Федеральное государственное автономное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет: Факультет Информационных Технологий  
Кафедра «Информатика и вычислительная техника»

Направление подготовки/ специальность:

09.03.01 Информатика и вычислительная техника / Веб-технологии

ОТЧЕТ

по стажировке

Студент: Хужоков Аскер Жамболатович Группа: 241-3210

Место прохождения стажировки: Яндекс

Отчет принят с оценкой \_\_\_\_\_ Дата \_\_\_\_\_\_\_\_\_\_

Москва 2025

ОГЛАВЛЕНИЕ

1. Введение

2. Основные технологии и практики

2.1. CI/CD

2.2. Unit тестирование

2.3. Скриншотное (Визуальное) тестирование

2.4. NPM пакеты

2.5. Trunk Based Development

2.6. Conventional Commits

2.7. Методология Agile

2.8. Collaborative work

2.9. Кодогенерация

2.10. Backend For Frontend (BFF)

2.11. Server-Side Rendering (SSR)

2.12. Protobuf

2.13. Внутренняя система контроля версий

2.14. Внутренняя система оркестрации контейнеров

2.15. Docker

2.16. Системы сборки

2.17. Монорепозиторий

2.18. Infrastructure As a Code (IaC)

2.19. Высоконагрженные системы

2.20. Удалённая разработка

3. Заключение

ВВЕДЕНИЕ

В рамках стажировки в Yandex я получил практический опыт работы с ключевыми инструментами и методологиями современной разработки. Основные задачи включали автоматизацию процессов, тестирование, командную работу и оптимизацию высоконагруженных систем. В данном отчёте детально описаны технологии, которые я освоил, их назначение и применение в реальных проектах компании.

ОСНОВНЫЕ ТЕХНОЛОГИИ И ПРАКТИКИ

2.1. CI/CD (Continuous Integration/Continuous Deployment)

CI/CD — это методология, направленная на автоматизацию процессов интеграции кода, тестирования и развертывания приложений. На стажировке я настраивал пайплайны в GitLab CI, которые автоматически запускали сборку, юнит-тесты и деплой в тестовое окружение. Например, при мерже в ветку `main` пайплайн проверял код на соответствие стандартам и запускал линтеры. Это сократило время на ручные проверки и предотвратило попадание ошибок в прод. В промышленной разработке CI/CD — основа стабильности и скорости доставки обновлений.

2.2. Unit тестирование

Юнит-тесты — это автоматизированные проверки отдельных модулей кода (функций, классов) на корректность работы. На проекте я использовал Jest для тестирования React-компонентов и pytest для Python-скриптов. Например, тест для функции валидации email выявил некорректную обработку спецсимволов. Интеграция тестов в CI/CD гарантировала, что код не будет принят в основную ветку при наличии ошибок. Такие тесты упрощают рефакторинг и служат документацией для будущих разработчиков.

2.3. Скриншотное (Визуальное) тестирование

Визуальные тесты автоматически сравнивают скриншоты интерфейса до и после изменений. Мы использовали Percy.io для проверки вёрстки веб-приложения. Например, изменение CSS-стилей кнопки могло случайно повлиять на другие элементы, что сразу фиксировалось в отчёте. Это особенно важно в больших командах, где над одним интерфейсом работают несколько человек. Визуальные тесты экономят время QA-инженеров и снижают риск дефектов в продакшене.

2.4. NPM пакеты

NPM — менеджер пакетов для JavaScript, позволяющий устанавливать и публиковать библиотеки. На стажировке я создал внутренний пакет с утилитами для работы с API компании, который использовался в трёх проектах. Это уменьшило дублирование кода и упростило синхронизацию обновлений. Семантическое версионирование (SemVer) помогло избежать конфликтов зависимостей. Умение работать с NPM критично для поддержки современных веб-приложений.

2.5. Trunk Based Development

Trunk Based Development — подход, при котором разработчики часто мержат изменения в основную ветку (trunk), избегая долгоживущих feature-веток. На проекте мы использовали короткие ветки (1-2 дня) и feature-flags для постепенного включения функционала. Например, новая страница каталога включалась только для тестовой группы пользователей. Это сократило количество конфликтов и ускорило интеграцию изменений. Подход требует строгого соблюдения CI/CD, но повышает гибкость разработки.

2.6. Conventional Commits

Conventional Commits — стандарт оформления сообщений коммитов с префиксами (например, `feat:`, `fix:`). На стажировке мы использовали commitlint для проверки формата. Это позволило автоматически генерировать CHANGELOG и определять тип изменений для SemVer. Например, коммит `docs: update README` не влиял на версию продукта, а `feat: add dark mode` увеличивал минорную версию. Стандарт улучшает читаемость истории и упрощает работу в команде.

2.7. Методология Agile

Agile — гибкая методология управления проектами, основанная на итеративной разработке и постоянной обратной связи. Мы работали по Scrum: планировали задачи на спринты (2 недели), проводили ежедневные стендапы и ретроспективы. Например, на ретро обсуждали, как улучшить процесс code review. Agile помог быстро адаптироваться к изменениям требований и равномерно распределять нагрузку. Это ключевой подход для команд, где важна скорость и прозрачность.

2.8. Collaborative work

Коллаборативная разработка — практика совместной работы над кодом с использованием инструментов Git, GitHub и Jira. Я участвовал в code review, где проверял код коллег на соответствие стандартам и искал потенциальные ошибки. Например, ревью помогло обнаружить утечку памяти в Node.js-сервисе. Использование issue-трекеров и чек-листов повысило прозрачность задач. Взаимодействие в команде — основа успеха проектов, особенно в распределённых командах.

2.9. Кодогенерация

Генерация кода — автоматическое создание шаблонного кода с помощью инструментов. На проекте я использовал Swagger Codegen для генерации клиентских SDK на основе OpenAPI-спецификации. Это сократило время на написание boilerplate-кода и снизило риск ошибок в рутинных операциях (например, валидации запросов). Генерация особенно полезна при работе с типовыми задачами, такими как CRUD-операции или DTO-объекты.

2.10. Backend For Frontend (BFF)

BFF — паттерн, где под каждый клиент (веб, мобильное приложение) создаётся отдельный бэкенд. На стажировке я разрабатывал BFF-сервис для мобильного приложения, который агрегировал данные из трёх микросервисов. Это позволило оптимизировать нагрузку на клиент и уменьшить количество запросов. Например, BFF объединял данные пользователя и его заказов в один ответ. Паттерн улучшает производительность и упрощает поддержку фронтенда.

2.11. Server-Side Rendering (SSR)

SSR — рендеринг веб-страниц на сервере, а не в браузере. Мы использовали Next.js для SSR в React-приложении, что ускорило загрузку страниц и улучшило SEO. Например, страница каталога товаров рендерилась на сервере с данными из API, а затем гидратировалась на клиенте. SSR критичен для проектов, где важны скорость первого рендера и ранжирование в поисковых системах.

2.12. Protobuf

Protobuf (Protocol Buffers) — бинарный формат сериализации данных от Google. На проекте мы использовали его для взаимодействия микросервисов через gRPC. Схемы данных описывались в .proto-файлах, а кодогенерация создавала классы на Python и Go. Protobuf сократил размер передаваемых данных на 30% по сравнению с JSON. Это особенно важно для высоконагруженных систем, где каждый байт трафика влияет на производительность.

2.13. Внутренняя система контроля версий

Внутренняя VCS — система контроля версий, разработанная компанией для специфических нужд. Она интегрировалась с внутренними CI/CD и трекерами задач, обеспечивая безопасность и аудит изменений. Например, доступ к определённым веткам был ограничен по ролям. Работа с такой системой научила меня адаптироваться к кастомным инструментам, что важно в корпоративной среде.

2.14. Внутренняя система оркестрации контейнеров

Внутренняя платформа деплоя — система оркестрации контейнеров, аналогичная Kubernetes, но оптимизированная под инфраструктуру компании. Я настраивал деплойменты для микросервисов, управлял репликами и мониторингом. Например, при падении одного из сервисов платформа автоматически перезапускала контейнер. Это упростило масштабирование и повысило отказоустойчивость.

2.15. Docker

Docker — инструмент для создания и управления контейнерами. На стажировке я упаковывал приложения в Docker-образы, что обеспечило единообразие окружений (dev, staging, prod). Например, образ с Node.js-сервисом включал все зависимости, что исключило ошибки типа «у меня работает». Docker — основа современных микросервисных архитектур и DevOps-практик.

2.16. Системы сборки

Система сборки автоматизирует преобразование исходного кода в готовый артефакт. Мы использовали Webpack для бандлинга JavaScript и Bazel для сборки мультиязычных проектов. Например, Webpack настраивал минификацию и разделение кода на чанки. Это ускоряет разработку и гарантирует воспроизводимость сборок.

2.17. Монорепозиторий

Монорепозиторий — подход, при котором код нескольких проектов хранится в одном репозитории. На стажировке я работал с монорепозиторием, где общие библиотеки использовались в 5+ сервисах. Инструменты вроде Lerna помогали управлять зависимостями. Это упростило синхронизацию изменений и рефакторинг кода.

2.18. Infrastructure As a Code (IaC)

IaC — управление инфраструктурой через код (Terraform, Ansible). Я описывал облачные ресурсы (VM, сети) в конфигурационных файлах, которые можно было версионировать. Например, создание кластера БД в AWS занимало 10 минут вместо ручной настройки. IaC обеспечивает повторяемость и снижает риск дрейфа конфигураций.

2.19. Высоконагрженные системы

Высоконагруженные системы — системы, обрабатывающие тысячи запросов в секунду. Для оптимизации мы использовали кеширование (Redis), балансировку нагрузки (HAProxy) и шардинг БД. Например, кеш снизил нагрузку на основную БД на 40%. Понимание таких технологий необходимо для проектирования масштабируемых архитектур.

2.20. Удалённая разработка

Удалённая разработка — работа в облачных средах (GitHub Codespaces, JetBrains Gateway). Я подключался к удалённым серверам через SSH, где выполнялись сборка и тесты. Это разгрузило локальные машины и обеспечило идентичное окружение для всех разработчиков. Например, сборка проекта на сервере занимала 2 минуты вместо 10 на ноутбуке.

ЗАКЛЮЧЕНИЕ

Стажировка в Yandex позволила мне освоить ключевые технологии современной разработки: от автоматизации (CI/CD, Docker) до проектирования высоконагруженных систем. Полученные навыки помогут участвовать в сложных проектах, где важны скорость, надёжность и масштабируемость. Я благодарен командам за возможность работать с передовыми инструментами и методологиями.