработку запросов и ответов между клиентом и сервером. DRF предлагает готовые компоненты, гибкую настройку, поддержку различных схем аутентификации и авторизации, а также совместимость с различными форматами данных и принципами REST для интеграции с другими системами и сервисами.
2. React — в данной главе описывается React - открытая JavaScript-библиотека, разработанная Facebook для создания пользовательских интерфейсов веб-приложений. React является одной из самых популярных и востребованных технологий в веб-разработке благодаря своим возможностям, гибкости и производительности. Основные концепции React включают компоненты, JSX, виртуальный DOM, однонаправленный поток данных, интеграцию с другими библиотеками и фреймворками. Компонентный подход React позволяет разделять интерфейс на независимые и переиспользуемые компоненты, упрощая разработку, тестирование и поддержку кода.
3. TypeScript — в данной главе рассматривается значимость выбора подходящего инструмента разработки веб-приложений в контексте цифровых технологий. Описывается роль TypeScript как надстройки над JavaScript, которая предоставляет усовершенствованные функции типизации для создания более безопасных и надежных приложений. В дипломном проекте по созданию приложения для просмотра фильмов был использован

TypeScript по ряду причин. Первая причина - статическая типизация. TypeScript вводит систему статической типизации в JavaScript, что уменьшает количество ошибок на этапе выполнения кода и повышает надежность приложения. Вторая причина - ясность и поддерживаемость кода. Система типов TypeScript делает код более понятным, что облегчает работу в команде и упрощает поддержку кода в долгосрочной перспективе. Третья причина - инструменты разработки. TypeScript улучшает автодополнение кода, проверку типов на лету и другие функции сред разработки.

4. MobX — в данной главе описывается важность управления состоянием приложения, особенно в случае создания крупномасштабных или динамических веб-приложений, таких как приложение для просмотра фильмов. Для эффективного управления состоянием приложения используется библиотека MobX, основанная на реактивной функциональной парадигме.
5. Styled-components — в данной главе описывается библиотека styled-components, используемая для стилизации React-приложений. Эта библиотека позволяет применять стили непосредственно к компонентам, упрощая процесс управления и поддержки стилей в приложении. Основные особенности и преимущества styled-components, которые особенно полезны, включают использование теговых шаблонов ES6 для стилизации компонентов, что позволяет писать стандартный CSS прямо в JavaScript коде. Использование пропсов и тем для динамической стилизации компонентов, что обеспечивает гибкость в изменении стилей в зависимости от пропсов или текущей темы приложения. Каждый стилизованный компонент имеет свои собственные стили, которые не влияют на другие компоненты, что избавляет от проблем с каскадностью и переопределением стилей. styled-components автоматически управляет вставкой и удалением стилей, что помогает оптимизировать производительность приложения.
6. Material-UI — в данной главе рассматривается выбор использования библиотеки Material UI (Mui) в дипломном проекте. Material UI является популярной библиотекой компонентов пользовательского интерфейса для React. Ее выбор обусловлен несколькими факторами. Во-первых, Material UI обладает высокой стабильностью и надежностью, так как активно поддерживается и обновляется. Она также имеет большую базу пользователей и активное сообщество, что облегчает процесс разработки. Во-вторых,

библиотека предлагает уникальный дизайн, основанный на принципах Google's Material Design, что обеспечивает привлекательный внешний вид и отличный пользовательский опыт. Она также обеспечивает единообразие и последовательность в оформлении приложения. В-третьих, Material UI гибкая и настраиваемая, позволяя легко настраивать компоненты под конкретные требования проекта.

7. PostgreSQL — в данной главе рассматривается выбор системы управления базами данных (СУБД) для веб-приложения. PostgreSQL (Postgres) был выбран в качестве основной СУБД для дипломного проекта по просмотру фильмов. Postgres является мощной и надежной реляционной СУБД, предоставляющей широкий набор функциональных возможностей.
8. основные принципы и методы создания эффективного пользовательского интерфейса, такие как простота и интуитивность, модульность, разделение обязанностей, оптимизация производительности.

Ниже представлены основные моменты по реализации серверной части, которые будут подробно рассмотрены

1. Описание моделей — в данной главе описывается процесс создания сервера API на Django. Начинается с установки и настройки Django на компьютере с помощью инструмента pip. Затем создается новый проект Django с помощью команды "django-admin startproject projectname". Далее создается новое приложение Django с помощью команды "python manage.py startapp appname". Затем описываются модели, которые будут использоваться в API. Для этого обновляется файл models.py внутри Django приложения, где определяются поля моделей, их типы данных и связи между моделями. В данном случае, описываются модели User и Films с их соответствующими полями и типами данных. Класс User наследуется от базового класса models.Model и имеет дополнительные поля, такие как gender, age, occupation и т.д. Класс Films также наследуется от базового класса models.Model и имеет поля, такие как film_id_api, rating_kp, rating_imdb и т.д. Каждая модель имеет внутренний класс Meta, в котором определяется имя таблицы базы данных, связанной с моделью.
2. Создание сериализаторов — в данной главе описывается создание сериализаторов для сервера API на Django. Сериализаторы определяют, как данные моделей будут представлены в формате JSON, используемом для

передачи данных между клиентом и сервером в API. В файле serializers.py внутри Django приложения определяются классы сериализаторов для каждой модели.

3. Создание представлений — в данной главе описывается создание представлений для сервера API на Django. Первым шагом создается класс UserCreate, который наследуется от APIView и определяет методы и атрибуты. Метод "post"обрабатывает POST-запросы для регистрации новых пользователей, создавая запись в базе данных. Затем создается класс MyTokenObtainPairView, наследуемый от TokenObtainPairView, с определением метода "post"для обработки POST-запросов и получения JWT-токенов. Далее создается класс UserDetail для получения информации о конкретном пользователе с использованием метода "get". Затем описывается метод "put"для редактирования данных пользователя. Наконец, описывается класс SearchFilms для поиска и фильтрации фильмов с использованием метода "get". Все представления используют сериализаторы для обработки данных и возвращают результаты в формате JSON.
4. Настройка маршрутов — в данной главе описывается заключительный шаг в создании сервера API на Django. Этот шаг связан с настройкой маршрутов, которые определяют соответствие между URL-адресами и представлениями в приложении. Для настройки маршрутов в Django используется файл urls.py внутри приложения. В этом файле определяются пути (URL-шаблоны) и указывается, какие представления должны быть связаны с этими путями.

Ниже представлены основные моменты по реализации клиентской части, которые будут подробно рассмотрены

1. Роутинг — в данной главе описывается реализация маршрутизации в приложении с использованием компонента Routing. Этот компонент управляет маршрутами и возвращает JSX-элемент, который настраивает маршрутизацию в приложении. Внутри компонента Routing используется компонент Main, который загружается асинхронно с помощью механизма динамической загрузки. Компонент Main экспортирует MainPage, который является основной страницей приложения. Для установки маршрутов применяется компонент Router с базовым путем /movies. Внутри компонента Router определен компонент Routes, который содержит все доступные

маршруты и соответствующие им компоненты.

2. API — В данной главе описывается конфигурация Axios для работы с HTTP-запросами в проекте. Создается экземпляр объекта $api, который использует библиотеку axios для взаимодействия с API. Устанавливаются параметры запросов, такие как передача учетных данных и настройка заголовков. Также настраивается перехватчик ответов, который выполняет обновление токена доступа при получении ошибки авторизации.
3. Менеджер состояний — Глава описывает создание класса Store, который является хранилищем данных для управления состоянием приложения. Класс Store имеет два свойства: isAuth, которое хранит информацию о состоянии авторизации пользователя (true/false), и isLoading, которое хранит информацию о состоянии загрузки данных (true/false). Конструктор класса и метод makeAutoObservable(this) используются для автоматического создания наблюдаемых свойств, которые могут быть отслеживаемыми в MobX. Далее, в классе определены методы: setAuth, setLoading, login, registration. Таким образом, класс Store предоставляет методы для управления авторизацией и регистрацией пользователей, а также хранит информацию о текущем состоянии авторизации и загрузки данных в приложении
4. UI-компоненты, Страница регистрации и авторизации, Главная страница, Страница фильмама, Страница со всеми фильмами, Личный кабинет - в данных главах рассмотрена реализация функциональных компонент на основе TSX в React. В ходе изложения было уделено внимание использованию различных хуков, которые значительно упрощают и улучшают процесс разработки, стилизации модулей с использованием принципов адаптивной и резиновой верстки, работе с данными. Подход к созданию компонентов пользовательского интерфейса с использованием TSX позволил создавать повторно используемые и легко поддерживаемые компоненты. Использование TypeScript способствовал улучшению производительности и надежности приложения. В процессе разработки были использованы различные методы (хуки), предлагаемые React. Например, хук useState использовался для управления состоянием компонента, что позволяет отслеживать и обновлять значения полей формы. Хук useEffect позволял выполнять побочные эффекты, такие как отправка данных фор-

мы на сервер или обновление локального хранилища. Хуки useMemo и useCallback способствовали оптимизации производительности, позволяя избежать ненужных вычислений и повторных вызовов функций.

Заключение

В рамках дипломного проекта был разработан комплексный программный проект, включающий в себя как клиентскую, так и серверную части.

Одной из важных задач было создание пользовательского интерфейса, который бы предоставлял пользователям удобный доступ к функциям приложения. С использованием современных веб-технологий, таких как HTML, CSS и JavaScript, были разработаны страницы, формы, элементы управления и анимации, которые обеспечивают интуитивно понятный и привлекательный интерфейс.Также важной задачей было создание сервера, обеспечивающего обработку запросов от клиентской части и взаимодействие с базой данных. С использованием Django были разработаны модели данных, созданы API-эндпоинты для получения и обновления информации о фильмах, реализована авторизация и аутентификация пользователей, а также механизмы обработки запросов и валидации данных.

Для обеспечения динамического и отзывчивого поведения приложения был использован фреймворк React. С помощью React были созданы компоненты, которые позволяют эффективно организовать структуру приложения, управлять состоянием, обрабатывать события и обновлять интерфейс при изменении данных.

В процессе разработки клиентской части было уделено внимание оптимизации производительности приложения. Были применены техники ленивой загрузки компонентов, кэширования данных и оптимизации рендеринга. Это позволяет ускорить загрузку страниц, снизить нагрузку на сервер и обеспечить плавную работу интерфейса даже при большом объеме данных.