ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

РЕФЕРАТ

по дисциплине: «**Администрирование информационных систем**»

на тему «**Разработка клиент-серверных приложений: взаимодействие React.js фронтенда с Django/FastAPI/Spring бэкендом**»

Выполнил: студент гр. ПРО-436Б
Багауов К.И.
Проверил:
доцент каф. ВМиК
Чернышев Е. С.

Уфа 2025 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном мире веб-разработки создание полнофункциональных клиент-серверных приложений является одной из наиболее востребованных задач. Архитектура, разделяющая приложение на фронтенд и бэкенд компоненты, стала стандартом индустрии, обеспечивая масштабируемость, гибкость и удобство разработки.

React.js, разработанный компанией Facebook, завоевал позицию одного из наиболее популярных фронтенд-фреймворков благодаря своей эффективности, компонентному подходу и обширной экосистеме. Для создания серверной части приложений разработчики имеют широкий выбор технологий, среди которых особое место занимают Django, FastAPI и Spring — каждый из которых обладает уникальными характеристиками и преимуществами.

Актуальность данной темы обусловлена растущей потребностью в создании высокопроизводительных веб-приложений, способных обрабатывать большие объемы данных и обеспечивать отличный пользовательский опыт. Понимание принципов взаимодействия между фронтенд и бэкенд компонентами является критически важным для современного разработчика.

Цель данного реферата — рассмотреть основные принципы разработки клиент-серверных приложений с использованием React.js на стороне клиента и альтернативных бэкенд-фреймворков: Django, FastAPI и Spring. Особое внимание будет уделено механизмам взаимодействия между компонентами, паттернам проектирования и практическим аспектам реализации.

1 Архитектура клиент-серверных приложений

Клиент-серверная архитектура представляет собой распределенную модель вычислений, в которой задачи и рабочая нагрузка распределяются между поставщиками ресурсов или услуг (серверами) и потребителями этих услуг (клиентами). В контексте веб-приложений клиент обычно представлен браузером, выполняющим код на стороне пользователя, в то время как сервер обрабатывает бизнес-логику, управляет данными и предоставляет API для взаимодействия.

Фронтенд является клиентской частью приложения, работающей в браузере пользователя. Его основные задачи включают отображение пользовательского интерфейса и управление его состоянием, обработку пользовательского ввода и взаимодействий, отправку HTTP-запросов к бэкенду и обработку ответов, валидацию данных на стороне клиента, маршрутизацию и навигацию в рамках приложения.

Бэкенд представляет собой серверную часть приложения и выполняет следующие функции: обработка бизнес-логики приложения, управление базой данных и хранением информации, аутентификация и авторизация пользователей, предоставление API для взаимодействия с клиентами, интеграция с внешними сервисами и системами, обеспечение безопасности и защиты данных.

Взаимодействие между фронтендом и бэкендом в веб-приложениях осуществляется преимущественно через протокол HTTP. REST (Representational State Transfer) — это архитектурный стиль для создания веб-сервисов, который использует HTTP-методы для выполнения операций над ресурсами. Данные между клиентом и сервером передаются в формате JSON, который является легковесным и удобным для работы.

2 React.js как фронтенд-фреймворк

2.1 Основные характеристики React.js

React.js — это JavaScript-библиотека с открытым исходным кодом, разработанная Facebook в 2013 году для создания пользовательских интерфейсов. React фокусируется на построении компонентов UI, которые управляют собственным состоянием и эффективно обновляют отображение при изменении данных.

Ключевые особенности React включают компонентный подход, при котором интерфейс разбивается на независимые, переиспользуемые компоненты; Virtual DOM для оптимизации производительности через виртуальное представление DOM; однонаправленный поток данных, упрощающий отслеживание изменений и отладку; JSX синтаксис как расширение JavaScript для описания UI; и Hooks — механизм для использования состояния и других возможностей React в функциональных компонентах.

2.2 Управление состоянием

Управление состоянием является критически важным аспектом разработки React-приложений. Состояние определяет, какие данные отображаются в компонентах и как компоненты реагируют на действия пользователя.

Основные подходы к управлению состоянием включают: локальное состояние (useState) для данных, специфичных для отдельного компонента; Context API для передачи данных через дерево компонентов без пропсов; Redux/Zustand как централизованное хранилище состояния для крупных приложений; React Query/SWR — специализированные решения для управления серверным состоянием и кэшированием.

2.3 Взаимодействие с API

Для взаимодействия с бэкендом React-приложения используют асинхронные HTTP-запросы. Наиболее распространенные способы включают Fetch API — встроенный в браузеры механизм для выполнения HTTP-запросов; Axios — популярная библиотека с расширенными возможностями и удобным API; React Query — библиотека для управления асинхронными данными с автоматическим кэшированием и синхронизацией.

В современных React-приложениях запросы к API обычно выполняются внутри хуков useEffect или с помощью специализированных библиотек, которые обеспечивают автоматическую обработку состояний загрузки, ошибок и кэширования данных.

.

3 Бэкенд-фреймворки

3.1 Django

Django — это высокоуровневый Python веб-фреймворк, который способствует быстрой разработке и чистому, прагматичному дизайну. Создан в 2005 году, Django следует принципу «батарейки включены», предоставляя множество встроенных функций.

Основные характеристики Django включают ORM (Object-Relational Mapping) — мощную систему для работы с базами данных без написания SQL; автоматически генерируемую административную панель для управления данными; Django REST Framework — расширение для создания RESTful API с сериализацией и аутентификацией; встроенную защиту от CSRF, XSS, SQL-инъекций; и масштабируемость, подтвержденную использованием в крупных проектах.

3.2 FastAPI

FastAPI — современный, быстрый веб-фреймворк для создания API на Python 3.7+ с использованием стандартных аннотаций типов. Выпущен в 2018 году, FastAPI быстро завоевал популярность благодаря своей производительности и удобству.

Ключевые особенности FastAPI включают высокую производительность, сопоставимую с NodeJS и Go благодаря асинхронности; автоматическую генерацию документации OpenAPI (Swagger) и ReDoc; валидацию данных через Pydantic для автоматической проверки типов; нативную поддержку async/await для высокой пропускной способности; использование аннотаций типов Python для улучшения читаемости и IDE-поддержки.

3.3 Spring (Spring Boot)

Spring — это комплексная платформа для разработки корпоративных Java-приложений. Spring Boot, созданный в 2014 году, упрощает создание production-ready приложений с минимальной конфигурацией.

Основные характеристики Spring Boot включают инверсию контроля (IoC) — управление зависимостями через Dependency Injection; Spring Data JPA для упрощенной работы с базами данных; Spring Security — мощную систему аутентификации и авторизации; встроенный сервер (Tomcat, Jetty или Undertow) для standalone запуска; возможности для построения микросервисной архитектуры через Spring Cloud.

4 Механизмы взаимодействия

4.1 RESTful API и JSON

REST (Representational State Transfer) представляет собой архитектурный стиль, определяющий набор ограничений для создания веб-сервисов. RESTful API использует HTTP-методы для выполнения CRUD-операций (Create, Read, Update, Delete) над ресурсами.

Принципы RESTful API включают stateless подход, при котором каждый запрос содержит всю необходимую информацию; единообразие интерфейса через стандартизированные методы и форматы; ресурсную ориентацию, где каждый объект представлен как ресурс с уникальным URI; кэшируемость ответов. JSON является стандартным форматом обмена данными, обеспечивая компактность, читаемость и нативную поддержку в JavaScript.

.

4.2 Аутентификация и авторизация

Безопасность является критически важным аспектом взаимодействия клиента и сервера. Основные механизмы включают JWT (JSON Web Tokens) — самодостаточные токены, содержащие информацию о пользователе и не требующие хранения сессий на сервере; OAuth 2.0 как протокол авторизации для делегирования доступа, используемый для входа через сторонние сервисы; session-based аутентификацию, при которой сервер хранит информацию о сессии, а клиент получает session ID в cookie.

4.3 CORS (Cross-Origin Resource Sharing)

CORS — это механизм безопасности браузера, который позволяет веб-странице делать запросы к домену, отличному от того, с которого была загружена страница. В разработке клиент-серверных приложений фронтенд часто работает на отдельном порту или домене от бэкенда.

Настройка CORS в различных фреймворках осуществляется следующим образом: в Django используется django-cors-headers для настройки ALLOWED_ORIGINS; в FastAPI применяется встроенный CORSMiddleware с указанием разрешенных источников; в Spring используется аннотация @CrossOrigin или глобальная конфигурация WebMvcConfigurer.

5 Практические примеры взаимодействия

5.1 React + Django REST Framework

Рассмотрим типичный пример создания API endpoint в Django и его использование в React приложении. В Django создается модель Task,

сериализатор для преобразования данных в JSON, и ViewSet для обработки CRUD-операций. Django REST Framework автоматически создает endpoints для всех операций.

5.2 React + FastAPI

FastAPI предлагает более современный подход с использованием асинхронности и автоматической валидации данных через Pydantic. Определяются Pydantic модели для валидации входящих и исходящих данных, создаются асинхронные функции-обработчики с декораторами. FastAPI автоматически генерирует OpenAPI документацию, что значительно упрощает разработку фронтенда.

5.3 React + Spring Boot

Spring Boot предлагает корпоративный подход к разработке API с упором на типобезопасность и масштабируемость. В Spring создаются Entity классы для представления данных, Repository интерфейсы для работы с базой данных, Service классы для бизнес-логики, и REST Controller для обработки HTTP-запросов. Spring автоматически сериализует Java объекты в JSON.

.

6 Сравнительный анализ

FastAPI демонстрирует наилучшую производительность благодаря асинхронной природе и использованию современных возможностей Python. При правильной настройке FastAPI может обрабатывать десятки тысяч запросов в секунду, что сопоставимо с Node.js и Go.

Spring Boot обеспечивает стабильную производительность за счет оптимизаций JVM и многопоточности. Хотя Spring может уступать FastAPI в синтетических бенчмарках, его производительность в реальных enterprise-сценариях часто превосходит конкурентов благодаря зрелой экосистеме.

Django традиционно считается медленнее FastAPI из-за синхронной природы WSGI. Однако с введением ASGI и Django 3.0+ производительность значительно улучшилась. Для большинства веб-приложений производительность Django более чем достаточна.

Django предлагает максимальную скорость разработки благодаря принципу «батарейки включены». FastAPI обеспечивает быструю разработку API-центричных приложений с автоматической документацией. Spring Boot требует больше начального времени на настройку, но Spring Initializr упрощает начало работы.

Области применения различаются: Django идеален для content-driven сайтов, CMS, e-commerce; FastAPI отлично подходит для микросервисов, API-гейтвеев, real-time приложений; Spring Boot — лучший выбор для enterprise-приложений, финансовых систем, банковских приложений.

ЗАКЛЮЧЕНИЕ

В ходе данного исследования были рассмотрены основные принципы разработки клиент-серверных приложений с использованием React.js на фронтенде и трех популярных бэкенд-фреймворков: Django, FastAPI и Spring Boot. Каждая из рассмотренных технологий обладает уникальными характеристиками и предназначена для решения определенного класса задач.

React.js зарекомендовал себя как мощный и гибкий инструмент для создания современных пользовательских интерфейсов. Его компонентная архитектура, эффективное управление состоянием и обширная экосистема делают его отличным выбором для фронтенд-разработки независимо от выбранного бэкенда.

Django предлагает комплексное решение «все в одном» с акцентом на быструю разработку и практичность. FastAPI представляет собой современный подход к разработке API с фокусом на производительность и удобство разработчика. Spring Boot обеспечивает enterprise-уровень надежности и масштабируемости.

Владение навыками разработки клиент-серверных приложений с использованием современного стека технологий является важной компетенцией для любого веб-разработчика и открывает широкие возможности для карьерного роста в индустрии разработки программного обеспечения.

СПИСОК ЛИТЕРАТУРЫ

1. React Documentation. Official React Documentation. URL: https://react.dev/ (дата обращения: 23.12.2024)
2. Django Documentation. Django Project. URL: https://docs.djangoproject.com/ (дата обращения: 23.12.2024)
3. FastAPI Documentation. Sebastián Ramírez. URL: https://fastapi.tiangolo.com/ (дата обращения: 23.12.2024)
4. Spring Boot Documentation. VMware. URL: https://spring.io/projects/spring-boot (дата обращения: 23.12.2024)
5. Fielding, R.T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000
6. Banks, A., Porcello, E. Learning React: Modern Patterns for Developing React Apps. 2nd Edition. O'Reilly Media, 2020
7. Walls, C. Spring Boot in Action. Manning Publications, 2016
8. Greenfeld, D.R., Greenfeld, A.R. Two Scoops of Django: Best Practices for Django 3.x. 5th Edition. Two Scoops Press, 2020
9. MDN Web Docs. HTTP Overview. Mozilla Foundation. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата обращения: 23.12.2024)
10. Ramírez, S. FastAPI: Modern Python Web Development. Independently published, 2021

Реферат размещен по ссылке:

https://github.com/AIS-436/Bagauov_Karim_Ildarovich_28