МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра системного анализа и автоматического управления

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ НА ОСНОВЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 5 курса 551 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Шибаева Александра Дмитриевича

Научный руководитель
доцент, к. ф.-м. н. _____ М. В. Корнилов

Заведующий кафедрой
к. ф.-м. н., доцент _____ И. Е. Тананко

Саратов 2025

ВВЕДЕНИЕ

Актуальность темы. Стремительное развитие цифровых технологий коренным образом меняет подходы к разработке программного обеспечения. Еще полтора десятилетия назад типичные информационные системы представляли собой монолитные приложения, развернутые на нескольких серверах и обрабатывающие гигабайты статичных данных. Пользователи часто сталкивались с сообщениями о временной недоступности сервисов из-за плановых обновлений или технического обслуживания [1].

Современная цифровая среда предъявляет совершенно иные требования. Технологии настолько глубоко интегрировались в повседневную жизнь, что любые перебои в работе сервисов вызывают у пользователей значительный дискомфорт. Представим, например, что банковский сайт, используемый для быстрых денежных переводов, внезапно начинает обрабатывать запросы по полминуты вместо привычных долей секунды. В современном мире подобные задержки недопустимы.

Современный пользователь редко терпит низкую производительность приложений и, скорее, быстрее переключится на какой-нибудь другой сервис, нежели будет терпеть. Именно в ответ на эти вызовы возникла и получила широкое распространение микросервисная архитектура, способная обеспечить неплохой уровень надежности, отказоустойчивости и масштабируемости приложений [1,2].

Актуальность данного исследования определяется необходимостью переосмысления традиционных архитектурных подходов в пользу инновационных решений. Новые методы позволяют создавать динамичные, адаптивные и высокопроизводительные информационные системы, способные эффективно функционировать под значительными нагрузками.

Цель бакалаврской работы — спроектировать и реализовать веб-приложение для управления задачами на основе микросервисной архитектуры.

Поставленная цель определила **следующие задачи**:

- изучить основы микросервисной архитектуры;
- разработать архитектуру системы управления задачами, предусматривающую декомпозицию на функциональные микросервисы;
- реализовать спроектированную систему с использованием актуального технологического стека;

— внедрить методологию DevOps и автоматизировать процессы разработки и развертывания;
— развернуть систему в среде оркестрации контейнеров.

Методологические основы микросервисной архитектуры представлены в работах таких авторов, как С. Ньюман [1,2], Д. Ким [3].

Практическая значимость бакалаврской работы заключается в формировании комплексного подхода к проектированию и реализации высоконагруженных систем на основе микросервисной архитектуры. Результаты исследования и предложенные решения могут быть использованы разработчиками и архитекторами программного обеспечения для повышения производительности, отказоустойчивости и масштабируемости создаваемых ими информационных систем.

Структура и объем работы. Бакалаврская работа состоит из введения, 3 разделов, заключения, списка использованных источников, шести приложений и цифрового носителя в качестве приложения. Общий объем работы — 76 страниц, из них 56 страниц — основное содержание, включая 24 рисунка, список использованных источников информации — 20 наименований.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Эволюция архитектур веб-приложений и микросервисная архитектура» посвящен рассмотрению исторического развития архитектур веб-приложений, а также освещаются основные принципы и концепции микросервисной архитектуры.

В подразделе 1.1 рассматриваются особенности монолитной архитектуры, её преимущества и ограничения.

Подраздел 1.2 посвящен сервис-ориентированной архитектуре (SOA) и её отличиям от монолита и микросервисов.

В подразделе 1.3 описываются основные принципы микросервисной архитектуры.

Подраздел 1.4 содержит определение микросервиса и его ключевые характеристики.

В подразделе 1.5 перечислены преимущества и недостатки микросервисной архитектуры.

В подразделе 1.6 описываются паттерны проектирования микросервисов.

В подразделе 1.7 рассматриваются современные технологии и подходы к разработке микросервисных приложений. В частности были рассмотрены технологии контейнеризации, оркестрации контейнеров, принципы GitOps и непрерывной доставки, API Gateway и сервисные сети, Single-Page Applications (SPA), REST API и альтернативные протоколы, JWT, Шаблоны Frontend-Backend взаимодействия.

Подраздел 1.8 содержит анализ предметной области и требований к функциональности системы.

В подразделе 1.9 рассмотрено проектирование компонентов системы и описание их взаимодействия.

Второй раздел «Технологический стек и реализация микросервисов» посвящен реализации компонентов системы на основе микросервисной архитектуры.

Выбор технологического стека основывался на требованиях к системе, популярности инструментов в индустрии и их совместимости с микросервисной архитектурой. Для пользовательского интерфейса была выбрана библиотека React. Бэкенд-сервисы реализованы на языке Go с использованием фреймворка Gorilla Mux. В качестве СУБД была выбрана документо-ориентированная база данных MongoDB, обеспечивающая простоту интеграции с Go через официальный MongoDB Go Driver.

Логику реализованного приложения можно рассмотреть на рисунке 1.

Внешний вид реализованного приложения после регистрации и входа можно увидеть на рисунке 2.

Инфраструктурная часть разработанной системы опирается на современные технологии, обеспечивающие автоматизацию, гибкость и устойчивость к сбоям. Для контейнеризации микросервисов используется инструмент Docker, который позволяет упаковать каждый сервис в изолированную среду со всеми его зависимостями, обеспечивая единообразие поведения при запуске на разных платформах.

Для оркестрации контейнеров выбран Kubernetes — де-факто стандарт в управлении контейнеризированными приложениями. Он предоставляет средства автоматического масштабирования, самовосстановления при сбоях, балансировки нагрузки и управления конфигурациями. В системе применяется также Ingress-контроллер Traefik, выполняющий функции маршрутизации внешнего

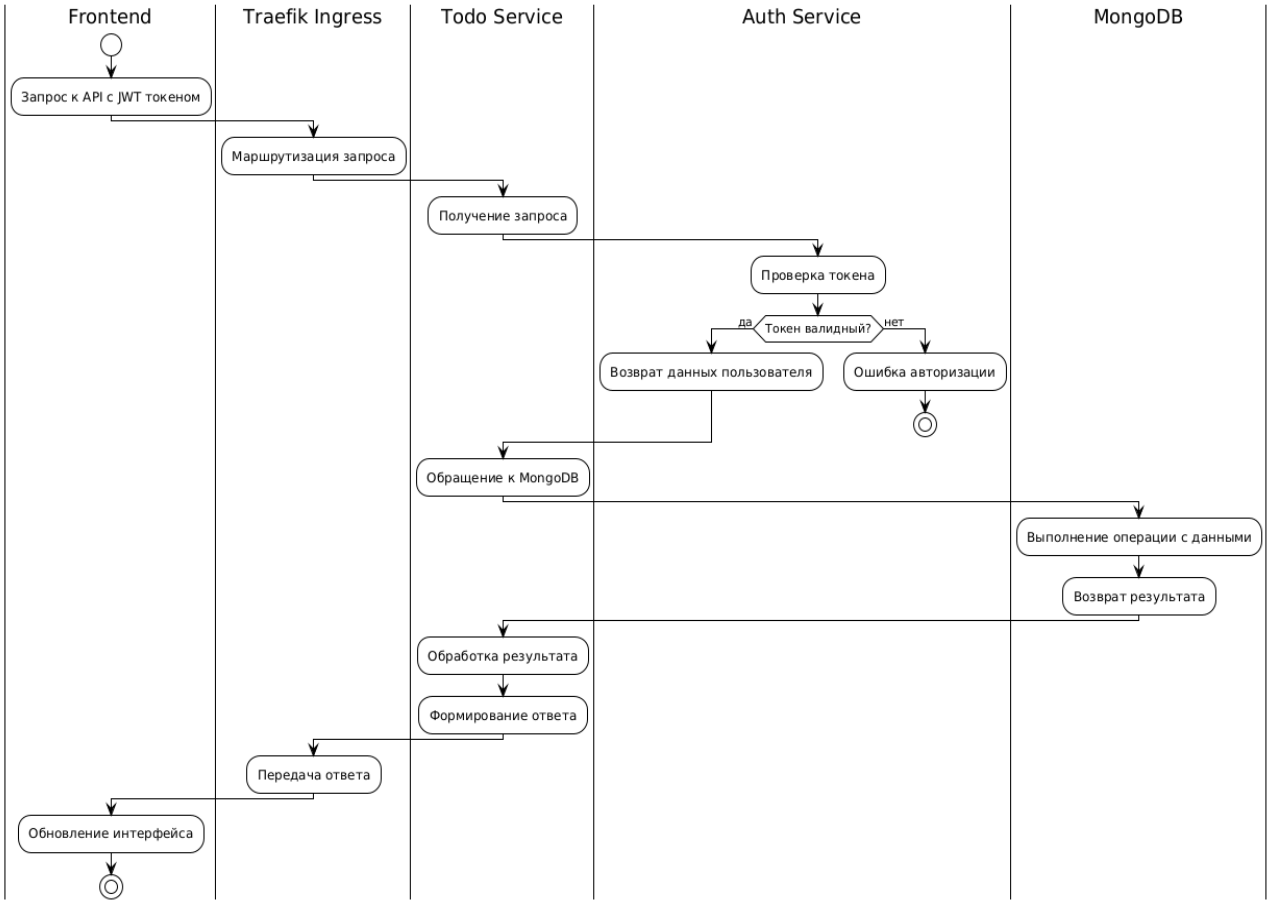

Рисунок 1 – Логика реализованного приложения

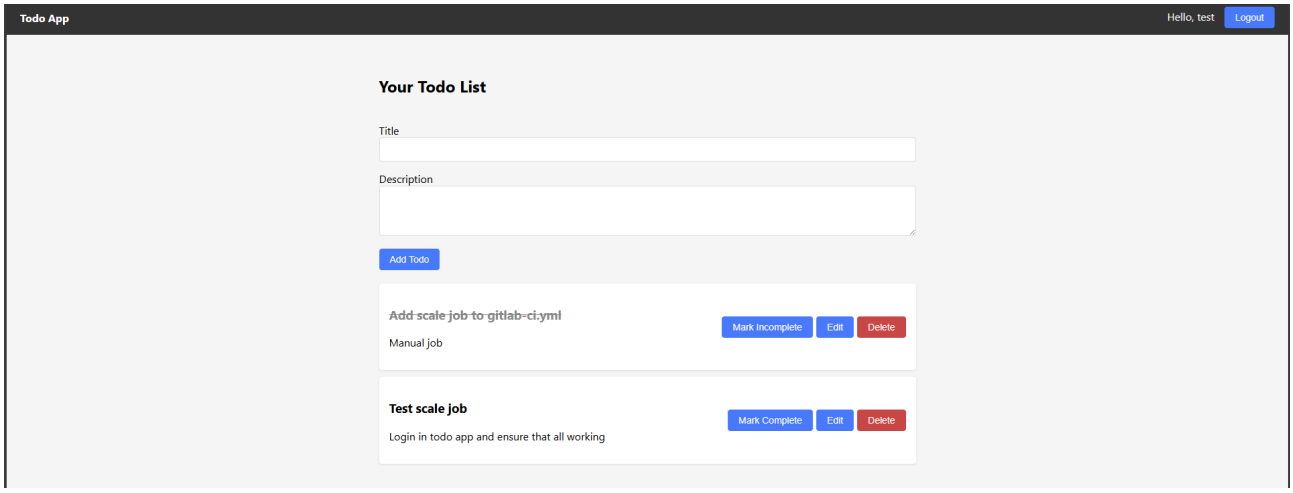

Рисунок 2 – Внешний вид приложения

трафика к соответствующим сервисам, управления TLS-сертификатами и служащий API-шлюзом.

Процессы непрерывной интеграции (CI) и непрерывной доставки (CD) реализованы с использованием GitLab CI/CD. При каждом изменении кода запускается пайплайн, включающий сборку образов Docker, выполнение тестов и публикацию артефактов. Собранные образы помещаются в реестр контейнеров,

откуда впоследствии развёртываются в кластер Kubernetes.

Управление инфраструктурой осуществляется по принципам GitOps, при которых текущее состояние системы описывается декларативно в Git-репозитории, а изменения в коде автоматически синхронизируются с рабочей средой. Для реализации GitOps-подхода применяется инструмент ArgoCD, осуществляющий мониторинг репозитория с конфигурациями и обеспечивающий автоматическое или полуавтоматическое приведение текущего состояния к заданному.

Автоматический процесс внесения изменений в код представлен на рисунке 3.

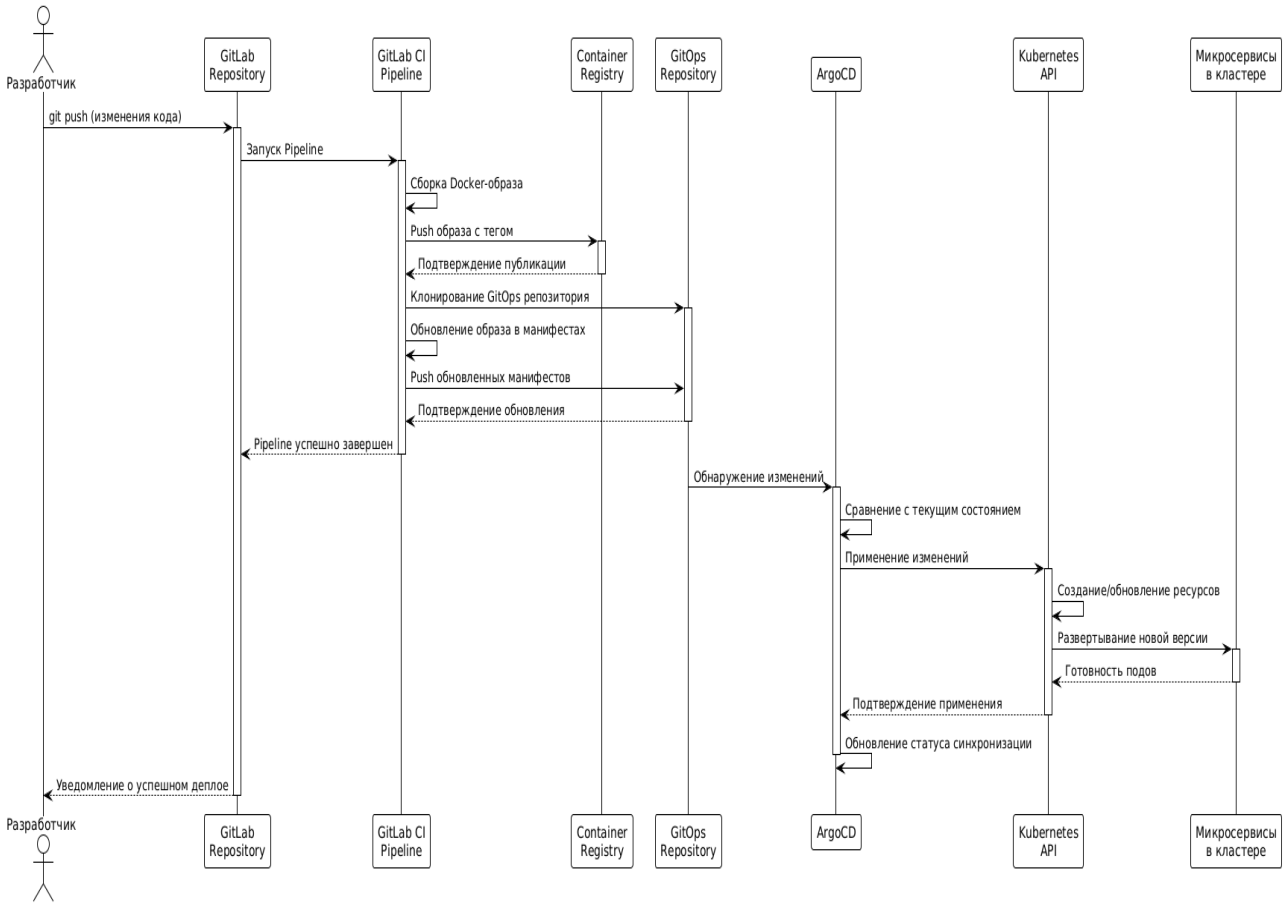

Рисунок 3 – Процесс внесения изменений в код

В подразделе 2.1 приведено описание подхода к тестированию системы. Было проведено ручное тестирование на проверку соответствия функциональным и нефункциональным требованиям.

Реализованный функционал соответствует функциональным требованиям:

1. Регистрация нового пользователя, последующий вход и выход из системы.
2. Создание новой задачи аутентифицированным пользователем.

3. Отображение списка задач, принадлежащих только текущему пользователю.
4. Редактирование существующей задачи (изменение текста, отметка о выполнении).
5. Удаление задачи.
6. Проверка авторизации: попытка доступа к API задач без валидного JWT или с JWT другого пользователя должна приводить к ошибке.

Также достигнуты нефункциональные требования:

— Масштабируемость: архитектура позволяет горизонтально масштабировать сервисы через изменение количества реплик в Kubernetes Deployment.
— Отказоустойчивость: Kubernetes обеспечивает базовую отказоустойчивость за счет перезапуска отказавших контейнеров. Изоляция сервисов предотвращает полный отказ системы при сбое одного компонента.
— Развертывание и эксплуатация: процесс развертывания полностью автоматизирован с помощью GitLab CI/CD и ArgoCD, реализуя принципы GitOps.
— Безопасность: реализована аутентификация на основе JWT и базовая авторизация. Пароли хэшируются.

Подраздел 2.1.1 описывает процесс масштабирования за счёт автоматического изменения количества реплик прямо из CI/CD пайплайна в Kubernetes.

На рисунке 4 демонстрация изменения реплик - вместо трех подов теперь пять.

Команда kubectl get pods −n dev −o wide выводит все поды в пространстве имен dev с дополнительной (wide) информацией.

```
vagrant@master-node-1:~$ kubectl get pods -n dev -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP              NODE
 GATES
auth-service-f7558bbc6-s7qqf    1/1     Running   0          13h     10.233.67.196   node-1
frontend-646568d788-fsg7m       1/1     Running   0          13h     10.233.67.194   node-1
mongodb-645dc47794-d8s9d        1/1     Running   0          17h     10.233.67.163   node-1
todo-service-58889b584c-467qh   1/1     Running   0          2m56s   10.233.68.63    node-2
todo-service-58889b584c-6cgk7   1/1     Running   0          13h     10.233.67.197   node-1
todo-service-58889b584c-jvdj7   1/1     Running   0          14m     10.233.67.198   node-1
todo-service-58889b584c-sfk5v   1/1     Running   0          14m     10.233.68.62    node-2
todo-service-58889b584c-tlqll   1/1     Running   0          2m56s   10.233.67.199   node-1
traefik-68946b4dd7-whd4k        1/1     Running   0          14h     10.233.67.186   node-1
```

Рисунок 4 – Поды todo−service, выведенные через kubectl

В подразделе 2.1.2 рассматривается автоматизированное внесение изменений в код и последующее обновление системы с помощью CI/CD и GitOps.

Третий раздел «Стратегия развертывания и эксплуатации системы» посвящен процессу развертывания системы, конфигурации инфраструктуры и отказоустойчивости программной системы.

В подразделе 3.1 описаны принципы масштабирования, настройка отказоустойчивости и балансировки нагрузки.

Подраздел 3.2 содержит анализ полученных результатов и обсуждение перспектив дальнейшего развития системы.

Ключевыми техническими достижениями проекта являются:

- Успешная декомпозиция приложения на независимые микросервисы (frontend, auth-service, todo-service).
- Реализация взаимодействия между сервисами через REST API с использованием JWT для аутентификации.
- Построение полностью автоматизированного CI/CD пайплайна для сборки и развертывания сервисов.
- Внедрение GitOps-подхода с использованием ArgoCD для декларативного управления конфигурацией Kubernetes.
- Развертывание всей системы в среде Kubernetes с использованием Docker-контейнеров.

Опыт разработки данного проекта подтвердил как преимущества, так и сложности микросервисного подхода. К преимуществам можно отнести технологическую гибкость (использование React и Go, с возможностью переписать какой-нибудь сервис на, например, Rust), независимость разработки и развертывания отдельных сервисов, а также потенциал для гранулярного масштабирования и повышения отказоустойчивости. Однако проявились и недостатки, в первую очередь — повышенная операционная сложность, связанная с необходимостью управления множеством сервисов, контейнеров, конфигураций и настройкой взаимодействия между ними.

В подразделе 3.2.1 сформулированы ключевые выводы и рекомендации.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была спроектирована и реализована микросервисная архитектура для веб-приложения управления задачами, которая демонстрирует ключевые преимущества данного подхода перед традиционными монолитными решениями.

В процессе работы были исследованы теоретические основы микросервисной архитектуры, изучены современные подходы к проектированию и разработке распределенных систем. Была разработана архитектурная концепция системы с декомпозицией на функциональные микросервисы.

Практическая реализация системы включала в себя разработку сервисов с использованием современного технологического стека: React для фронтенд-части, Go и Gorilla Mux для бэкенд-сервисов, MongoDB в качестве СУБД. Контейнеризация приложения с помощью Docker и оркестрация с использованием Kubernetes обеспечили гибкую и масштабируемую инфраструктуру. Внедрение процессов CI/CD с использованием GitLab и GitOps-подхода с ArgoCD позволило автоматизировать процессы разработки и развертывания.

Тестирование системы подтвердило соответствие разработанного решения поставленным требованиям. Микросервисная архитектура продемонстрировала преимущества в части изоляции сбоев, независимого масштабирования компонентов и гибкости при внесении изменений в отдельные сервисы.

Полученные результаты имеют практическую значимость и могут быть использованы при проектировании и разработке высоконагруженных и отказоустойчивых веб-приложений. Опыт, накопленный в процессе работы, формирует ценные рекомендации для архитекторов и разработчиков, стремящихся внедрить микросервисный подход в своих проектах.

Основные источники информации:

1. Ньюман, С. Создание микросервисов / С. Ньюман — СПб. : Питер, 2025. - 624 с.
2. Ньюман, С. От монолита к микросервисам / С. Ньюман — СПб. : БХВ, 2021. - 272 с.
3. Ким, Д. Руководство по DevOps / Д. Ким, Д.Хамбл, П. Дебуа, Д. Уиллис — М. : МИФ, 2018. - 512 с.
4. Клеппман, М. Высоконагруженные приложения. Программирование, масштабирование, поддержка / М. Клеппман — Алматы : Sprint Book, 2024. - 640 с.