# СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

Глава 1. ОБЩАЯ ХАРАКТЕРИСТИКА ОРГАНИЗАЦИЙ 4

1.1. Краткая история, основные направления деятельности и структура организации 4

1.2. Экологическая обстановка в организации, охрана окружающей среды, организация энергосбережения 4

Глава 2. Изучение технологических процессов (услуга, деятельность, разработка и т.д.) 6

Глава 3. Организационная и функциональная структура организации 9

Глава 4. Выявление задач, подлежащих автоматизации 11

Глава 5. ПЛАН ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 12

5.1 План индивидуального задания 12

5.2 Постановка задачи 13

5.3 Анализ необходимости автоматизации и существующих решений 13

5.4 Проектирование 15

5.5 Реализация 17

5.6 Тестирование 18

ЗАКЛЮЧЕНИЕ 20

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21

# ВВЕДЕНИЕ

В рамках прохождения технологической практики по программированию, я имел возможность погрузиться в процесс разработки интернет-магазина. Практика проходила в компании WebDad, где я был частью команды разработчиков. Целью практики было получить практические навыки работы с современными технологиями и инструментами, используемыми в разработке веб-приложений.

В частности, мне предстояло освоить язык программирования Go и его фреймворки для разработки серверной части приложения, а также научиться работать с фронтенд-фреймворком Vue.js.

В течение месяца я выполнял задачи, связанные с разработкой различных компонентов интернет-магазина. В процессе работы я использовал такие технологии, как Git для контроля версий и Docker для контейнеризации приложений. Это позволило мне глубже понять, как организовывается работа в команде, какие подходы применяются для упрощения разработки и поддержки приложений, а также как обеспечивается стабильность и масштабируемость решений в продакшене.

Основной упор в моей практике был сделан на развитие практических навыков программирования, а также на понимание и применение лучших практик разработки программного обеспечения. Данный отчет содержит описание основных этапов работы, используемых технологий и полученных результатов.

# Глава 1. ОБЩАЯ ХАРАКТЕРИСТИКА ОРГАНИЗАЦИЙ

# 1.1. Краткая история, основные направления деятельности и структура организации

Компания зарекомендовала себя как надежный партнер в сфере разработки веб-систем. Примерами могут служить успешно выпущенные системы, которые включают в себя сайты, телеграм-боты, Telegram mini apps.

На сегодняшний день основными направлениями деятельности компании являются:

1. Дизайн продукта
2. Разработка веб-приложений.
3. Консалтинг в области IT
4. Поддержка и сопровождение ПО
5. Продвижение продукта

Структура компании организована таким образом, чтобы обеспечить гибкость и эффективность при выполнении различных проектов. Команда компании состоит из:

- разработчиков: каждый разработчик достаточно компетентен для разработки бекенда, фронтенда, а также деплоя приложения.

- тестировщика: занимается тестированием разрабатываемых приложений, а также контролем качества на всех этапах разработки.

- проектного менеджера: ответственен за планирование, управление проектами и взаимодействие с клиентами.

- scrum мастера: обеспечивает контроль выполнения работы команды, помогает освоиться новым работникам.

Такой организационный подход позволяет компании оперативно реагировать на запросы клиентов и обеспечивать высокое качество предоставляемых услуг.

# 1.2. Экологическая обстановка в организации, охрана окружающей среды, организация энергосбережения

Компания WebDad уделяет значительное внимание вопросам охраны окружающей среды и устойчивого развития. В рамках своей деятельности организация активно внедряет меры, направленные на минимизацию негативного воздействия на окружающую среду и эффективное использование ресурсов.

Экологическая обстановка в организации определяется стремлением к снижению углеродного следа и экологически безопасной работе. Для этого компания придерживается следующих принципов:

1. Цифровизация и минимизация использования бумаги: Все процессы внутри компании максимально переведены в цифровой формат. Ведение документации, управление проектами и коммуникации осуществляются с использованием современных IT-инструментов, что позволяет значительно сократить расход бумаги и других канцелярских товаров.

2. Использование экологически чистых материалов: При закупке товаров и оборудования компания отдает предпочтение экологически безопасным материалам и продуктам с минимальным воздействием на окружающую среду.

Охрана окружающей среды осуществляется через внедрение различных инициатив и программ. В частности:

1. Энергоэффективность.

Компания использует энергоэффективные осветительные приборы и бытовую технику с высоким классом энергопотребления. Это позволяет существенно сократить потребление электроэнергии и уменьшить нагрузку на окружающую среду.

2. Зеленые офисы.

В офисных помещениях активно внедряются элементы озеленения, что способствует улучшению качества воздуха и созданию более комфортных условий труда.

3. Образовательные программы.

Компания проводит для сотрудников обучающие программы, направленные на повышение экологической грамотности. Это включает семинары и тренинги по теме рационального использования ресурсов и экологически ответственного поведения.

Организация энергосбережения является одним из приоритетных направлений в экологической политике компании. В этой области реализованы следующие меры:

1. Оптимизация энергопотребления.

В офисах установлены автоматические системы управления освещением, которые регулируют интенсивность света в зависимости от времени суток и присутствия сотрудников в помещениях.

# Глава 2. Изучение технологических процессов (услуга, деятельность, разработка и т.д.)

В ходе прохождения технологической практики основной акцент был сделан на изучении процессов, связанных с разработкой программного обеспечения, в частности, интернет-магазина. В компании разработка программного продукта осуществляется по четко структурированному и выверенному процессу, который включает несколько ключевых этапов.

1. Анализ требований

Первым этапом любого проекта является сбор и анализ требований. В рамках данного этапа команда разработки взаимодействует с клиентом для определения функционала будущего приложения, выявления потребностей и пожеланий, а также анализа рынка и конкурентов. Этот процесс включает следующие шаги:

- Сбор требований: Команда проектировщиков и аналитиков совместно с клиентом определяют основные функции, которые должны быть реализованы в интернет-магазине. Проводятся встречи и обсуждения, на которых детализируются бизнес-цели и задачи проекта.

- Документирование требований: На основе собранных данных составляется техническое задание (ТЗ), которое описывает функциональные и нефункциональные требования к проекту. Этот документ служит основой для всех последующих этапов разработки.

2. Рисование дизайна

После утверждения ТЗ, дизайнер компании приступает к разработке дизайна сайта. Чаще всего используется сервис Figma.

3. Проектирование архитектуры

После утверждения требований начинается этап проектирования архитектуры системы. В данном случае архитектура включает как бекенд, реализованный на языке Go, так и фронтенд, построенный с использованием Vue.js. Проектирование проводится с учетом масштабируемости, надежности и производительности приложения.

- Бекенд: Архитектура серверной части приложения разрабатывается с использованием микросервисного подхода, что позволяет легко масштабировать систему и упрощает поддержку. Бекенд на Go обеспечивает высокую производительность и минимальные задержки в обработке запросов.

- Фронтенд: Визуальная часть приложения проектируется с использованием Vue.js, что обеспечивает гибкость интерфейса и возможность создания динамичных пользовательских интерфейсов. Применение компонентного подхода позволяет легко обновлять и расширять функционал.

- Базы данных: Для хранения данных выбираются подходящие СУБД, учитывающие специфику проекта и объемы данных. Проектируется структура баз данных, определяются схемы и связи между данными.

4. Разработка

Этап реализации включает в себя непосредственную разработку функционала интернет-магазина. Работа разбивается на спринты, в течение которых команда постепенно внедряет и тестирует новые функции.

- Разработка бекенда: Реализация серверной части приложения начинается с создания основных микросервисов, отвечающих за работу с базой данных, обработку пользовательских запросов и интеграцию с внешними системами. Язык Go и его фреймворки (например, Gin) обеспечивают быстрый и эффективный процесс разработки.

- Разработка фронтенда: Создание интерфейса интернет-магазина начинается с реализации базовой структуры страниц и основных компонентов. Vue.js позволяет быстро собирать интерфейсы и легко интегрировать их с бекендом через API.

- Интеграция и контейнеризация: Для обеспечения совместимости и удобства развертывания проекта используются Docker-контейнеры. Это позволяет легко разворачивать и тестировать приложение в различных средах, а также ускоряет процесс интеграции.

5. Тестирование и отладка

После завершения реализации основных модулей проекта проводится тестирование. Этот этап включает:

- Юнит-тестирование: Проверка работы отдельных компонентов системы, чтобы убедиться в их корректной работе.

- Интеграционное тестирование: Тестирование взаимодействия между различными модулями приложения для выявления и устранения возможных проблем на стыках.

- Регрессионное тестирование: Проверка всей системы на наличие ошибок, которые могли возникнуть после внесения изменений или добавления новых функций.

- Ручное тестирование: Проверка работы системы с точки зрения конечного пользователя, чтобы убедиться, что функционал соответствует ожиданиям и требованиям.

6. Развертывание и поддержка

Завершающий этап разработки включает развертывание интернет-магазина на продакшн-среде и настройку его работы. Для этого используется инфраструктура, построенная на Docker и системах автоматизации развертывания.

- Развертывание: Приложение развертывается на серверах или в облачных средах с использованием контейнеров. Это обеспечивает гибкость и возможность быстрого масштабирования в случае необходимости.

- Поддержка и мониторинг: После развертывания приложения команда переходит к этапу поддержки, который включает мониторинг работы системы, устранение возможных ошибок и обновление функционала по мере необходимости.

Таким образом, изучение технологических процессов в компании WebDad позволило мне получить глубокое понимание всех этапов разработки программного обеспечения, от анализа требований до развертывания и поддержки готового продукта.

# Глава 3. Организационная и функциональная структура организации

Организационная и функциональная структура компании является ключевым элементом, обеспечивающим успешную реализацию проектов и эффективное управление ресурсами. В данной главе рассматривается, как именно устроена компания, какие отделы и функции существуют, и как они взаимодействуют между собой.

Функциональная структура компании WebDad организована таким образом, чтобы обеспечить четкое распределение обязанностей и ответственность на каждом уровне. Взаимодействие между отделами построено на принципах сотрудничества и прозрачности, что позволяет эффективно достигать поставленных целей.

- Разработка: Основной фокус в компании делается на командной работе. Каждая команда разработки состоит из специалистов, обладающих различными навыками, которые совместно работают над проектом. В рамках команды есть четкое разделение на фронтенд- и бекенд-разработчиков, что позволяет глубже сосредоточиться на специфических задачах.

- Проектный менеджмент: Проектные менеджеры выполняют роль связующего звена между клиентом и командой разработки. Они занимаются планированием, управляют рисками и обеспечивают своевременное выполнение всех задач. ПМ также координирует взаимодействие между различными отделами, обеспечивая согласованность действий.

- Тестирование и качество: QA-инженер работает параллельно с разработчиками, чтобы своевременно обнаруживать и исправлять ошибки. Их задача — не только тестировать готовые продукты, но и участвовать в планировании и проектировании для предотвращения потенциальных проблем.

- Поддержка и сопровождение: Отдел поддержки тесно взаимодействует с отделом разработки, особенно на этапах развертывания и эксплуатации продукта. Это позволяет быстро реагировать на возникающие у пользователей проблемы и эффективно их решать.

- Маркетинг и продажи: Этот отдел работает в тесной связке с руководством компании для разработки и реализации стратегий продвижения. Они активно взаимодействуют с потенциальными и текущими клиентами, собирают обратную связь и помогают адаптировать продукты под нужды рынка.

Такое распределение обязанностей и функциональная структура позволяют компании WebDad эффективно управлять проектами различного масштаба и сложности, поддерживать высокие стандарты качества и удовлетворенность клиентов, а также обеспечивать устойчивое развитие на рынке.

# Глава 4. Выявление задач, подлежащих автоматизации

Одним из ключевых аспектов успешной разработки программного обеспечения является идентификация процессов и задач, которые могут и должны быть автоматизированы.

Одной из необходимых задач, подлежащих автоматизации, стала настройка процесса непрерывной интеграции и доставки (CI/CD). Автоматизация этого процесса обеспечивает быструю сборку, тестирование и развертывание приложения, что критически важно для интернет-магазина, который должен быть всегда доступен и оперативно обновляться.

Сборка и тестирование: Система CI/CD автоматически запускает процессы сборки кода и выполнения тестов при каждом изменении в репозитории. Это позволяет мгновенно обнаруживать ошибки и устранять их еще на этапе разработки.

Автоматизированное развертывание: Использование Docker и системы оркестрации контейнеров (например, Kubernetes) позволяет автоматически разворачивать обновления на сервере без простоя. Это особенно важно для крупных проектов, где время простоя может приводить к значительным убыткам.

# Глава 5. ПЛАН ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

# 5.1 План индивидуального задания

В рамках моей практики в компании WebDad одной из ключевых задач, которую мне было поручено реализовать, стала настройка и внедрение процесса непрерывной интеграции и доставки (CI/CD) для проекта интернет-магазина. Эта задача включала несколько этапов, которые требовали тщательной подготовки и планирования. Основная цель заключалась в автоматизации процесса сборки, тестирования и развертывания приложения, что позволило бы ускорить разработку и обеспечить высокое качество продукта.

План выполнения индивидуального задания включал следующие этапы:

1. Анализ требований и определение целей:

- Ознакомление с текущей инфраструктурой проекта, используемыми технологиями и требованиями к процессу CI/CD.

- Определение ключевых целей, таких как сокращение времени на развертывание, автоматизация тестирования, и обеспечение стабильности работы приложения.

2. Выбор инструментов и технологий:

- Исследование и выбор подходящих инструментов для реализации CI/CD, таких как Jenkins, GitLab CI, CircleCI или GitHub Actions для автоматизации процесса.

- Определение подходящего инструментария для контейнеризации и оркестрации, включая Docker и Kubernetes.

3. Настройка системы контроля версий:

- Организация репозитория с кодом проекта и настройка системы контроля версий (Git).

- Определение стратегий ветвления и слияния (branching strategies), таких как Git Flow или Trunk-Based Development, для эффективного управления разработкой.

4. Создание скриптов для автоматической сборки и тестирования:

- Написание скриптов, которые автоматически собирают и тестируют проект при каждом коммите в репозиторий.

- Настройка автоматизированного юнит-тестирования и интеграционного тестирования.

5. Настройка автоматического развертывания (deployment):

- Конфигурирование автоматического развертывания приложения на сервер или в облачное окружение после успешного прохождения всех тестов.

- Настройка разных сред (staging, production) для безопасного развертывания и тестирования новых версий.

8. Тестирование и оптимизация процесса:

- Проведение тестирования настроенного процесса CI/CD на реальных сценариях работы проекта.

- Оптимизация процесса на основе результатов тестирования, выявление и устранение узких мест.

# 5.2 Постановка задачи

Постановка задачи включала следующие ключевые компоненты:

1. Автоматизация сборки проекта:

- Требовалось создать процесс, при котором каждый коммит в репозиторий автоматически запускает сборку проекта. Это должно гарантировать, что код всегда находится в рабочем состоянии и может быть собран без ошибок.

2. Автоматизация тестирования:

- Задачей было настроить автоматический запуск юнит-тестов и интеграционных тестов после каждой сборки. Это необходимо для того, чтобы любые изменения в коде проходили тщательную проверку перед тем, как попасть в основную ветку или на продакшн.

3. Настройка системы управления версиями и ветвления:

- Требовалось разработать и внедрить стратегию ветвления, которая бы обеспечивала плавное и безопасное управление версиями проекта, позволяя параллельно разрабатывать новые функции, исправлять ошибки и выпускать обновления.

4. Автоматизация развертывания:

- Задачей было настроить автоматическое развертывание приложения на сервер или в облачное окружение после успешного прохождения всех тестов. Это должно обеспечивать быструю доставку обновлений на тестовые и продакшн-среды.

# 5.3 Анализ необходимости автоматизации и существующих решений

Необходимость автоматизации

В условиях современного рынка, где скорость и качество разработки играют ключевую роль, внедрение процесса непрерывной интеграции и доставки (CI/CD) стало необходимым шагом для проекта интернет-магазина в компании WebDad .

Анализ текущих рабочих процессов показал ряд проблем и узких мест, которые можно было бы решить с помощью автоматизации:

1. Длительный цикл развертывания и обновления:

- Ручные процессы развертывания и тестирования занимали значительное время, что приводило к задержкам в выпуске обновлений и новым функциям. Это негативно влияло на конкурентоспособность и пользовательский опыт.

2. Риск человеческих ошибок:

- Ручное выполнение повторяющихся задач, таких как сборка и развертывание, увеличивало вероятность ошибок, которые могли приводить к багам и сбоям в работе системы, особенно при работе с большим количеством кода.

3. Ограниченная видимость и контроль качества:

- Отсутствие автоматизированного тестирования и мониторинга затрудняло выявление проблем на ранних этапах разработки. Это приводило к тому, что ошибки обнаруживались слишком поздно, когда их исправление требовало значительных затрат времени и ресурсов.

Эти факторы указали на необходимость автоматизации ключевых процессов, что позволило бы значительно сократить время на выполнение рутинных задач, снизить количество ошибок и улучшить качество программного обеспечения.

Существующие решения

Перед началом реализации задачи по внедрению CI/CD был проведен анализ существующих решений, которые могли бы удовлетворить потребности проекта:

1. Jenkins:

- Jenkins является одним из самых популярных инструментов для реализации CI/CD. Он предлагает широкий выбор плагинов для интеграции с различными системами контроля версий, сборки, тестирования и развертывания. Основные преимущества Jenkins — это его гибкость, активное сообщество и возможность кастомизации под специфические требования проекта. Однако, Jenkins может требовать значительных усилий по настройке и поддержке.

2. GitLab CI/CD:

- GitLab CI/CD предоставляет встроенные средства для автоматизации всех этапов разработки, начиная от кодирования и заканчивая развертыванием. Его преимущество в том, что он полностью интегрирован с GitLab, что упрощает управление проектами и позволяет легко настраивать пайплайны для CI/CD. Это решение привлекательно благодаря своей простоте и удобству использования, особенно в проектах, где уже используется GitLab.

3. CircleCI:

- CircleCI — это облачный сервис CI/CD, который предлагает мощные инструменты для автоматизации процессов сборки, тестирования и развертывания. Он известен своей высокой скоростью выполнения пайплайнов и поддержкой контейнеризированных сред. CircleCI особенно полезен для проектов, требующих быстрого развертывания и интеграции с Docker и Kubernetes.

4. GitHub Actions:

- GitHub Actions — это относительно новое решение, интегрированное в платформу GitHub. Оно позволяет настраивать автоматизацию рабочих процессов непосредственно в репозитории. GitHub Actions хорошо подходит для проектов, размещенных на GitHub, и предлагает возможность легко интегрировать CI/CD в существующие процессы разработки. Преимущества включают простоту настройки и использования, а также широкую поддержку различных языков программирования и инструментов.

5. Docker и Kubernetes:

- Docker и Kubernetes стали стандартом для контейнеризации и оркестрации приложений. Docker позволяет упаковать приложение и все его зависимости в контейнер, обеспечивая совместимость и предсказуемость среды разработки и продакшн. Kubernetes, в свою очередь, предоставляет мощные инструменты для масштабирования, управления и автоматизации развертывания контейнеризированных приложений. Эти технологии являются ключевыми элементами для реализации надежного и масштабируемого процесса CI/CD.

# 5.4 Проектирование

**Архитектура процесса CI/CD**

Для обеспечения эффективного процесса CI/CD была спроектирована следующая архитектура:

1. Репозиторий с исходным кодом (GitHub):

- Исходный код проекта хранился в GitHub, что предоставляло все необходимые возможности для организации работы с кодом, включая управление версиями, пулл-реквесты и контроль за изменениями.

- В GitHub также была настроена структура ветвления, которая поддерживала параллельную работу над новыми функциями и исправлением ошибок. Основными ветками были `main` для стабильных релизов и `develop` для активной разработки.

2. GitHub Actions для автоматизации CI/CD:

- Для автоматизации всех процессов CI/CD были выбраны GitHub Actions, так как они обеспечивают тесную интеграцию с репозиторием и поддерживают запуск различных пайплайнов на основе событий (например, пуш, пулл-реквесты и т.д.).

- Основной пайплайн CI/CD состоял из нескольких этапов:

1. Сборка проекта: Этот этап запускался при каждом коммите или пулл-реквесте. GitHub Actions автоматически загружал код, устанавливал зависимости и выполнял сборку проекта.

2. Юнит-тестирование: После успешной сборки проект проходил автоматическое юнит-тестирование. GitHub Actions запускал тесты и проверял, что все они прошли без ошибок.

3. Интеграционное тестирование: Следующим этапом было интеграционное тестирование, которое проверяло взаимодействие различных модулей системы.

4. Сборка Docker-образа: После успешного прохождения всех тестов, создавался Docker-образ приложения. Это обеспечивало предсказуемую среду для развертывания на любых серверах.

5. Развертывание: В случае успешного выполнения предыдущих шагов, Docker-образ автоматически разворачивался на тестовый сервер. Если все проверки на тестовом сервере проходили успешно, образ разворачивался на продакшн-сервер.

3. Docker для контейнеризации:

- Docker был выбран для упаковки приложения и его зависимостей в контейнер. Это обеспечивало предсказуемость и стабильность работы приложения на различных средах, минимизируя вероятность возникновения проблем, связанных с различиями в конфигурации серверов.

- Для каждого коммита и релиза создавался новый Docker-образ, что позволяло легко управлять версиями приложения и поддерживать возможность отката к предыдущим версиям в случае необходимости.

4. Среда развертывания (тестовая и продакшн):

- Проект был развернут в двух средах: тестовой и продакшн. Тестовая среда использовалась для проверки новых версий приложения до их выпуска в продакшн. Это позволяло выявить и устранить возможные ошибки и проблемы на ранних этапах.

- Продакшн-среда обеспечивала стабильную работу интернет-магазина для конечных пользователей. Развертывание в продакшн происходило только после успешного прохождения всех этапов тестирования.

**Проектирование рабочего процесса**

Рабочий процесс был спроектирован таким образом, чтобы минимизировать ручные операции и обеспечить высокую степень автоматизации:

1. Пайплайн CI:

- Каждый коммит в ветку `develop` автоматически запускал пайплайн CI. Этот процесс включал сборку и тестирование приложения, обеспечивая немедленное выявление ошибок на этапе разработки.

2. Пайплайн CD:

- После успешного завершения пайплайна CI, пайплайн CD выполнял развертывание приложения на тестовую среду. Если тестирование на этой среде проходило успешно, команда могла принять решение о развертывании на продакшн-среду.

3. Стратегия развертывания:

- Для продакшн-среды был выбран подход с постепенным развертыванием (canary deployment), что позволяло выпускать обновления постепенно, минимизируя риски для конечных пользователей.

4. Безопасность и контроль доступа:

- В проекте были настроены роли и права доступа, которые ограничивали возможность выполнения развертываний и внесения изменений в критически важные части инфраструктуры только для определенных членов команды.

# 5.5 Реализация

Настройка репозитория и GitHub Actions

1. Конфигурация репозитория в GitHub:

- Для начала, репозиторий проекта на GitHub был структурирован таким образом, чтобы поддерживать выбранную стратегию ветвления. Основные ветки включали `main` для стабильных релизов и `develop` для активной разработки.

- Для удобства управления версиями и автоматического создания релизов была настроена система тегов.

2. Создание и настройка GitHub Actions:

- Были созданы YAML-файлы конфигурации для GitHub Actions, которые описывали пайплайны CI/CD. Эти конфигурации включали все необходимые шаги для автоматизации сборки, тестирования и развертывания приложения.

- Например, файл `.github/workflows/ci.yml` содержал следующий базовый набор шагов:

- Checkout кода: получение исходного кода из репозитория.

- Установка зависимостей: запуск скриптов для установки всех необходимых библиотек и пакетов.

- Сборка проекта: запуск сборки проекта на основе полученного кода и зависимостей.

- Тестирование: автоматический запуск юнит- и интеграционных тестов.

- Для CD был создан отдельный workflow, который запускался после успешного завершения CI и занимался развертыванием Docker-образа в тестовую среду.

Использование Docker для контейнеризации

1. Создание Dockerfile:

- В корневой директории проекта был создан `Dockerfile`, который описывал шаги по созданию Docker-образа для приложения. В Dockerfile были указаны следующие ключевые этапы:

- Выбор базового образа: выбор официального образа для выбранного стека (например, для Go или Node.js).

- Копирование исходного кода: перенос кода проекта в контейнер.

- Установка зависимостей и сборка: выполнение команд для установки зависимостей и сборки проекта внутри контейнера.

- Настройка запуска приложения: указание команд для запуска приложения в контейнере.

2. Создание и управление Docker-образами:

- На каждом этапе CI/CD пайплайна после успешного прохождения тестов создавался новый Docker-образ приложения. Эти образы сохранялись в Docker Registry (например, Docker Hub или GitHub Packages) для последующего использования при развертывании.

- Образы версионировались автоматически с помощью тегов, что позволило легко управлять разными версиями приложения и откатываться к предыдущим версиям при необходимости.

3. Развертывание и интеграция

1. Развертывание на тестовую среду:

- После создания Docker-образа и успешного прохождения всех тестов, образ автоматически разворачивался на тестовую среду. Для этого в конфигурации GitHub Actions был добавлен шаг по развёртыванию, включающий использование Docker Compose или Kubernetes.

- Тестовая среда использовалась для проверки новых функций и интеграций перед их выпуском в продакшн.

2. Развертывание на продакшн-среду:

- Если тестирование на тестовой среде проходило успешно, команда могла вручную инициировать развертывание на продакшн-сервер. Для этого в GitHub Actions был настроен отдельный workflow, который использовал Docker-образ, прошедший тесты.

- В качестве подхода к развертыванию был использован canary deployment, что позволило постепенно вводить изменения, минимизируя риски.

# 5.6 Тестирование

1. Тестирование процесса CI/CD:

- После завершения настройки всех компонентов, процесс CI/CD был протестирован на реальных сценариях. Это включало проверку успешности автоматической сборки, тестирования и развертывания приложения в различных условиях.

2. Оптимизация пайплайнов:

- По результатам тестирования были внесены изменения в конфигурацию GitHub Actions и Docker, чтобы ускорить выполнение пайплайнов и минимизировать время на развертывание.

- Например, была оптимизирована сборка Docker-образов за счет кэширования промежуточных слоев, а также настроены более эффективные стратегии работы с тестовыми данными.

# ЗАКЛЮЧЕНИЕ

Прохождение технологической практики в компании WebDad стало для меня ценным опытом, позволившим глубже понять процессы разработки программного обеспечения и применить на практике современные технологии и инструменты. В течение месяца я работал над проектом интернет-магазина, участвуя в различных этапах разработки и внедрения, начиная от написания кода и заканчивая настройкой автоматизированных процессов развертывания.

В ходе практики я освоил и успешно применил такие технологии, как Go и его фреймворки для разработки бекенда, Vue.js для создания интерактивного пользовательского интерфейса, а также системы контроля версий Git и контейнеризацию с Docker. Одним из ключевых направлений моей работы стало внедрение процесса непрерывной интеграции и доставки (CI/CD), для чего были использованы GitHub Actions и Docker. Это позволило автоматизировать важные процессы сборки, тестирования и развертывания, что значительно повысило эффективность разработки и качество выпускаемого продукта.

Практика дала мне возможность не только развить технические навыки, но и глубже понять организационные аспекты разработки, такие как управление версиями, работа в команде и поддержание качества кода. Важной частью работы стало документирование всех этапов проекта, что является неотъемлемой частью профессионального подхода к разработке программного обеспечения.

Подводя итоги, можно сказать, что практика в компании позволила мне приобрести ценный опыт, который будет полезен в дальнейшей карьере. Я улучшил свои навыки программирования, научился эффективно использовать современные инструменты и технологии, а также укрепил понимание важности автоматизации и организационных процессов в разработке программного обеспечения. Эти знания и опыт станут основой для моего дальнейшего профессионального роста и успешной работы в сфере IT.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Джей Макгаврен. "Head First. Изучаем Go" – СПб.: Питер, 2020. – 544 с.

2. Моуэт Эдриен . "Использование Docker" – М.: ДМК-Пресс, 2017 г. – 354 с.

3. Effective Go [Электронный ресурс] — Режим доступа: https://go.dev/doc/effective\_go — Дата доступа: 20.06.2024.

4. Руководства для GitHub Actions [Электронный ресурс] — Режим доступа: https://docs.github.com/ru/actions/guides — Дата доступа: 20.06.2024.