МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра дифференциальных уравнений и математической экономики

Модернизация архитектуры веб-форума с внедрением GraphQL, Docker и React

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

Студента(ки) 2 курса 247 группы
направления 09.04.03 - Прикладная информатика
код и наименование направления
механико-математического факультета
наименование факультета, института, колледжа
Черненко Артема Васильевича
фамилия, имя, отчество

Научный руководитель
доцент, к.ф. м.н. _____ Л.П. Кувардина
должность, уч. степень, уч. звание дата, подпись инициалы, фамилия

Заведующий кафедрой
зав.каф.,д.ф-м.н., профессор _____ С.И. Дудов
должность, уч. степень, уч. звание дата, подпись инициалы, фамилия

Саратов 2025

ВВЕДЕНИЕ

С развитием интернет-технологий и ростом числа пользователей, активно использующих онлайн-сервисы, возрастает потребность в создании высокопроизводительных, масштабируемых и удобных в использовании веб-приложений. Особенно это актуально для форумов и платформ коллективного взаимодействия, где критичны скорость отклика, стабильность при нагрузке и адаптивность интерфейса.

По статистике, уже в 2024 году число интернет-пользователей превысило 5,3 миллиарда человек. Это накладывает дополнительные требования к архитектуре веб-приложений, способных обслуживать большое количество одновременных пользователей. Монолитные системы, построенные на классических технологиях вроде Django с серверным рендерингом, уже не способны удовлетворить требованиям современного веба. Они ограничены в гибкости, трудны в сопровождении и масштабировании, не обеспечивают должного уровня интерактивности и интеграции с внешними клиентами, такими как мобильные приложения.

Актуальность выбранной темы обуславливается необходимостью модернизации архитектуры таких систем с применением современных технологий, позволяющих добиться высокой производительности, гибкости и удобства развертывания. В данной работе рассматриваются три ключевые технологии: GraphQL, React и Docker, каждая из которых по-своему решает важные задачи при построении современного веб-приложения.

GraphQL предоставляет эффективный способ взаимодействия клиента с сервером, позволяя клиенту запрашивать ровно те данные, которые ему необходимы, и избегать избыточных запросов, характерных для REST API. React обеспечивает построение удобного и отзывчивого пользовательского интерфейса с компонентным подходом и возможностью адаптивного отображения контента. Docker позволяет изолировать окружение разработки и облегчает развертывание и масштабирование приложения.

Данные технологии активно применяются в крупных международных компаниях. Так, Facebook использует React и является его разработчиком, GitHub реализовал переход на GraphQL API для повышения гибкости работы со своими данными, а Netflix, Uber, Bank of America и Deutsche Bank

применяют Docker для масштабируемого и надёжного развертывания своих сервисов. Применение этих технологий стало стандартом для построения высоконагруженных и устойчивых веб-приложений.

Целью данной работы является изучение технологий, разработка и реализация новой архитектуры веб-форума, использующего связку GraphQL, Docker и React, с целью повышения производительности, гибкости, удобства поддержки и расширяемости.

Для достижения поставленной цели необходимо решить следующие задачи:

- Проанализировать исходное состояние веб-форума;
- Изучить и обосновать выбор современных технологий;
- Реализовать backend на Django с GraphQL API;
- Разработать клиентскую часть на React с использованием Apollo Client;
- Выполнить контейнеризацию приложения с помощью Docker;
- Провести тестирование и подготовить проект к развёртыванию.

Структура. Магистерская работа состоит из введения, четырёх глав, заключения, списка использованных источников и приложений.

В первых трех главах работы описана теоретическая база выбранных технологий — GraphQL, React и Docker. Будет рассмотрено их влияние на производительность и масштабируемость веб-приложений. Также в этой главе будет проведен анализ преимущества и недостатков этих технологий по сравнению с традиционными подходами разработки веб-приложений.

В четвертой главе работы проведен анализ исходного состояния веб-форума, описана реализация всех этапов модернизации форума, включая создание GraphQL API, разработку фронтенда на React, контейнеризацию с Docker и интеграцию всех компонентов.

В заключении подведены итоги работы и представлены перспективы развития проекта.

Наконец, в Приложениях приводится код программы. В конце приводится список литературы из 22 истоников.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

GraphQL — это язык запросов к данным и среда выполнения для выполнения этих запросов, разработанный Facebook. В отличие от REST API, который использует множественные эндпоинты для получения данных, GraphQL предоставляет единую точку входа для всех запросов. Это позволяет клиенту формировать запросы гибко, получая ровно тот объем информации, который необходим, без лишних или дублирующихся данных.

GraphQL идеально подходит для современных веб-приложений с динамическими интерфейсами, в которых пользователь ожидает мгновенного отклика и точности отображаемых данных. Кроме того, он решает такие проблемы REST, как избыточная загрузка данных (overfetching) и недостаточная загрузка (underfetching), которые особенно ярко проявляются в сложных структурах данных.

В рамках данной работы GraphQL используется как основа взаимодействия между фронтендом (React) и сервером (Django). Было принято решение использовать библиотеку Graphene-Django для интеграции GraphQL в серверную часть. Это решение позволило организовать схему API на основе моделей Django, сохранив привычную структуру данных и обеспечив удобный механизм создания запросов, мутаций и резолверов.

Схема API включает следующие типы:

- ForumType — раздел форума;
- ThreadType — тема в разделе;
- PostType — посты в теме;
- CommentType — комментарии к посту;
- UserType — пользователи;
- UserProfileType — профили пользователей.

Были реализованы запросы (query) и мутации (mutation), позволяющие:

- Получать списки форумов, тем, постов и комментариев;
- Создавать, обновлять и удалять посты и комментарии;
- Аутентифицироваться через JWT и получать данные о текущем пользователе.

Для управления доступом к мутациям использован декоратор @login_required, позволяющий выполнять действия только авторизо-

ванным пользователям. Это особенно важно для обеспечения безопасности операций записи.

Пример запроса в GraphQL, реализующего вывод тем определённого форума с вложенными постами:

```
query {
  threads(forumId: 1) {
    title
    posts {
      content
      author {
        username
      }
    }
  }
}
```

Такой подход позволяет избежать многократных обращений к серверу и формирует компактные, логичные ответы. Это особенно эффективно при отображении сложных вложенных структур, таких как дерево комментариев.

GraphQL также обеспечивает удобную интеграцию с инструментами фронтенда, особенно при использовании Apollo Client. Запросы и мутации можно легко описывать прямо в компонентах, а полученные данные — кэшировать и переиспользовать без повторного запроса к серверу.

Внедрение GraphQL в архитектуру веб-форума позволило обеспечить гибкое и предсказуемое взаимодействие между клиентом и сервером, повысить производительность приложения и упростить разработку и поддержку интерфейса.

Docker — это платформа, предназначенная для упаковки, доставки и запуска приложений в изолированных средах, называемых контейнерами. В отличие от традиционных виртуальных машин, Docker-контейнеры являются легковесными и не содержат собственной операционной системы. Это позволяет запускать несколько контейнеров одновременно с минимальными издержками, что особенно важно при построении микросервисной архитектуры.

В данной работе Docker использовался для контейнеризации всех ключевых компонентов веб-приложения: серверной части (Django + GraphQL),

клиентской части (React) и базы данных (SQLite). Использование контейнеров позволило создать воспроизводимую среду для разработки, тестирования и развертывания, минимизировать ошибки, связанные с несовместимостью зависимостей и версий.

Для каждого компонента приложения был создан собственный Dockerfile, описывающий процесс сборки и запуска:

- backend/Dockerfile — содержит инструкции по установке зависимостей Python, настройке Django и запуску сервера;
- frontend/Dockerfile — собирает React-приложение с помощью Next.js;
- docker-compose.yml — объединяет все сервисы в единую систему, автоматически настраивая сетевое взаимодействие и зависимости.

Пример фрагмента docker-compose.yml:

```
version: '3.8'
services:
  backend:
    build: ./backend
    ports:
      - "8000:8000"
    depends_on:
      - db
  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    depends_on:
      - backend
  db:
    image: postgres:15
    environment:
      POSTGRES_DB: forumdb
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
```

Такой подход обеспечивает простоту запуска проекта: достаточно одной команды (docker-compose up), чтобы все сервисы автоматически поднялись и начали взаимодействовать.

Контейнеризация принесла несколько ключевых преимуществ:

- Изоляция: каждый компонент работает в своей среде, не влияя на другие;
- Масштабируемость: можно легко масштабировать отдельные части приложения;

- Упрощение развертывания: одинаковая конфигурация работает как в среде разработчика, так и на сервере;
- Интеграция с CI/CD: Docker позволяет автоматизировать процесс сборки и деплоя через системы непрерывной интеграции, такие как GitLab CI, GitHub Actions, Jenkins.

Также была реализована возможность создания и хранения Docker-образов на удалённом реестре, что упрощает деплой проекта на другие серверы или в облако.

Внедрение Docker значительно повысило гибкость управления инфраструктурой веб-форума и облегчило совместную работу в команде за счёт унифицированной среды.

React — это популярная JavaScript-библиотека для создания пользовательских интерфейсов, разработанная Facebook. Её ключевыми преимуществами являются компонентный подход, высокая производительность благодаря виртуальному DOM и богатая экосистема, поддерживающая разработку масштабируемых и интерактивных веб-приложений.

Для реализации клиентской части веб-форума в данной работе использовался фреймворк Next.js, построенный поверх React. Он предоставляет поддержку серверного рендеринга (SSR), статической генерации страниц, встроенную маршрутизацию и удобную структуру проекта. Это делает Next.js отличным выбором для создания SEO-оптимизированных, быстро загружаемых и расширяемых веб-приложений.

Связь между фронтендом и сервером обеспечивалась с помощью Apollo Client — мощной библиотеки для взаимодействия с GraphQL API. Она позволяет выполнять запросы и мутации, управлять кэшем и состоянием данных, обрабатывать ошибки и упрощать работу с асинхронными операциями.

В клиентской части были реализованы следующие ключевые компоненты:

- Login — форма авторизации, использующая GraphQL-мутацию tokenAuth. Токен авторизации сохраняется в localStorage, обеспечивая доступ к защищённым разделам;
- Forums — компонент для отображения списка форумов. Использует GraphQL-запрос allForums, обрабатывает состояние загрузки и ошибок;
- Threads — компонент, отображающий список тем в выбранном форуме;

- Posts — компонент, отображающий посты в теме, с возможностью добавления новых постов и комментариев;
- CreatePostPage — форма для создания нового поста, с динамической загрузкой форумов и тредов.

Пример GraphQL-запроса, используемого в компоненте:

```
query {
  allForums {
    id
    title
  }
}
```

Для хранения состояния авторизации и управления токеном использовался контекст React (AuthContext). Он предоставляет методы login, logout и доступ к текущему токену, что упрощает управление доступом к страницам.

Также была реализована маршрутизация с помощью встроенного роутера Next.js, что позволило создать отдельные страницы для:

- /login — авторизация;
- /forums — список форумов;
- /threads/[id] — просмотр тем форума;
- /posts/[id] — просмотр и обсуждение конкретного поста.

Для визуального оформления интерфейса использовалась библиотека Material-UI (MUI), которая предоставляет готовые компоненты, соответствующие стандартам Material Design. Это позволило быстро создать современный, адаптивный и удобный интерфейс, поддерживающий как настольные, так и мобильные устройства.

После реализации серверной и клиентской частей, а также настройки контейнеризации, был выполнен этап интеграции всех компонентов веб-форума в единую систему. Благодаря использованию Docker и Docker Compose, удалось автоматизировать запуск всех сервисов: база данных, Django backend с GraphQL API и фронтенд на Next.js.

Архитектура итогового приложения имеет следующую структуру:

- Frontend: React + Next.js, взаимодействует с сервером через Apollo Client и GraphQL;

- Backend: Django с Graphene-Django, реализует API и логику работы форума;
- Database: PostgreSQL, хранит данные пользователей, постов, комментариев и др.;
- Auth: система JWT-аутентификации через библиотеку graphql-jwt;
- Docker: обеспечивает воспроизводимое окружение для всех компонентов.

Интеграция всех частей происходила поэтапно:

1. Проверка взаимодействия между Apollo Client и GraphQL API — успешный обмен запросами и мутациями.
2. Настройка авторизации: получение и хранение токена, автоматическое обновление и защита маршрутов.
3. Тестирование отображения данных с сервера и создание новых объектов (постов, комментариев).
4. Отладка взаимодействия компонентов при работе внутри контейнеров Docker.

Кроме того, был реализован механизм обработки ошибок и валидации данных:

- На стороне клиента — проверка введённых данных перед отправкой;
- На сервере — обработка исключений и сообщений об ошибках в мутациях GraphQL;
- При отсутствии токена или просроченном токене доступ к защищённым мутациям блокируется.

Результатом интеграции стало создание модульной архитектуры, позволяющей при необходимости масштабировать компоненты, легко заменять или обновлять части системы и подключать сторонние клиенты.

Проект готов к развертыванию на облачных платформах (например, Heroku, Render, DigitalOcean).

Таким образом, реализованный веб-форум представляет собой современное, масштабируемое и удобное веб-приложение, построенное по принципам разделения логики, гибкой архитектуры и независимости компонентов.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы была реализована модернизация архитектуры веб-форума с применением современных технологий — GraphQL, Docker и React. Проведённый анализ показал, что традиционная монолитная архитектура на Django ограничивает масштабируемость, гибкость и удобство поддержки веб-приложения. Внедрение новых подходов позволило решить эти проблемы и создать современную, расширяемую и производительную систему.

Основные результаты работы:

- Разработан GraphQL API на основе Django и библиотеки Graphene-Django, реализующий гибкую модель взаимодействия клиента и сервера;
- Реализована клиентская часть на базе React и Next.js, с использованием Apollo Client для обращения к GraphQL;
- Выполнена полная контейнеризация проекта с помощью Docker и Docker Compose, обеспечившая простое и воспроизводимое развертывание;
- Проведено тестирование всех компонентов приложения, обеспечена безопасная аутентификация и контроль доступа;

Теоретическая значимость работы заключается в изучении и сопоставлении технологий современного веб-разработки, таких как GraphQL и Docker, а также в сравнительном анализе архитектурных решений.

Практическая ценность проекта выражается в создании работоспособного, расширяемого веб-форума, который может быть использован как база для учебных, исследовательских или коммерческих целей. Разделение клиентской и серверной частей, гибкий API и унифицированная среда развёртывания делают приложение удобным в поддержке и дальнейшем развитии.

Перспективы развития проекта включают:

- Подключение мобильных клиентов;
- Расширение функционала (уведомления, рейтинг, личные сообщения);
- Интеграцию с другими сервисами через REST или GraphQL;
- Настройку CI/CD и автоматического мониторинга.

Таким образом, поставленные в работе цели достигнуты, задачи выполнены, а полученные результаты демонстрируют преимущества использования современных технологических решений при разработке полноценных веб-приложений.