**СОДЕРЖАНИЕ**

[СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ 5](#_Toc136335464)

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 6](#_Toc136335465)

[ВВЕДЕНИЕ 8](#_Toc136335466)

[1 ТЕОРЕТИЧЕСКИЙ ОБЗОР 10](#_Toc136335467)

[1.1 Актуальность и практическая значимость 10](#_Toc136335468)

[1.2 Обзор аналогов 12](#_Toc136335469)

[1.3 Обзор используемых инструментов 13](#_Toc136335470)

[1.4 Обзор средств реализаций 19](#_Toc136335471)

[1.5 Схема цифровой подписи EdDSA 21](#_Toc136335472)

[1.6 Вывод по теоретическому обзору 23](#_Toc136335473)

[2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ 24](#_Toc136335474)

[2.1 Этапы разработки и развёртывания приложения 24](#_Toc136335475)

[2.2 Архитектура высокого уровня 27](#_Toc136335476)

[2.3 Диаграмма компонентов 34](#_Toc136335477)

[2.4 Диаграмма вариантов использования 35](#_Toc136335478)

[2.5 Диаграмма классов анализа 36](#_Toc136335479)

[3 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ 37](#_Toc136335480)

[3.1 Этап разработки 37](#_Toc136335481)

[3.2 Этап CI/CD 41](#_Toc136335482)

[3.3 Этап развёртывания 42](#_Toc136335483)

[3.4 Этап мониторинга 45](#_Toc136335484)

[3.5 Тестирование 47](#_Toc136335485)

[ЗАКЛЮЧЕНИЕ 48](#_Toc136335486)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 49](#_Toc136335487)

# **СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

K8S – Kubernetes

CI/CD – Continuous integration / continuous delivery или continuous deployment (Непрерывная интеграция / Непрерывная доставка или Непрерывное развёртывание)

AWS – Amazon Web Services (Сервисы облачных вычислений Amazon)

EdDSA – Edwards-curve Digital Signature Algorithm (Алгоритм цифровой подписи кривой Эдвардса)

# **ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ**

*Сервис* *—* в микросервисной архитектуре, сервис представляет собой небольшую, независимую компоненту программного обеспечения, которая выполняет определенную функцию. Каждый сервис должен иметь чётко определенную границу, которая отделяет его от других сервисов. Это позволяет разрабатывать, тестировать и развертывать каждый сервис независимо от других компонентов приложения.

Кластер *(англ. cluster)* — это группа взаимосвязанных узлов или серверов, которые работают вместе для предоставления определенного сервиса или набора сервисов. Каждый узел в кластере обычно представляет собой отдельный экземпляр микросервиса, и они взаимодействуют друг с другом через четко определенные API.

*Контейнер (англ. container)* *—* это лёгкий и автономный исполняемый пакет, который содержит все компоненты, необходимые для запуска приложения, включая код приложения, библиотеки и другие зависимости. Контейнеры обычно работают в изолированной среде, что обеспечивает надёжность и безопасность приложения.

*Под (англ. pod)* *—* это абстрактный объект, который представляет собой логическую группу одного или нескольких контейнеров. Внутри пода контейнеры работают вместе, используя общие сетевые ресурсы и хранилище. Kubernetes не управляет контейнерами непосредственно, а управляет подами, которые в свою очередь могут содержать один или несколько контейнеров.

*CI/CD —* это сокращение от непрерывная интеграция/непрерывное развёртывание, что является практикой разработки программного обеспечения, которая направлена на автоматизацию и ускорение процесса разработки, тестирования и развёртывания приложений.

*Непрерывная интеграция* *—* это процесс автоматической сборки, тестирования и интеграции кода в общую ветку разработки на основе набора правил и тестов. Это позволяет разработчикам быстро выявлять и исправлять ошибки, улучшать качество кода и минимизировать конфликты при слиянии кода.

*Непрерывная доставка —* это практика в области разработки программного обеспечения, которая заключается в автоматизации процесса доставки (развертывания) приложения из репозитория до окружения, доступного для тестирования и использования конечным пользователям.

*Непрерывное развёртывание —* это процесс автоматического развёртывания приложений после успешного прохождения всех этапов тестирования. Это позволяет быстро и безопасно доставлять новые версии приложений в продакшн, минимизируя время простоя и улучшая качество развёртывания.

*ИИ* *(искусственный интеллект)* *—* это область компьютерной науки, которая занимается созданием интеллектуальных систем, способных выполнять задачи, которые обычно требуют наличия человеческого интеллекта. ИИ может включать в себя различные технологии и методы, такие как машинное обучение, глубокое обучение, нейронные сети, обработку естественного языка, компьютерное зрение и другие.

*Фреймворк (англ. framework**)* *—* набор соглашений, правил и шаблонов, который предоставляет разработчикам компоненты для решения типичных задач. Фреймворк определяет структуру приложения, предоставляет набор инструментов для работы с базами данных, управления пользовательским интерфейсом.

# **ВВЕДЕНИЕ**

Машинное обучение играет все более важную роль во многих аспектах нашей повседневной жизни. Соответственно, спрос на центр хранения наборов данных также растёт, поскольку объем генерируемых и собираемых данных продолжает расти. Возникла необходимость создать сообщество исследователей машинного обучения, где люди могли бы подключаться, хранить свои собственные наборы данных или делиться ими с другими.

Сообщество исследователей машинного обучения, использующих облачные вычисления и безопасное хранилище, будет играть решающую роль в развитии и прогрессе машинного обучения. Оно предоставляет платформу, на которой эксперты и новички могут делиться своими знаниями и опытом. Сообщество будет пространством для этих исследователей, разработчиков и специалистов по данным, чтобы сотрудничать, обмениваться идеями и работать вместе для достижения общих целей. Оно также действует как центр обработки данных, предоставляя центральное место для хранения и управления всеми данными, облегчая доступ к ним и их анализ. Это может помочь исследователям машинного обучения преодолеть проблемы сбора данных, управления, безопасности, обмена, масштабируемости и анализа.

Это веб-приложение сообщества может быть построено с использованием современной микросервис архитектуры и облачных вычислений. Используя облачные технологии и алгоритмы шифрования данных, веб-приложение сообщества может воспользоваться передовыми технологиями для создания высокопроизводительной, отказоустойчивой и безопасной системы.

Используя микросервис архитектуру и платформу облачных вычислений, система может удовлетворить потребности в хранении и пропускной способности больших наборов данных. Расширенные механизмы безопасности, такие как шифрование, обеспечивают безопасность наборов данных и пользовательских данных. Непрерывный мониторинг обеспечивает высокую производительность и доступность. Участники могут искать и загружать наборы данных для использования в своих исследованиях и проектах в области машинного обучения.

*Целью* *работы* является создание и развёртывание общедоступного веб-сайта, который имеет достаточную функциональность для сообщества исследователей моделей машинного обучения и готов обслуживать большое количество пользователей, мог бы выполнять основные функции сообщества, включая обмен наборами данных, моделями обучения и другими статьями знаний, доступ контролировать управление и коммуникацию между исследователями с помощью облачных технологий и обеспечивать безопасность данных.

Для достижения цели были определены следующие *задачи*:

1. обзор предметной области и анализ аналогов;
2. проектирование высокоуровневой архитектуры веб-приложения;
3. выбор подходящих сервисов Amazon Web Services для реализации веб-приложения;
4. использование PostgreSQL в качестве базы данных и применение Redis в качестве уровня кэширования для повышения производительности приложений;
5. использование Sping (Java), Flask (Python) и NestJs (JavaScript) для разностных сервисов на логическом уровне;
6. использование NextJs (JavaScript) для создания дружественного, ориентированного на пользователя пользовательского интерфейса;
7. описание использования веб-приложения и тестирование некоторых вариантов использования;
8. развёртывание и публикация программного продукта в открытом Интернете с использованием Amazon Web Services.

# **1 ТЕОРЕТИЧЕСКИЙ ОБЗОР**

## 1.1 Актуальность и практическая значимость

В последние годы использование машинного обучения и искусственного интеллекта в социальной жизни быстро росло, влияя на различные аспекты нашей повседневной жизни [1]. Он стал важным компонентом современных исследований и технологий, стимулируя инновации во множестве областей. Эти технологии используются, чтобы сделать нашу жизнь более персонализированной, эффективной и безопасной.

С помощью алгоритмов машинного обучения платформы социальных сетей, поисковые системы и другие онлайн-сервисы могут предоставлять нам контент и рекомендации, соответствующие нашим интересам [2]. Машинное обучение также используется в здравоохранении для повышения точности диагностики заболеваний, разработки индивидуальных планов лечения и открытия новых лекарств [3]. В образовании искусственный интеллект и машинное обучение могут персонализировать учебный процесс для учащихся [4], а в финансах эти технологии могут обнаруживать мошенничество, прогнозировать цены на акции и управлять рисками [5]. В сфере транспорта машинное обучение и ИИ могут оптимизировать транспортный поток, уменьшить заторы и повысить безопасность [6].

Однако машинное обучение — не единственное, что быстро растёт и меняется в наше время. С развитием машинного обучения и связанных с ним технологий объем генерируемых данных значительно увеличился [7], что стало одним из главных вызовов для современной информационной технологии. По данным *Statista* [8], каждый день создаётся огромный объём данных — целых **328,77 млн терабайт**. Системы искусственного интеллекта/машинного обучения в значительной степени полагаются на большие объёмы данных для обучения и прогнозирования, и, соответственно, чем больше данных, тем точнее и эффективнее работает такая система.

По мере того, как все больше и больше предприятий и организаций внедряют ИИ и другие технологии, которые генерируют большие объёмы данных, ожидается, что объем генерируемых данных будет только увеличиваться, что создаёт дополнительные вызовы для их хранения, обработки и анализа. Таким образом, важно развивать не только технологии машинного обучения, но и инфраструктуру для хранения и обработки данных, включая технологии облачных вычислений, распределённых систем хранения данных и т.д.

В целом, рост объёма генерируемых данных является одним из главных вызовов для современной информационной технологии, требующим развития новых технологий и инфраструктуры для обработки и анализа данных, а также более эффективных алгоритмов машинного обучения и искусственного интеллекта.

С другой стороны, из-за чрезвычайно быстрого прогресса и растущего внедрения машинного обучения в различных отраслях и дисциплинах возрастает потребность в создании онлайн-платформы сообщества, где исследователи машинного обучения, учёные и студенты могут общаться, сотрудничать и делиться ресурсами для дальнейшего ускорения исследований и разработок в этой важной и быстрорастущей области технологий. Исследователи могут присоединиться к сообществу, создать профиль, поделиться информацией о своей работе и научных интересах. Что ещё более важно, они могут загружать и делиться наборами данных, которые они собрали или курировали. Объединяя наборы данных от многих исследователей, сообщество предоставляет централизованное хранилище данных, которое можно использовать для улучшения моделей машинного обучения и продвижения в этой области.

Специальное сообщество исследователей машинного обучения жизненно важно по нескольким причинам. Во-первых, он способствует сотрудничеству, предоставляя исследователям платформу для обмена идеями, выводами и опираясь на результаты работы друг друга. Это ускоряет разработку новых алгоритмов и методов, в итоге раздвигая границы возможного. Во-вторых, такое сообщество культивирует культуру открытости и прозрачности, что позволяет быстро распространять передовые исследования и воспроизводимость результатов. Это помогает поддерживать научную честность и гарантирует, что исследователи несут ответственность за свои выводы. Наконец, активное сообщество машинного обучения играет важную роль в привлечении и развитии талантов, а также в продвижении междисциплинарных исследований. Объединяя экспертов из разных областей, мы можем решать сложные задачи и раскрывать истинный потенциал машинного обучения, в итоге принося пользу обществу в целом.

Подводя итог, экспоненциальное распространение машинного обучения и ИИ в нашей повседневной жизни в сочетании с огромным объёмом генерируемых данных подчёркивает необходимость сообщества исследователей машинного обучения. Используя облачные вычисления, веб-приложение для исследователей может эффективно хранить огромные объёмы данных, обеспечивать исключительный пользовательский опыт благодаря первоклассной производительности и обеспечивать безопасность с помощью надёжных инструментов безопасности.

## 

## 1.2 Обзор аналогов

Машинное обучение играет все более важную роль во многих аспектах нашей повседневной жизни. Соответственно, спрос на наличие центра хранения наборов данных также растёт, поскольку объем данных, генерируемых и собираемых, продолжает расти. Возникла необходимость создать сообщество для исследователей машинного обучения, где люди могут подключаться, хранить свой собственный набор данных или делиться им с другими. И есть несколько вебсайтов, которые предоставляют некоторые функции, такие как Kaggle может хранить наборы, Model Zoo предоставляет некоторые модели, Towards Data Science есть статьи и лекции и т. д. Однако до настоящего времени не существует единого централизованного приложения, которое бы обладало всеми необходимыми функциями, использовало современные технологии, включая облачные вычисления, и обеспечивало высокий уровень доступности, масштабируемости и безопасности.

Ниже представлена таблица 1.1, которая включает в себя функциональность для каждого из этих веб-сайтов.

Таблица 1.1 – Сравнительная таблица

|  |  |  |  |
| --- | --- | --- | --- |
| **Функциональность** | **Kaggle [9]** | **ModelZoo [10]** | **Towards Data Science [11]** |
| Общий доступ к набору данных | + | **–** | **–** |
| Совместное использование моделей машинного обучения | + | + | **–** |
| Управление загруженной моделью | + | **–** | **–** |
| Создание статьи | **–** | **–** | **+** |
| Карьерный путь в области машинного обучения | **–** | **+** | **+** |
| Иметь сообщество | **+** | **–** | **+** |
| Высокая производительность и безопасность. | **+** | **–** | **–** |
| Русскоязычная локализация | **–** | **–** | **–** |

После сравнения Kaggle, Model Zoo и Toward Data Science, мы сделали вывод, что данные платформы имеют ограниченный охват в качестве сообщества для исследований в области машинного обучения. Например, Kaggle в основном занимается хранением и управлением данными, в то время как Model Zoo занимается сбором и организацией моделей, а Toward Data Science — построением карьеры учёного.

Кроме того, мы обнаружили, что не все они построены по новейшим технологиям, которые необходимы для работы с большим количеством пользователей одновременно. К сожалению, все они также не имеют русскоязычной локализации, что может быть проблемой для русскоязычных пользователей.

Исходя из этого, мы можем заключить, что для эффективного создания сообщества исследователей в области машинного обучения необходимо разработать инновационное решение с достаточным функционалом, высокой производительностью и безопасностью. Также важно, чтобы платформа была локализована на русский язык, чтобы сделать **ещё** доступной для широкой аудитории пользователей в России.

## 1.3 Обзор используемых инструментов

**Docker** — это платформа для контейнеризации приложений с открытым исходным кодом, которая позволяет упаковывать приложения и их зависимости в легковесные и переносимые контейнеры (рисунок 1.1). Контейнеры Docker похожи на виртуальные машины, но они более легковесны и используют общее ядро операционной системы хоста, что делает их более эффективными и экономичными в использовании ресурсов.

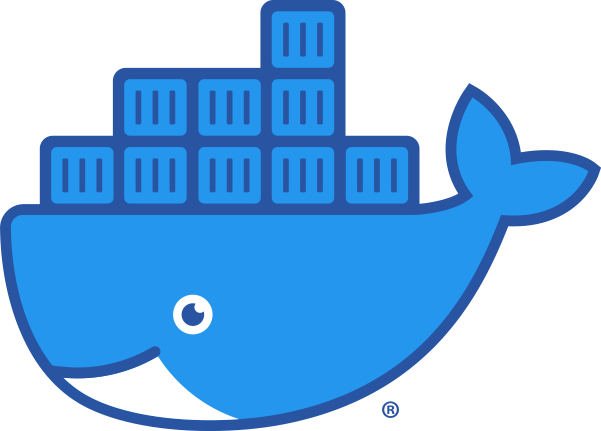


Рисунок 1.1 – Docker

Контейнеры Docker позволяют разработчикам упаковывать приложения вместе со всеми необходимыми компонентами, включая операционную систему, библиотеки и другие зависимости, в единую единицу, которую можно развернуть на любой машине, поддерживающей Docker. Это делает процесс развёртывания и масштабирования приложений более простым и быстрым.

На практике мы используем Docker только для создания контейнера при разработке нашего веб-приложения. Мы не будем использовать Docker для развёртывания веб-приложения в качестве рабочей среды. На самом деле мы будем использовать другой инструмент под названием Kubernetes. Однако, поскольку образы в Docker также соответствуют спецификации образа OSI (OSI image-spec), поэтому образы, созданные из Dockerfile, также имеют преимущество.

**Kubernetes** (K8s) — это современная и популярная платформа для автоматизации развертывания, масштабирования и управления контейнеризованными приложениями (рисунок 1.2). Он был создан компанией Google и предоставляется как открытое программное обеспечение. Kubernetes обеспечивает унифицированное управление контейнерами, что позволяет разработчикам легко развернуть и масштабировать приложения в различных средах, включая облачные и локальные инфраструктуры. Он также обладает мощной системой управления ресурсами, которая позволяет эффективно использовать доступные ресурсы и предотвращать их перегрузку.

A white ship wheel on a blue hexagon

Description automatically generated with medium confidence

Рисунок 1.2 – Kubernetes

Для развертывания нашего веб-приложения мы планируем использовать Kubernetes. В основе нашего подхода лежит создание образа приложения из Docker, который позволит нам легко создавать и управлять подами с контейнерами. Как только мы создадим образ приложения, мы будем использовать Kubernetes для создания пода с контейнером и управления им. Это позволит нам быстро и легко масштабировать приложение, добавляя дополнительные контейнеры при необходимости. Кроме того, мы сможем оркестровать наши контейнеры, что обеспечит высокую доступность и отказоустойчивость приложения.

Итак, использование Kubernetes для развертывания и управления нашим веб-приложением позволит нам создавать и масштабировать контейнеры быстро и легко, обеспечивать высокую доступность и отказоустойчивость приложения, а также сосредоточиться на разработке, в то время как Kubernetes автоматизирует процессы развертывания и управления.

**Prometheus** — это система мониторинга и алертинга с открытым исходным кодом, разработанная для обнаружения и решения проблем в распределённых приложениях и системах (рисунок 1.3). Она была создана в компании SoundCloud и затем перешла под управление Cloud Native Computing Foundation (CNCF).

A picture containing clipart, creativity

Description automatically generated

Рисунок 1.3 – Prometheus

Одна из ключевых особенностей Prometheus — это возможность настраивать правила алертинга на основе определенных условий, которые могут указывать на проблемы в системе. Когда эти условия выполняются, Prometheus генерирует и отправляет уведомления администраторам о проблеме, что позволяет им быстро реагировать и решать проблемы в системе.

В нашем веб-приложении я буду использовать Prometheus, чтобы передать все метрики приложения и настроить оповещение, чтобы быстро уведомлять нас о выдаче Telegram всякий раз, когда сервер проявляет подозрительную активность или даже аварийно завершает работу. Это поможет нам быстро восстановить сеть в случае сбоя и предотвратить появление проблемы.

**Grafana** - это современный и популярный инструмент для визуализации и мониторинга данных (рисунок 1.4). Он позволяет создавать красивые и информативные дашборды, настраивать уведомления и алерты, а также анализировать данные в реальном времени. Grafana поддерживает множество источников данных, таких как базы данных, облачные сервисы, системы мониторинга, API и другие.



Рисунок 1.4 – Grafana

Grafana будет использоваться для визуализации показателей нашего веб-развёртывания. В сочетании с Prometheus эти 2 инструмента помогут использовать визуальное представление приложения, отслеживая и контролируя Интернет. В случае возникновения какой-либо проблемы мы могли бы легко определить и предотвратить или устранить её.

**Fluentd** — это мощный инструмент с открытым исходным кодом, который позволяет пользователям собирать, объединять и передавать журналы и другие данные в режиме реального времени (рисунок 1.5). Разработанный в Японии в 2011 году, Fluentd быстро завоевал популярность в сообществах DevOps и Big Data благодаря своей эффективности и универсальности. В своей работе я буду использовать Fluentd для загрузки логов из развернутых контейнеров на сервер Elasticsearch. С Fluentd я могу легко собирать и передавать журналы, созданные приложением, на сервер Elasticsearch, что позволяет анализировать данные и управлять ими в реальном времени.



Рисунок 1.5 – Fluentd

**Elasticsearch** — это мощная поисковая и аналитическая система с открытым исходным кодом, основанная на Apache Lucene (рисунок 1.6). Он специально разработан для хранения, поиска и анализа больших объемов данных в режиме реального времени. Разработанный Elastic в Швейцарии в 2010 году, Elasticsearch быстро завоевал популярность в сообществах Big Data и DevOps благодаря своим надежным функциям и высокой производительности. В моей дипломной работе журналы будут храниться в Elasticsearch после загрузки Fluentd, что позволит эффективно управлять огромными объемами данных, генерируемых приложением.



Рисунок 1.6 – Elasticsearch

**Kibana** — это мощный инструмент исходного кода, который позволяет пользователям анализировать данные, хранящиеся в Elasticsearch, популярной поисковой и аналитической системе (рисунок 1.7). Разработанный Elastic в 2013 году, Kibana быстро завоевал популярность в сообществах Big Data и DevOps. В моей работе Kibana используется для отображения журналов, хранящихся в Elasticsearch, обеспечивая четкое и лаконичное представление данных. Благодаря интуитивно понятному интерфейсу и надежным функциям Kibana является ценным инструментом для всех, кому необходимо эффективно и действенно анализировать и визуализировать большие объемы данных.

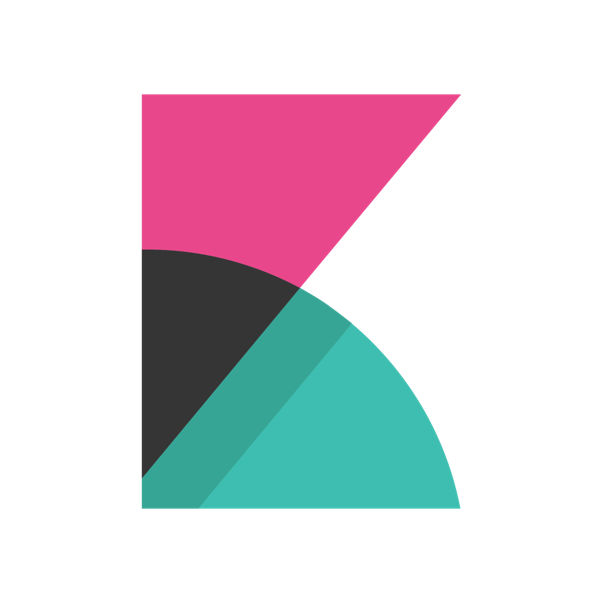


Рисунок 1.7 – Kibana

**PostgreSQL**, также известная как Postgres SQL, является широко используемой и высоко оцененной системой управления реляционными базами данных (СУБД) с открытым исходным кодом (рисунок 1.8). Она известна своими мощными функциями и надежностью, что делает ее одной из самых популярных и надежных СУБД в мире. Учитывая его преимущества, PostgreSQL является основной базой данных, используемой в моем приложении, обеспечивая прочную основу для управления и хранения огромных объемов данных, генерируемых приложением. PostgreSQL с его проверенной репутацией и обширными возможностями является ценным активом для любой организации, которой требуется масштабируемое и надежное решение для базы данных.

A blue elephant with white outline

Description automatically generated with low confidence

Рисунок 1.8 – PostgreSQL

**Git** - это распределенная система управления версиями, которая используется для отслеживания изменений в исходном коде и совместной работы над проектами (рисунок 1.9). Git был создан Линусом Торвальдсом в 2005 году и стал одним из самых популярных инструментов разработки программного обеспечения. Он позволяет разработчикам работать с исходным кодом как локально, так и удаленно, обеспечивая быстрый доступ к версиям кода, возможность отмены изменений и слияние изменений из разных веток.



Рисунок 1.9 – Git

В этой работе я буду использовать Git для управления версией кода и развёртыванием. Благодаря конвейеру CI/CD в GitHub я мог управлять изменением кода и быстро проверять новую версию веб-приложения с развёртыванием автоматизированного тестирования.

**AWS** (Amazon Web Services) — это облачная платформа, которая предоставляет широкий спектр услуг для облачных вычислений, хранения данных, аналитики, машинного обучения, Интернета вещей (IoT) и других целей (рисунок 1.10). AWS была создана Amazon.com в 2006 году и является одной из самых популярных облачных платформ в мире.



Рисунок 1.10 – Amazon Web Services

AWS позволяет пользователям арендовать виртуальные серверы, хранить данные в облаке, выполнять вычисления и развёртывать приложения без необходимости покупки и поддержки собственного аппаратного обеспечения и инфраструктуры. AWS также предоставляет сервисы для управления базами данных, машинного обучения, аналитики данных, а также инструменты для разработки, тестирования и развёртывания приложений.

AWS предоставляет высокую степень масштабируемости, гибкости и возможностей настройки, что позволяет пользователю выбирать только те услуги, которые необходимы для их конкретных потребностей. AWS также предлагает гибкую систему оплаты, с которой пользователи могут платить только за те услуги, которые они используют, и могут масштабировать необходимые ресурсы по мере роста их бизнеса.

## 1.4 Обзор средств реализаций

В обзоре обсуждаются инструменты реализации, необходимые для разработки веб-приложения для исследователей машинного обучения. Инструменты, рассмотренные для этого проекта, включают Spring, NestJs и NextJs.

**Spring** — это один из самых популярных и мощных фреймворков для разработки приложений на языке Java (рисунок 1.11). Он предоставляет разработчикам множество инструментов и функций для упрощения и ускорения процесса разработки приложений.



Рисунок 1.11 – Spring

**Spring Boot** — это расширение фреймворка Spring, которое позволяет быстро создавать готовые к использованию приложения с минимальной конфигурацией. Spring Boot предоставляет множество удобных функций, таких как автоматическая настройка, управление зависимостями и встроенный сервер приложений. Это позволяет разработчикам сосредоточиться на бизнес-логике приложения, а не на его конфигурации и настройке.

**NestJS** — это фреймворк для разработки серверных приложений на языке TypeScript (рисунок 1.12). NestJS использует модульную структуру приложения, которая позволяет разделить код на небольшие и многократно используемые модули. Он также предоставляет механизмы внедрения зависимостей, которые позволяют легко управлять зависимостями между различными компонентами приложения и сократить связность между ними.



Рисунок 1.12 – NestJs

**NextJs** — это фреймворк для создания серверных приложений на базе ReactJs. Он предоставляет множество инструментов и функций, которые упрощают разработку веб-приложений и позволяют создавать лучшие пользовательские интерфейсы (рисунок 1.13). Одной из главных особенностей NextJs является его поддержка серверного рендеринга, что позволяет приложениям быстро отображаться на стороне сервера. Это улучшает производительность приложений и обеспечивает оптимизацию для поисковых систем.



Рисунок 1.13 – NextJs

Подводя итог, можно сказать, что процесс разработки веб-приложения для исследователей машинного обучения представляет собой сложную задачу, требующую использования нескольких языков реализации и сред.

Что касается логического уровня, будет использоваться комбинация Flask, Spring и NestJs. Для более простых сервисов, таких как управление статьями, достаточно легковесной среды Flask. Однако для более сложных сервисов, включая управление пользователями, управление наборами данных и контроль доступа, необходимы дополнительные возможности Spring. NestJs служит отличным инструментом связи между этими двумя фреймворками.

Для уровня представления NextJs на основе ReactJs предоставляет удобный интерфейс, в котором приоритет отдается высокому удобству пользователя и отличной производительности с хорошим SEO.

Используя эти инструменты в сочетании, можно разработать полнофункциональное веб-приложение для исследователей машинного обучения, которое работает с оптимальной производительностью и безопасностью.

## 1.5 Схема цифровой подписи EdDSA

Схема цифровой подписи EdDSA (Edwards-curve Digital Signature Algorithm) — это алгоритм цифровой подписи, который использует эллиптические кривые [26]. EdDSA использует *эллиптические кривые Эдвардса* в качестве математической основы для создания цифровых подписей. Эллиптические кривые Эдвардса являются особой формой эллиптических кривых, которые были разработаны для упрощения вычислений и улучшения безопасности.

Для создания цифровой подписи с помощью EdDSA, следуйте этим шагам:

1. Генерация ключевой пары: сначала необходимо сгенерировать ключевую пару, состоящую из закрытого и открытого ключей. Закрытый ключ является **секретным ключом**, который используется для создания цифровой подписи, а **открытый ключ** — это публичный ключ, который используется для проверки цифровой подписи. Ключевая пара генерируется случайнымобразом с помощью определенного алгоритма.
2. Создание хеш-кода: Данные, которые будут подписаны, сжимаются с помощью криптографического хеш-алгоритма, такого как *SHA-256*, для создания хеш-кода.
3. Вычисление цифровой подписи: для создания цифровой подписи используется закрытый ключ и хеш-код данных. Алгоритм EdDSA использует определенную эллиптическую кривую, чтобы создать подпись, которая будет уникальна для данного набора данных и ключа.
4. Проверка цифровой подписи: для проверки цифровой подписи необходимо использовать открытый ключ и хеш-код данных. Если цифровая подпись верна, то это означает, что данные не были изменены и что подпись была создана с помощью соответствующего закрытого ключа.

EdDSA использует эллиптические кривые Эдвардса, которые обеспечивают более безопасные и эффективные вычисления, чем другие эллиптические кривые. Он также обладает рядом преимуществ, таких как устойчивость к атакам на основе коллизий, уменьшенные требования к вычислительным ресурсам и лучшую производительность.

EdDSA считается безопасным алгоритмом цифровой подписи, который обладаетвысокой производительностью и устойчивостью к атакам, таким как атаки на основе коллизий или квантовых вычислений. Он также имеет меньше требований к вычислительным ресурсам, чем некоторые другие схемы цифровой подписи, такие как RSA или DSA.

A picture containing text, diagram, font, screenshot

Description automatically generated

Рисунок 1.14 – Шифрование схемы с помощью алгоритма edDSA

В рамках данной работы на рисунке 1.14 демонстрируется схема кодирования файлов, основанная на алгоритме EdDSA, реализованная в разработанном приложении. В ходе процесса реализации было принято решение использовать указанную схему с целью обеспечения шифрования пользовательских наборов данных во время загрузки. Выбор данного алгоритма был обусловлен его способностью к сквозному шифрованию, что в итоге способствует созданию более безопасной среды для хранения и передачи пользовательской информации.

Учитывая, что наборы данных, загружаемые пользователями, невероятно разнообразны по своей природе и охватывают целый ряд типов данных, таких как текст, аудио, изображения, видео и т. д., важно было не только внедрить надёжный метод шифрования, но и обеспечить его может адаптироваться к уникальным характеристикам каждого формата данных. Таким образом, алгоритм EdDSA был изменён для работы с различными типами данных, гарантируя максимально эффективное и надёжное шифрование каждого набора.

Используя этот подход, я могу обеспечить высокий уровень безопасности данных для своих пользователей, гарантируя, что их информация защищена и защищена от любых потенциальных угроз. В целом, моё решение реализовать алгоритм EdDSA таким образом оказалось эффективным и надёжным решением для обеспечения конфиденциальности и безопасности пользовательских данных.

## 1.6 Вывод по теоретическому обзору

На основе обзора теории можно сделать вывод, что создание веб-приложения для исследования машинного обучения является актуальным и важным на практике. Хотя на рынке уже есть приложения с аналогичными функциями, им часто не хватает функциональности, безопасности или производительности.

Для разработки веб-приложения мы планируем использовать различные инструменты для контроля версий, разработки, развёртывания, мониторинга и визуализации. Это не только ускорит процесс разработки, но и упростит отслеживание хода выполнения приложения после его развёртывания. Кроме того, мы выбрали платформы для создания наших сервисов, которые обеспечивают максимальную безопасность и производительность приложения.

# **2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ**

## 2.1 Этапы разработки и развёртывания приложения

В рамках современных тенденций разработки программного обеспечения акцент смещается в сторону автоматизации процессов, что способствует сокращению числа участников, задействованных на разных этапах реализации. Для соответствия этому направлению и облегчения автоматизации процесса разработки, я предлагаю разделение его на четыре ключевых этапа, представленных на рисунке 2.1: этап разработки, этап CI/CD, этап развертывания и этап мониторинга.

A picture containing text, font, logo, screenshot

Description automatically generated

Рисунок 2.1 – 4 этапа в процессе разработки и развёртывания

Четыре вышеупомянутых этапа являются интегральными составляющими процесса реализации веб-приложения, способствуя повышению его эффективности и автоматизации. Данные этапы также важны для обеспечения качества приложения и своевременной его доставки, что является ключевым фактором успеха проекта. На каждом из этапов я отбираю соответствующие инструменты с целью ускорения процесса и его автоматизации. В своей работе используются широко распространенные в отрасли инструменты, рекомендованные специалистами, что позволяет максимально реализовать потенциал автоматизации.

**Этап разработки** является первым и фундаментальным шагом в процессе разработки приложения. На данном этапе я осуществлял сбор и анализ требований, планировал проект, проектировал архитектуру приложения и разрабатывал его код. Определение функциональных и нефункциональных требований представляет собой ключевой элемент этого этапа, поскольку это позволяет мне определить необходимые функции приложения и задачи, которые оно должно выполнять. Кроме того, на данном этапе определяется общая архитектура приложения, включая выбор компонентов и технологий, которые будут использоваться для его реализации. Также я выбирал инструменты и технологии, которые наилучшим образом соответствуют требованиям проекта и позволяют максимально эффективно реализовать его функциональность.

После написания кода приложения на этапе разработки проводятся эксперименты по тестированию и проверке. Целью этого этапа является проверка того, что приложение работает и соответствует требованиям. Для этого весь исполняемый код приложения может быть встроен в образ Docker с помощью Dockerfile, а затем использован инструмент docker-compose, который позволяет мне одновременно запускать несколько подключённых контейнеров. Это также позволяет мне тестировать и экспериментировать с развёртыванием приложения в различных средах, что помогает мне выявлять и устранять ошибки и проблемы на раннем этапе.

**Этап CI/CD** представляет собой второй этап в процессе автоматической сборки, тестирования и доставки приложения. После проведения тестирования и экспериментов с развёртыванием на предыдущем этапе, на этапе CI/CD я использую различные инструменты и технологии, такие как Git в качестве системы контроля версий, GitLab в качестве хоста удалённого репозитория и GitLab Pipeline в качестве инструмента непрерывной интеграции и доставки для автоматизированных тестов. Непрерывная интеграция позволяет мне убедиться, что приложение будет развёрнуто быстро и надёжно.

Цель этого этапа заключается в обнаружении и исправлении ошибок и проблем на раннем этапе, что снижает бизнес-риски и ускоряет доставку приложений. На этом этапе я могу проводить непрерывное тестирование приложения и автоматически выполнять различные проверки на соответствие требованиям. Использование инструментов непрерывной интеграции и доставки позволяет мне автоматизировать процесс сборки, тестирования и доставки приложения, что уменьшает вероятность ошибок и ускоряет процесс доставки приложения в рабочую среду.

**Этап развёртывания** является третьим этапом в процессе разработки приложения, на котором я подготавливаю приложение к запуску в рабочей среде. На этом этапе я использую контейнеры и оркестраторы контейнеров в качестве инструментов для автоматизации процесса развёртывания.

Для создания контейнеров на этом этапе я использую образ Docker, который был сгенерирован на предыдущем этапе. Развёртывание всех контейнеров осуществляется с помощью Kubernetes для оркестровки контейнеров. Использование образа Docker и развёртывание его с помощью Kubernetes обусловлено тем, что образ Docker соответствует формату образа OCI, который является приемлемым для Kubernetes.

Для подключения кластера Kubernetes к Amazon Web Services на данном этапе используется служба под названием Amazon EKS (Amazon Elastic Kubernetes Services). Это позволяет автоматически масштабировать кластер и обеспечивать высокую доступность приложения в рабочей среде.

**Этап мониторинга** является четвёртым этапом в процессе разработки приложения. На этом этапе происходит мониторинг приложения в реальной среде, сбор метрик и журналов, анализ данных и выявление проблем.

Для быстрого выявления и решения проблем, связанных с производительностью, безопасностью и доступностью приложения, на этом этапе я использую инструменты мониторинга и анализа. Для сбора метрик из приложений я использую Prometheus, который экспортирует метрики и собирает их для последующей визуализации в Grafana. Для сбора журналов я использую Fluentd, который отправляет журналы приложений в Elasticsearch для хранения и обработки Kibana.

Этот этап мониторинга позволяет мне убедиться, что приложение имеет высокое качество и быстро реагирует на проблемы. Анализ собранных метрик и журналов помогает мне выявлять проблемы с производительностью приложения, а также выявлять проблемы с безопасностью и доступностью. Это позволяет мне быстро реагировать на проблемы и устранять их до того, как они повлияют на пользователей и бизнес.

В целом разработка и развёртывание приложений включает в себя несколько этапов, начиная от этапа разработки и заканчивая мониторингом сцены. Важно учитывать каждый этап, чтобы обеспечить эффективную и качественную работу приложения.

Использование методов автоматизации и непрерывной интеграции и доставки (CI/CD) помогает мне ускорить разработку и доставку приложений, а мониторинг сцены обеспечивает высокую производительность и надёжность приложения в реальной среде. Эти методы сокращают время, затрачиваемое на разработку и развёртывание приложений, и повышают качество проекта.

Хорошо отлаженный процесс разработки и развёртывания приложений позволяет приложению эффективно реагировать на изменения и динамические потребности бизнеса, тем самым обеспечивая конкурентное преимущество. При этом необходимо учитывать каждый этап и применять современные методы и инструменты, такие как Docker, Kubernetes, Prometheus, Grafana, Fluentd, Elasticsearch и Kibana.

В результате мой процесс разработки и развертывания приложений позволяет сократить время, необходимое для доставки приложений в производство, повысить качество проектов, обеспечить высокую производительность и надёжность приложения в реальной среде, а также быстро реагировать на изменения и потребности бизнеса.2.2 Этапы разработки и развёртывания приложения.

## 2.2 Архитектура высокого уровня

Архитектура высокого уровня (HLA) — это общий вид приложения или системы, который определяет её ключевые компоненты, взаимодействие между ними, и модульную структуру. HLA также учитывает функциональные и нефункциональные требования, и является основой для дальнейшей разработки архитектурных деталей и приложений.

HLA включает в себя описание архитектурных компонентов, таких как серверы, клиенты и базы данных, а также выбор технологий для их реализации, таких как платформы, языки программирования и инструменты. Обычно HLA разрабатывается на ранних этапах жизненного цикла разработки программного обеспечения, на этапе анализа и проектирования, и может быть представлен в виде диаграмм, схем, документации или других форматов, которые могут использоваться для дальнейшей детализации и реализации архитектуры.

Микросервисная архитектура представляет собой подход к созданию приложений, при котором они разбиваются на небольшие и независимые компоненты, называемые микросервисами. Каждый микросервис отвечает за выполнение конкретной функции и может быть разработан, протестирован и развернут независимо от других. Это обеспечивает гибкость и упрощает масштабирование приложений.

Микросервисы взаимодействуют друг с другом через API, что позволяет изменять и обновлять компоненты без влияния на работу других частей приложения. Этот подход обеспечивает более гибкую и эффективную разработку, а также более высокую надежность и масштабируемость приложений.

Связь между компонентами может быть синхронной или асинхронной. В синхронной связи один компонент ожидает ответа от другого компонента перед продолжением своей задачи, а в асинхронной связи компоненты могут взаимодействовать без ожидания ответа друг от друга, что позволяет увеличить производительность и отзывчивость системы.

Хорошо спроектированная связь между компонентами является ключевым фактором для эффективной работы микросервисной архитектуры. Она должна обеспечивать гибкость и удобство внесения изменений в систему, минимизировать риски при обновлении и изменении компонентов, а также обеспечивать высокую производительность и доступность приложения. Для достижения этих целей могут использоваться различные подходы и инструменты, такие как API Gateway, очереди сообщений, асинхронные очереди и другие технологии.

A picture containing text, diagram, font, screenshot

Description automatically generated

Рисунок 2.2 – Схема архитектуры проекта и используемых технологий проекта

Веб-приложение, разработанное в рамках данной ВКР, основано на микросервисной архитектуре, состоящей из шести ключевых сервисов, схематически изображенных на рисунке 2.2:

1. Модуль «Пользовательский интерфейс»;
2. Модуль «Уведомления и подписки»;
3. Модуль «Управление ресурсами»;
4. Модуль «Шифрование данных»;
5. Модуль «База данных»;
6. Модуль «Управление идентификацией и доступом».

Каждый из этих модулей является независимым компонентом, который выполняет определенную функцию в приложении. Модуль "Пользовательский интерфейс" отвечает за взаимодействие с пользователем и предоставляет интерфейс для работы с приложением. Модуль "Уведомление и подписка" отвечает за отправку уведомлений и управление подписками пользователей. Модуль "Управление ресурсами" отвечает за управление ресурсами, используемыми приложением, такими как память и процессорное время. Модуль "Шифрование данных" отвечает за защиту конфиденциальных данных приложения. Модуль "База данных" отвечает за хранение и управление данными приложения. Модуль "Управление идентификацией и доступом" отвечает за управление доступом пользователей к различным функциям приложения.

Каждый из этих модулей может быть разработан, протестирован и развернут независимо от других модулей, что обеспечивает гибкость и упрощает масштабирование приложения. Для обмена данными между модулями используются соответствующие протоколы, такие как HTTP, Web Socket или TCP/IP.

Кроме того, для организации очередей сообщений может использоваться Kafka, а Amazon API Gateway может выступать в качестве шлюза. Такие подходы обеспечивают более гибкую и эффективную разработку, а также более высокую надёжность и масштабируемость приложения, позволяя его компонентам взаимодействовать между собой независимо, что упрощает добавление новых функций и изменения в системе.

A close up of a logo

Description automatically generated with low confidence

Рисунок 2.3 – Модуль «Пользовательский интерфейс»

На рисунке 2.3 изображен модуль «Пользовательский интерфейс», в рамках которого после реализации пользовательского интерфейса я загружаю код в GitLab. Благодаря автоматизированному процессу CI/CD и Amplify, новый развернутый код будет автоматически развернут.

В данном модуле я разрабатываю пользовательский интерфейс, опираясь на свой опыт в области JavaScript и внешней библиотеки NextJs, созданной на основе фреймворка ReactJs. После применения NextJs мне удалось создать высокопроизводительный, полностью адаптивный веб-сайт с оптимизацией для поисковых систем. В процессе проектирования особое внимание уделяется пользовательскому опыту, соблюдению рекомендаций Google и использованию их библиотеки для разработки интерфейса MaterialUI с целью предоставления выдающегося пользовательского опыта.

Я уделяю большое внимание поддержанию высокого стандарта качества кода на протяжении всего процесса разработки. Для этого я реализовал надежный рабочий процесс pre-commit, который гарантирует, что все изменения кода проходят тщательную проверку перед отправкой в staging среду. В частности, перед фиксацией любого нового кода я использую pre-commit hook, который запускает выполнение как Eslint, так и Prettier. Используя эти инструменты в сочетании с моим рабочим процессом перед фиксацией, я могу поддерживать кодовую базу, которая является эффективной и простой для понимания, снижая вероятность ошибок и улучшая общее качество конечного продукта.

Чтобы убедиться, что разработанный пользовательский интерфейс доступен для публики, я внедрил простой процесс развертывания. в частности, я подключил основную ветку репозитория к Amazon Amplify, высоконадежному и масштабируемому сервису для развертывания веб-приложений. В результате каждый раз, когда я отправляю новую фиксацию в GitLab, Amazon Amplify автоматически обновляется до последней версии моего нового пользовательского интерфейса, сводя к минимуму время простоя и предоставляя последнюю версию для общего пользования. Этот оптимизированный процесс стал надежным и эффективным способом развертывания пользовательского интерфейса на общедоступном сайте, а также обеспечил актуальность и высокую функциональность веб-сайта для пользователей.

A picture containing text, font, logo, screenshot

Description automatically generated

Рисунок 2.4 – Модуль «Уведомление и подписка»

Архитектура модуля «Уведомления и подписки» представлена на рисунке 2.4. В рамках данного модуля было принято решение использовать фреймворк Spring в качестве внутреннего сервера, интегрирующегося с очередью сообщений Kafka. Для оркестрации контейнеров при развертывании данного модуля выбрана технология Kubernetes, позволяющая размещать каждый сервис в виде контейнера. В дальнейшем, все контейнеры развертываются с использованием Amazon Elastic Kubernetes Service (EKS), применяемого для развертывания кластеров Kubernetes.

Для развёртывания очереди сообщений был сделан выбор в пользу Amazon Managed Streaming for Apache Kafka (MSK) — сервиса Amazon, предназначенного для развёртывания и управления службами обмена сообщениями на базе Apache Kafka. Он обеспечивает высокую отказоустойчивость и масштабируемость развёрнутого модуля, гарантируя надёжность и производительность обмена сообщениями.

Описанная архитектура представляет собой эффективное решение, обеспечивающее высокую производительность и надёжность системы. Использование среды Spring и очереди сообщений Kafka позволяет мне эффективно обрабатывать и передавать сообщения, а выбор Kubernetes в качестве оркестровки контейнеров позволяет мне масштабировать контейнеры и управлять ими с минимальными усилиями. Использование Amazon EKS, в свою очередь, обеспечивает высокую отказоустойчивость и масштабируемость развёрнутых контейнеров.

Использование Amazon MSK для развёртывания Kafka еще больше повышает надежность и производительность системы, обеспечивая максимальное удобство обмена сообщениями. Общая архитектура системы обеспечивает эффективное управление контейнерами и службами обмена сообщениями, а также высокую надёжность и производительность обмена информацией между различными службами.

A group of logos on a white background

Description automatically generated with low confidence

Рисунок 2.5 – Модуль «Управление ресурсами»

Модуль «Управление ресурсами» является ключевым внутренним компонентом, предназначенным для управления разнообразными ресурсами, такими как статьи, комментарии, наборы данных и прочее. В связи с этим данный модуль тесно взаимодействует с модулем «База данных», поскольку все управляемые ресурсы хранятся внутри модуля «База данных».

На рисунке 2.5 изображена реализация модуля с использованием фреймворка NestJs. В контексте развертывания планируется применение аналогичных технологий, как в модуле «Уведомления и подписки». Таким образом, для развертывания данного модуля будет использоваться технология Kubernetes, обеспечивающая эффективное управление контейнерами и сервисами. Для развертывания контейнеров также будет применяться Amazon EKS, позволяющий развертывать кластеры Kubernetes.

A picture containing text, font, screenshot, logo

Description automatically generated

Рисунок 2.6 – Модуль «Шифрование данных»

Модуль шифрования данных представляет собой системный компонент, обеспечивающий шифрование данных перед их сохранением в Amazon S3 — сервисе Amazon, предназначенном для хранения объектов и гиперданных, таких как изображения, видео, аудио, а также традиционных данных, включая текст. Данный модуль выполняет шифрование каждого нового набора данных, загружаемого пользователем на сервер, перед сохранением в Amazon S3 и последующей расшифровкой при извлечении данных пользователями. Схема работы модуля представлена на рисунке 2.6.

Использование данного модуля гарантирует, что в случае утечки данных, злоумышленники не смогут получить настоящие данные, так как все они будут зашифрованы. При загрузке данных обратно на сервер, модуль «Шифрование данных» производит их расшифровку, что позволяет пользователю получить фактические данные.

Описываемый модуль обеспечивает сквозное шифрование данных, что значительно повышает уровень безопасности и защиты веб-приложения даже в случае его взлома. Таким образом, использование модуля «Шифрование данных» является важным компонентом системы, который обеспечивает высокую безопасность и защиту данных при их хранении в Amazon S3. Данный модуль обеспечивает сквозное шифрование данных, что позволяет гарантировать сохранность информации в случае утечки данных или взлома системы.

A picture containing text, font, graphics, design

Description automatically generated

Рисунок 2.7 – Модуль «База данных»

Для управления данными в приложении применяется модуль «База данных». Как отображено на рисунке 2.7, в качестве SQL-базы данных выбран PostgreSQL. Основными причинами данного выбора являются высокая производительность и отказоустойчивость, предоставляемые PostgreSQL, а также его открытый исходный код.

Для размещения базы данных в качестве поставщика услуг был выбран Amazon RDS (Amazon Relational Database Server). Amazon RDS — это специализированный сервис, предоставляющий возможности хостинга для реляционных баз данных. Простота использования, масштабируемость и высокая доступность делают его идеальным выбором для размещения базы данных PostgreSQL.

A picture containing text, font, screenshot, logo

Description automatically generated

Рисунок 2.8 – Модуль «Управление идентификацией и доступом»

Кроме модуля «База данных», еще одним значимым компонентом приложения является модуль «Управление идентификацией и доступом», отвечающий за аутентификацию, авторизацию и контроль доступа. В качестве провайдера услуг выбран Amazon Cognito. Amazon Cognito представляет собой сервис Amazon, специализирующийся на управлении идентификацией и доступом, упрощающий работу с учетными записями пользователей, привилегиями и разрешениями на доступ. Общая архитектура данного модуля представлена на рисунке 2.8.

Однако использование Amazon Cognito привело к проблеме с синхронизацией данных между модулем «База данных» и модулем «Управление идентификацией и доступом». В частности, новые зарегистрированные пользователи должны храниться в обоих модулях, и любые обновления или удаления также должны отражаться в обоих модулях. Для решения этой проблемы между двумя модулями был реализован механизм синхронизации, гарантирующий согласованность и актуальность пользовательских данных во всем приложении.

Кроме того, в целях повышения безопасности для входа пользователей в систему реализован процесс двухфакторной аутентификации. После регистрации пользователи должны войти в систему с одноразовым паролём (OTP), отправленным на их мобильное устройство через SMS, в дополнение к их обычному паролю. Для облегчения этого процесса используется Amazon SNS (Amazon Simple Notification Service). Amazon SNS — это служба Amazon, используемая для отправки SMS-сообщений, обеспечивающая безопасную передачу одноразовых паролей на мобильные устройства пользователей.

## 2.3 Диаграмма компонентов

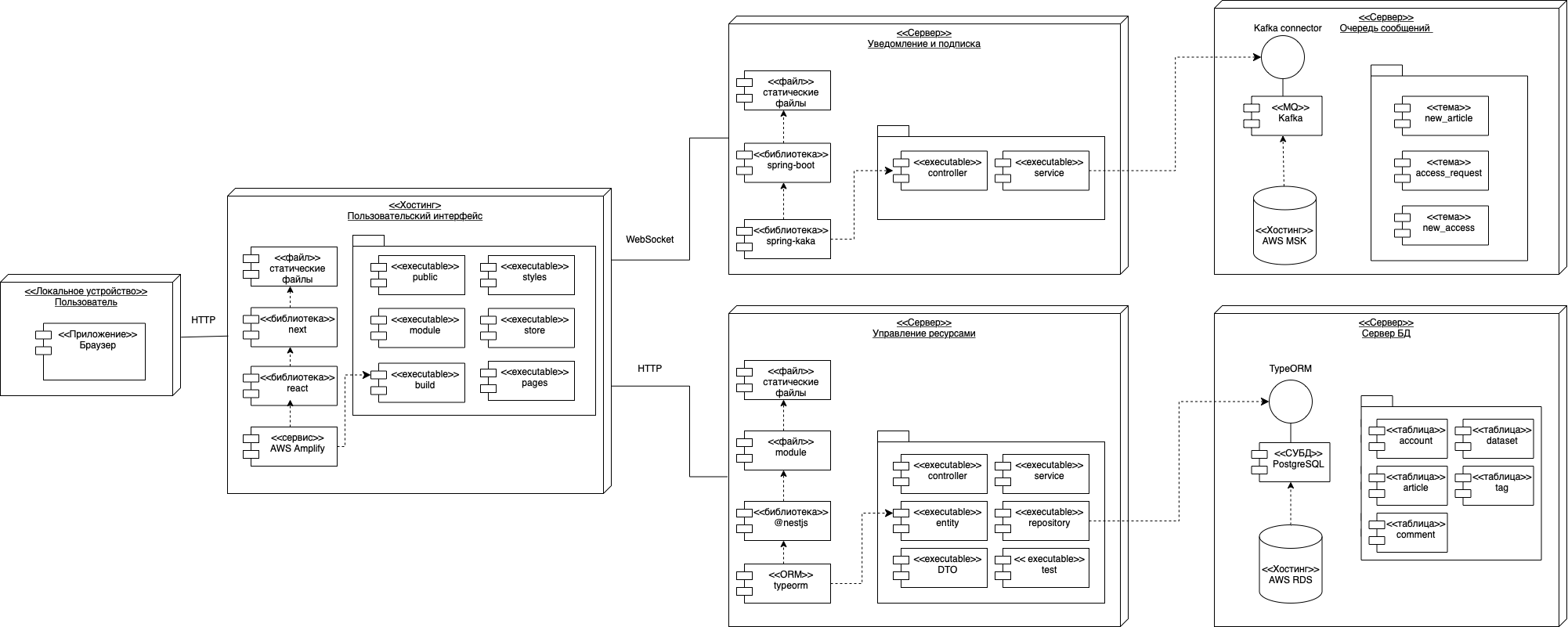


Рисунок 2.9 – Диаграмма компонентов

Согласно диаграмме компонентов на рисунке 2.9, приложение состоит из пяти отдельных компонентов, каждый из которых выполняет определенную функцию. Взаимодействие этих компонентов обеспечивает непрерывную и эффективную работу системы для конечных пользователей.

Первый компонент — это интерфейсный сервер, который служит стороной приложения, обращенной к пользователю. Он отвечает за обработку запросов пользователей и отображение информации в удобном для пользователя виде. Этот сервер взаимодействует с двумя внутренними серверами через протоколы HTTP и WebSocket, гарантируя своевременную и эффективную доставку данных на внутренние серверы.

Два внутренних сервера отвечают за обработку запросов, полученных от внешнего сервера. Эти серверы оснащены Kafka-connector, современным инструментом, который позволяет им подключаться к Kafka, распределенной системе обмена сообщениями с высокой пропускной способностью. Это гарантирует быструю и надежную обработку данных. Бэкенд-серверы также используют TypeORM, мощный инструмент объектно-реляционного сопоставления (ORM), для подключения к PostgresSQL, популярной и надежной системе управления реляционными базами данных. Это гарантирует, что данные хранятся и управляются эффективно.

Третьим компонентом системы является сервер очереди сообщений, отвечающий за управление уведомлениями. Этот сервер использует протокол WebSocket для обеспечения связи в реальном времени между серверами и очередью сообщений. Это гарантирует, что уведомления доставляются быстро и эффективно, обеспечивая бесперебойную работу для конечных пользователей.

Четвертый компонент — это сервер для базы данных, который отвечает за хранение и управление данными, используемыми приложением. Этот сервер использует PostgresSQL, известный своей надежностью, масштабируемостью и производительностью. Благодаря своим мощным функциям и возможностям PostgresSQL является идеальным выбором для управления огромными объемами данных, генерируемых приложением.

Таким образом, приложение спроектировано так, чтобы быть надежным и легко масштабируемым. Интерфейсный сервер, внутренние серверы, сервер очереди сообщений и сервер базы данных работают бесперебойно, обеспечивая бесперебойную и быструю работу приложений конечных пользователей. Используя новейшие технологии и инструменты, мое приложение хорошо адаптировано к требованиям современных вычислений благодаря своей эффективности, надежности и масштабируемости.

## 2.4 Диаграмма вариантов использования

A picture containing circle, screenshot, black and white, symmetry

Description automatically generated

Рисунок 2.10 – Диаграмма вариантов использования

Диаграмма вариантов использования на рисунке 2.10 отображает три актора: гостя, пользователя и администратора. Пользователи имеют возможность управлять своими статьями и наборами данных, в то время как администраторы обладают расширенными функциями управления статьями, наборами данных, тегами и другими пользователями. Таким образом, администраторы имеют больший контроль и доступ к функциональности приложения по сравнению с обычными пользователями.

## 2.5 Диаграмма классов анализа

A black background with white dots

Description automatically generated with low confidence

Рисунок 2.11 – Диаграмма классов анализа

На диаграмме классов анализа, представленной на рисунке 2.11, система состоит из 9 интерфейсов и 6 объектов, включая наборы данных, статьи, комментарии, теги, пользователей и статистику. Также в системе присутствует специальный объект — метаданные набора данных, действующий как промежуточный уровень между интерфейсом набора данных и объектом набора данных, управляемый интерфейсом набора данных.

Система предоставляет разнообразные типы интерфейсов, такие как интерфейсы статей, комментариев и наборов данных, доступные для всех категорий пользователей. Однако некоторые интерфейсы, например, создания статей и управления наборами данных, требуют авторизации и предоставляются только авторизованным пользователям. Администраторы имеют доступ к трём административным интерфейсам: управление тегами, управление пользователями и статистические интерфейсы.

# **3 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ**

## 3.1 Этап разработки

Чтобы создать пользовательский интерфейс, я использовал NextJs в качестве фреймворка и MaterialUI в качестве основной библиотеки, придерживаясь рекомендаций Google по взаимодействию с пользователем. Поскольку использовался **TypeScript**, необходимо было настроить TypeScript и использовать модуль **@types**.

Модуль был разделен на пять отдельных подмодулей, каждый из которых имеет свои уникальные функции и компоненты. Подмодуль *core* состоит из основных функций и основных компонентов, включая панель навигации, верхний и нижний колонтитулы, которые будут использоваться на всех страницах приложения. Подмодуль *auth*, подключенный к Amazon Cognito, включает такие интерфейсы, как вход в систему, регистрация и т. д. Подмодуль *account* содержит такие страницы, как сведения о пользователе. Подмодуль *article* состоял из исчерпывающего контента блога, списка блогов и т. д. Подмодуль *home* включал основные страницы, а также другие страницы, такие как страница 404, когда страница не может быть найдена, и страница документации.

Чтобы исходный код был легко читаемым и расширяемым, использовались **Eslint** и **Prettier**. Благодаря использованию хуков *pre-commit* и **Husky** автоматически оценивался перед каждой фиксацией.

Далее представлены интерфейсы различных страниц: главная страница (рисунок 3.1), страница блогов (рисунок 3.2), страница блога (рисунок 3.3), страница входа (рисунок 3.4) и страница регистрации (рисунок 3.5).

A screenshot of a website

Description automatically generated with low confidence

Рисунок 3.1 – Интерфейс главной страницы

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.2 – Интерфейс страницы блогов

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.3 – Интерфейс страницы блога

A screenshot of a computer screen

Description automatically generated with low confidence

Рисунок 3.4 – Интерфейс входа

A screenshot of a login form

Description automatically generated with medium confidence

Рисунок 3.5 – Интерфейс регистрации

Модуль «Управление ресурсами» был реализован с использованием NestJs с Typescript и разделен на шесть подмодулей. Подмодуль *account* был разработан для управления объектами Account, которые хранятся в таблице *account* в PostgreSQL. Точно так же подмодули *article*, *comment*, *dataset* и *tag* используются для управления первичными объектами в соответствующих таблицах. Кроме того, имеется специальный подмодуль *auth*, отвечающий за синхронизацию и подключение к сервисам Amazon Cognito, а также за управление аутентификацией, авторизацией и контролем доступа.

Каждый подмодуль имеет согласованный формат файла, включая файл «<имя модуля>.service.ts», отвечающий за управление логикой приложения. Файл «<имя модуля>.controller.ts» использует логику «<имя модуля>.service.ts» для соответствия запросам API. Файл «<имя модуля>.entity.ts» соединяет и передает объект из кода Typescript в SQL, используя **TypeORM** в качестве инструмента направленного рационального сопоставления. Файл «<имя модуля>.repository.ts» создает уровень абстракции между уровнем доступа к данным и уровнем бизнес-логики приложения, действующий как набор сущностей, взятых из базы данных (и преобразованных из «<имя модуля>.entity .тс»). Кроме того, у каждого подмодуля есть специальная папка с именем «dto», которая представляет собой объект передачи данных, используемый для преобразования объекта в стандартный формат при вводе или выводе, что облегчает разработку других модулей. Наконец, все эти файлы объединены в файл «<имя модуля>.module.ts».

В следующем разделе представлен документ API для модуля «Управление ресурсами», сформированный с использованием инструмента Swagger (рисунок 3.6).

A screenshot of a computer

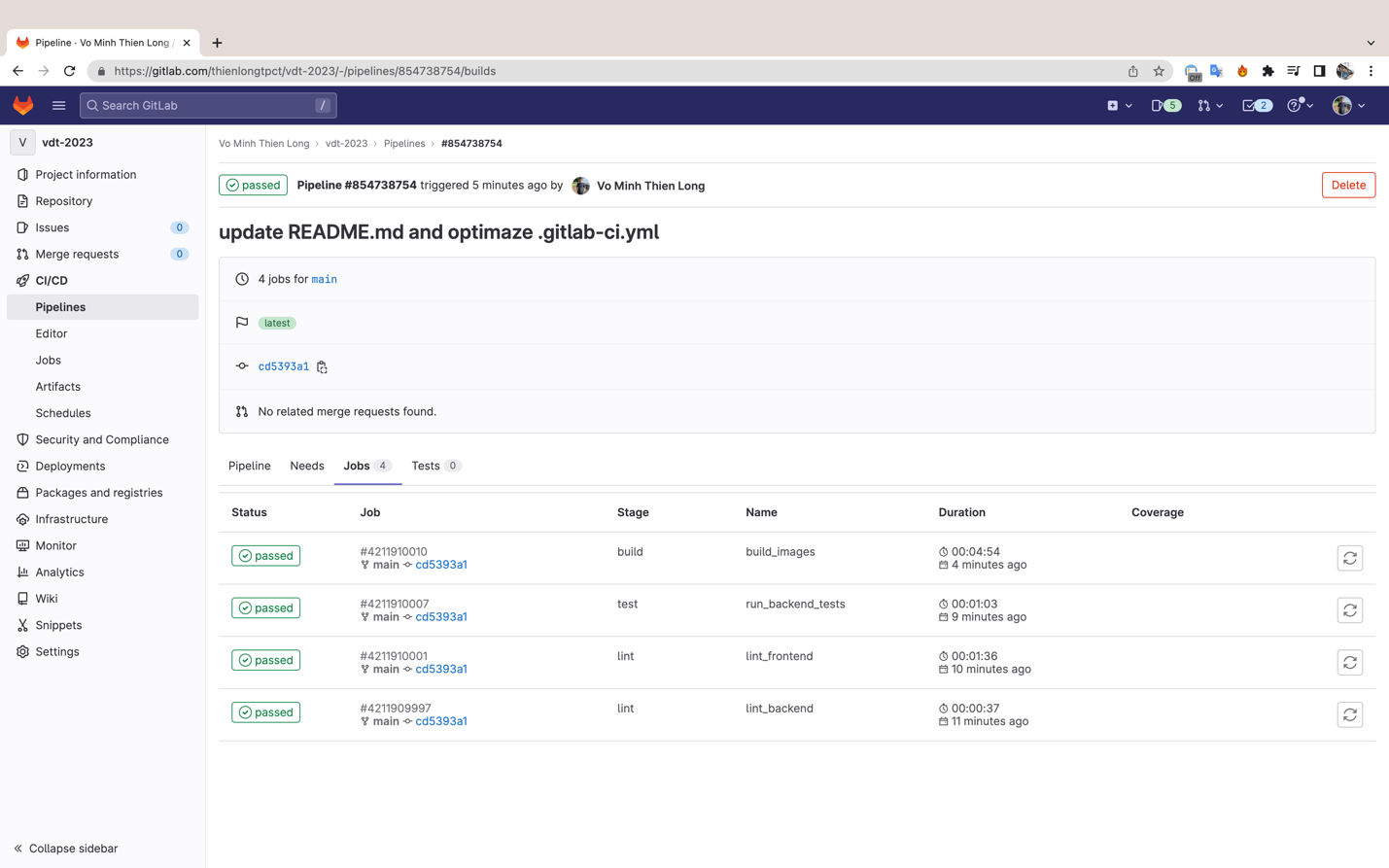
Description automatically generated

Рисунок 3.6 – API-документация

Модуль «Уведомление и подписка» будет реализован с использованием Spring Boot с подключаемым модулем spring-kafka, который будет связан с Apache Kafka. Протокол WebSocket будет использоваться для связи между этим модулем, модулем «Пользовательский интерфейс» и очередью сообщений Kafka. Такой подход обеспечит взаимодействие с пользователем в режиме реального времени, что делает его оптимальным решением для требований проекта.

## 3.2 Этап CI/CD

Проект состоит из четырех отдельных задач, представленных на рисунке 3.7: две задачи относятся к фазе анализа, одна задача связана с фазой тестирования, а последняя задача относится к фазе сборки. Первое задание, «lint\_backend», используется для анализа исходного кода внутреннего модуля. Второе задание, «lint\_frontend», также отнесенное к категории lint, выполняет проверку исходного кода внешнего модуля. Поскольку «lint\_backend» и «lint\_frontend» не зависят друг от друга, их можно выполнять одновременно. Третье задание, «run\_backend\_tests», отвечает за тестирование внутреннего модуля. Это задание может начаться только после завершения обоих заданий на этапе *lint* и должно быть завершено до начала этапа *build*. Последнее задание, «build\_images», отвечает за создание двух образов, с использованием соответствующего исходного кода из соответствующих модулей. В этом задании используются различные переменные, в том числе Docker TLS, а также имена и теги для образов.

 Рисунок 3.7 – Четыре задания в потоке CI/CD

Кроме того, будут использоваться два учётных данных реестра, а именно «REGISTRY\_USER» и «REGISTRY\_PASS». Эти переменные содержат конфиденциальную информацию и поэтому не могут быть определены непосредственно в этом задании. Чтобы гарантировать конфиденциальность наших переменных, мы будем использовать раздел **Variables**, расположенный в **Settings > CI/CD**. Кроме того, мы обеспечим включение параметров *Protected* и *Masked*, чтобы предотвратить несанкционированный доступ и защитить конфиденциальную информацию, содержащуюся в этих переменных.

В последующих разделах приводятся данные об этапе CI/CD, в том числе сведения о переменных и секретах (рисунок 3.8), а также описание фаз (рисунок 3.9).

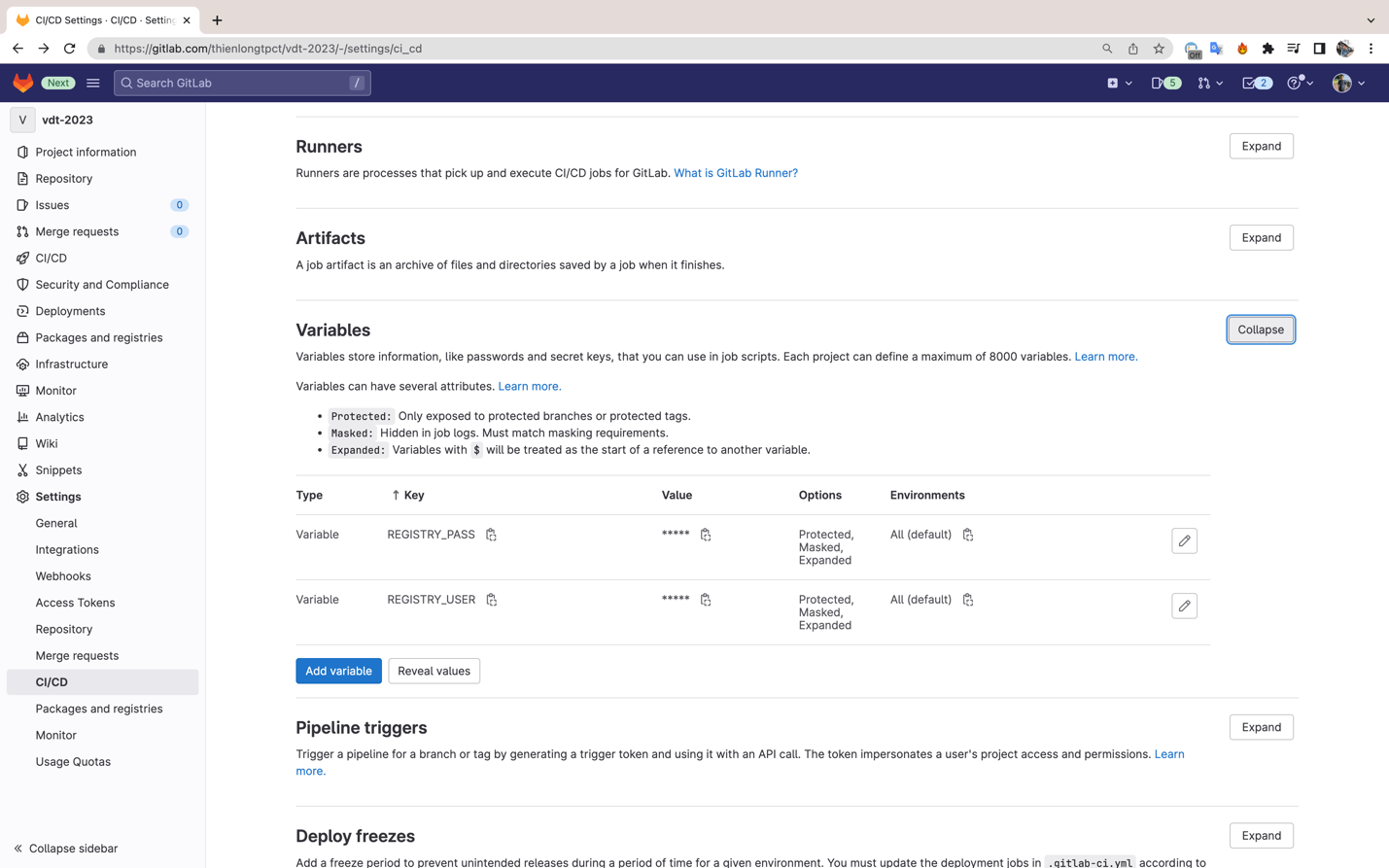


Рисунок 3.8 – Подробная информация о переменных и секретах в конвейере

A screenshot of a computer

Description automatically generated

Рисунок 3.9 – Подробная информация об этапах в конвейере

## 3.3 Этап развёртывания

Одна из первоначальных целей, которую необходимо выполнить перед развертыванием приложения, включает развертывание сервера PostgreSQL с использованием Amazon RDS. Этот процесс включает в себя несколько важных шагов, включая выбор соответствующего ядра базы данных, класса экземпляра и размера хранилища. Кроме того, необходимо настроить параметры базы данных, такие как имя базы данных, имя пользователя и пароль, а также параметры группы безопасности.

В рамках этого проекта я успешно установил базу данных PostgreSQL с использованием Aurora PostgreSQL на Amazon RDS. Это достижение обеспечивает надежное и масштабируемое решение для управления базами данных, которое имеет основополагающее значение для эффективного функционирования различных приложений. Успешное развертывание сервера PostgreSQL с использованием Aurora PostgreSQL на Amazon RDS гарантирует, что приложение получит доступ к надежной и безопасной базе данных, что необходимо для хранения данных и управления ими.

Во-первых, я создал базу данных и выбрал «Amazon Aurora» в качестве механизма базы данных, выбрав «Amazon Aurora PostgreSQL» в качестве редакции. Во-вторых, я выбрал соответствующую версию базы данных, класс экземпляра и размер хранилища. В частности, для этого проекта я выбрал Engine 15.6, поскольку он представляет собой последнюю поддерживаемую версию PostgreSQL в Amazon Web Services. Я также выбрал шаблон развертывания *production* со **100 ГБ** хранилища и конфигурацией кластерного хранилища Aurora Standard. В-третьих, я настроил параметры базы данных, включая имя базы данных, имя пользователя и пароль, а также выбрал частное облако (VPC) и группу подсети данных. Кроме того, я выбрал группы безопасности для базы данных, гарантируя создание новых групп безопасности, которые разрешают доступ только к порту **5432**, порту по умолчанию для PostgreSQL. Наконец, я настроил параметры резервного копирования и обслуживания базы данных, включая создание двух репликаций для обеспечения высокой доступности базы данных.

В рамках работы с сервисом Amazon RDS была создана база данных под названием "*vkr*" с двумя экземплярами. Первый экземпляр (*vkr-instance-1*) предназначен для выполнения операций записи, второй экземпляр (*vkr-instance-1-us-west-1b*) используется для операций чтения. Подробная информация о структуре БД представлена на рисунке 3.10.

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.10 – Список баз данных Amazon RDS, использованных в выпускной работе

В рамках стандартных функций Amazon поддерживает мониторинг статистики базы данных с помощью CloudWatch. Эта функция предоставляет многочисленные полезные статистические данные, такие как количество подключений и задержка каждой фиксации, которые можно эффективно использовать для управления базами данных и целей оптимизации.

Дополнительная информация о базе данных, такая как конечные точки, характеристики производительности, количество установленных подключений и прочее, представлена на рисунках 3.11 и 3.12.

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.11 – Подробная информация базы данных "vkr"

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.12 – База данных "vkr" статистическая информация

Следующая задача в этом проекте включает настройку Amazon Cognito для включения проверки подлинности, авторизации и контроля доступа. Этот процесс включает в себя создание пула пользователей и клиентского приложения в нем, что позволит приложению подключаться к Amazon Cognito.

При создании пула пользователей я создал и определил различные атрибуты пользователей, настроил многофакторную аутентификацию и настроил политики паролей для повышения безопасности. Параллельно я также создал клиентское приложение, которое позволяет приложению взаимодействовать с Amazon Cognito и аутентифицировать пользователей. Использование идентификатора клиентского приложения обеспечивает безопасное соединение между приложением и Amazon Cognito, обеспечивая надёжное управление аутентификацией и авторизацией пользователей.

На рисунке 3.13, демонстрируется конфигурация Amazon Cognito, примененная в рамках данной работы.

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.13 – Интерфейс конфигурации Amazon Cognito

## 3.4 Этап мониторинга

Для эффективного мониторинга приложения я реализовал инструменты Prometheus и Grafana. Prometheus используется для сбора метрик из приложения, получаемых через экспортеров, такие как Node Exporter, MongoDB Exporter и другие. Затем эти метрики визуализируются с помощью Grafana, что обеспечивает полное представление о производительности приложения и позволяет выявить любые потенциальные проблемы.

Кроме того, для ведения журналов я подключил Fluentd, который отвечает за сбор и отправку журналов в Elasticsearch. Затем платформа Elasticsearch используется для хранения и анализа журналов, собранных Fluentd. Наконец, Kibana используется для визуализации журналов, что позволяет выявлять ошибки и отслеживать любые потенциальные проблемы.

На рисунках 3.14 и 3.15 представлен пример выполнения запроса с использованием системы мониторинга Prometheus в контексте данной работы.

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.14 – Интерфейс запросов в Prometheus

A screen shot of a graph

Description automatically generated with medium confidence

Рисунок 3.15 – Графический интерфейс в Prometheus

На рисунке 3.15 демонстрируется интерфейс системы мониторинга, разработанный с применением инструмента Grafana и Prometheus в рамках выпускной квалификационной работы.A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.15 – Интерфейс статистического мониторинга

## 3.5 Тестирование

Для обеспечения оптимальной работы приложения и соответствия международным стандартам, были внедрены строгие процедуры тестирования. Процесс тестирования включал оценку производительности приложения на основе различных критериев, таких как базовый стандарт Google Web Vitals для пользовательского интерфейса и стандарт Якоба Нильсена для времени отклика серверной части.

Пользовательский интерфейс приложения оценивался с учетом основных стандартов Google Web Vitals, фокусирующихся на таких показателях, как скорость загрузки, интерактивность и визуальная стабильность. Оценка данных показателей обеспечивала удобство использования приложения, его отзывчивость и беспрепятственный пользовательский опыт.

Время отклика серверной части приложения оценивалось согласно стандарту Якоба Нильсена, который акцентирует внимание на времени отклика как критическом факторе, определяющем восприятие пользователем скорости и надежности приложения. Оценка времени отклика приложения проводилась с целью соответствия стандарту и обеспечения быстрого и надежного взаимодействия с пользователем.

Дополнительно к указанным стандартам были реализованы другие процедуры тестирования, включая модульное тестирование и тестирование скорости. Эти процедуры тестирования направлены на выявление и устранение проблем, связанных с функциональностью, производительностью и безопасностью приложения. В таблице 3.1 представлены проведенные тесты, ожидаемые результаты и фактически полученные результаты.

Таблица 3.1 – Таблица тестирования

A picture containing text, screenshot, number, line

Description automatically generated

# **ЗАКЛЮЧЕНИЕ**

В заключение, этот проект включал в себя несколько технических задач, направленных на создание общедоступного веб-сайта, ориентированного на сообщество исследователей моделей машинного обучения. Проект начался с обзора предметной области и анализа аналогов, после чего последовало проектирование высокоуровневой архитектуры веб-приложения.

Выбор подходящих Amazon Web Services имел решающее значение при реализации веб-приложения, при этом PostgreSQL и Redis использовались в качестве уровня базы данных и уровня кэширования, соответственно, для повышения производительности приложения. Использование Spring (Java), Flask (Python) и NestJs (JavaScript) для дельта-сервисов на логическом уровне и NextJs (JavaScript) для создания удобного интерфейса обеспечило надежность и удобство приложения.

Тестирование веб-приложения и исправление любых ошибок имели решающее значение для обеспечения того, чтобы приложение было надежным и готовым обслуживать большое количество пользователей. Наконец, развертывание и публикация программного продукта в открытом Интернете с использованием Amazon Web Services гарантировали, что приложение будет доступно для целевой аудитории.

В целом, этот проект был успешным в создании общедоступного веб-сайта, отвечающего потребностям сообщества исследователей моделей машинного обучения. Использование облачных технологий наряду с реализацией соответствующих мер безопасности обеспечило безопасность данных, а также позволило исследователям эффективно обмениваться наборами данных, учебными моделями и другими статьями знаний.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. AI and machine learning are everywhere [Электронный ресурс]. – URL: https://inforummichigan.org/artificial-intelligence-and-machine-learning-are-everywhere/ (дата обращения: 12.04.2023).
2. How TikTok’s algorithm made it a success: "It pushes the boundaries" [Электронный ресурс]. – URL: https://www.theguardian.com/technology/2022/oct/23/tiktok-rise-algorithm-popularity (дата обращения: 14.04.2023).
3. K. Shailaja, B. Seetharamulu and M. A. Jabbar, "Machine learning in healthcare: a review", Second International Conference on Electronics Communication and Aerospace Technology (ICECA), pp. 910-914, 2018.
4. Xuesong Zhai, Xiaoyan Chu, Ching Sing Chai, Morris Siu Yung Jong, Andreja Istenic, Michael Spector, Jia-Bao Liu, Jing Yuan, and Yan L, "A Review of Artificial Intelligence (AI) in Education from 2010 to 2020", Hindawi, pp.1-18, 2021.
5. F. G. D. C. Ferreira, A. H. Gandomi and R. T. N. Cardoso, "Artificial intelligence applied to stock market trading: a review", IEEE Access, vol. 9, pp. 30898-30917, 2021.
6. Lakshmi Shankar Iyer, "AI enabled applications towards intelligent transportation", Transportation Engineering, vol. 5, 2021.
7. Amount of Data Created Daily (2023) [Электронный ресурс]. – URL: https://explodingtopics.com/blog/data-generated-per-day (дата обращения: 14.04.2023).
8. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025) [Электронный ресурс]. – URL: https://www.statista.com/statistics/871513/worldwide-data-created/ (дата обращения: 14.04.2023).
9. Kaggle [Электронный ресурс]. – URL: https://www.kaggle.com/ (дата обращения: 16.04.2023).
10. ModelZoo [Электронный ресурс]. – URL: https://modelzoo.co/about (дата обращения: 17.04.2023).
11. Towards Data Science [Электронный ресурс]. – URL: https://towardsdatascience.com/ (дата обращения: 17.04.2023).
12. Edwards-Curve Digital Signature Algorithm (EdDSA) [Электронный ресурс]. – URL: https://www.rfc-editor.org/rfc/rfc8032.html (дата обращения: 26.04.2023).
13. Core Web Vitals report [Электронный ресурс]. – URL: https://support.google.com/webmasters/answer/9205520?hl=en (дата обращения: 29.05.2023).
14. Response Times: The 3 Important Limits [Электронный ресурс]. – URL: https://www.nngroup.com/articles/response-times-3-important-limits/ (дата обращения: 29.05.2023).