Московский авиационный институт
(национальный исследовательский университет)
Институт: №8 «Компьютерные науки и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Курсовой проект
по курсу «Нейроинформатика» на тему:

«Прогнозирование одобрения кредитной карты клиенту банка»

Работу выполнили студенты группы М8О-406Б-21
Беспалов В.А., Чистяков К.С., Шляхтина И.И.
Преподаватель: Мазаев А.С.

Описание

Прогнозирование одобрения кредитной карты клиенту банка на основании данных о клиенте (возраст, семейное положение, текущая задолженность по кредитам, отрасль в которой работает и тд).

```
example = {'Gender': [1], # Пол, 0= Женский, 1=Мужской
 'Age': [20.67], # Возраст в годах
 'Debt': [5.29], # Непогашенный долг (функция была масштабирована)
 'Married': [1], # Женат, 0 = холост/разведён/и т. д., 1 = женат
 'BankCustomer': [1], # Клиент банка, 0 = нет банковского счёта, 1 = есть банковский счёт
 'Industry': [9], # Отрасль – сектор занятости текущего или самого последнего места работы
 'Ethnicity': [4], # Этническая принадлежность
 'YearsEmployed': [0.375], # Отработанные годы
 'PriorDefault': [1], # Предыдущее значение по умолчанию, 0= никаких предыдущих значений по умолчанию, 1= предыдущее значение по умолчанию
 'Employed': [1], # Занятый, 0= не занятый, 1=занятый
 'CreditScore': [1], # Кредитный рейтинг (эта функция была масштабирована)
 'DriversLicense': [0], # Водительские права, 0 = нет лицензии, 1 = имеет лицензию
 'Citizen': [0], # Гражданство, полученное при рождении, иным способом или Временно
 'ZipCode': [160], # Почтовый индекс (5-значный номер)
 'Income': [0]} # Доход (эта функция была масштабирована)
                # 'Approved': Утверждено, 0= не утверждено, 1= утверждено

example_df = pd.DataFrame(example)
model.predict(example_df)
```

```
array([0])
```

Бизнес-цель проекта

1. Увеличить процент одобрения кредитных карт для клиентов, соответствующих критериям кредитоспособности, используя модель машинного обучения для анализа данных о клиентах и предсказания вероятности одобрения.

2. Сократить выдачу кредитных карт неплатежеспособным заемщикам.

ML-цель проекта

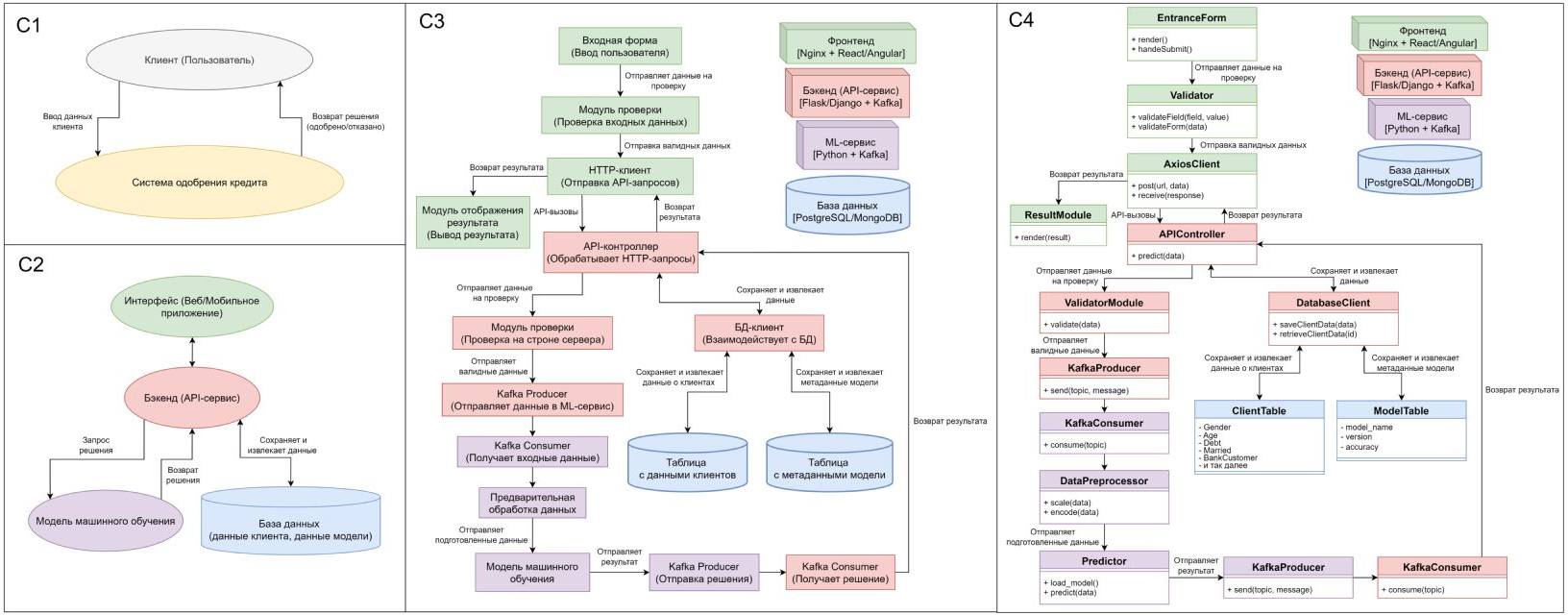
Проект направлен на решение задачи бинарной классификации, где модель машинного обучения предсказывает, относится ли клиент к категории: «Кредит одобрен» (1) или «Кредит не одобрен» (0).

Задача классификации используется для определения вероятности того, что клиент соответствует критериям кредитоспособности, основываясь на его входных характеристиках (возраст, задолженность, семейное положение и т.д.).

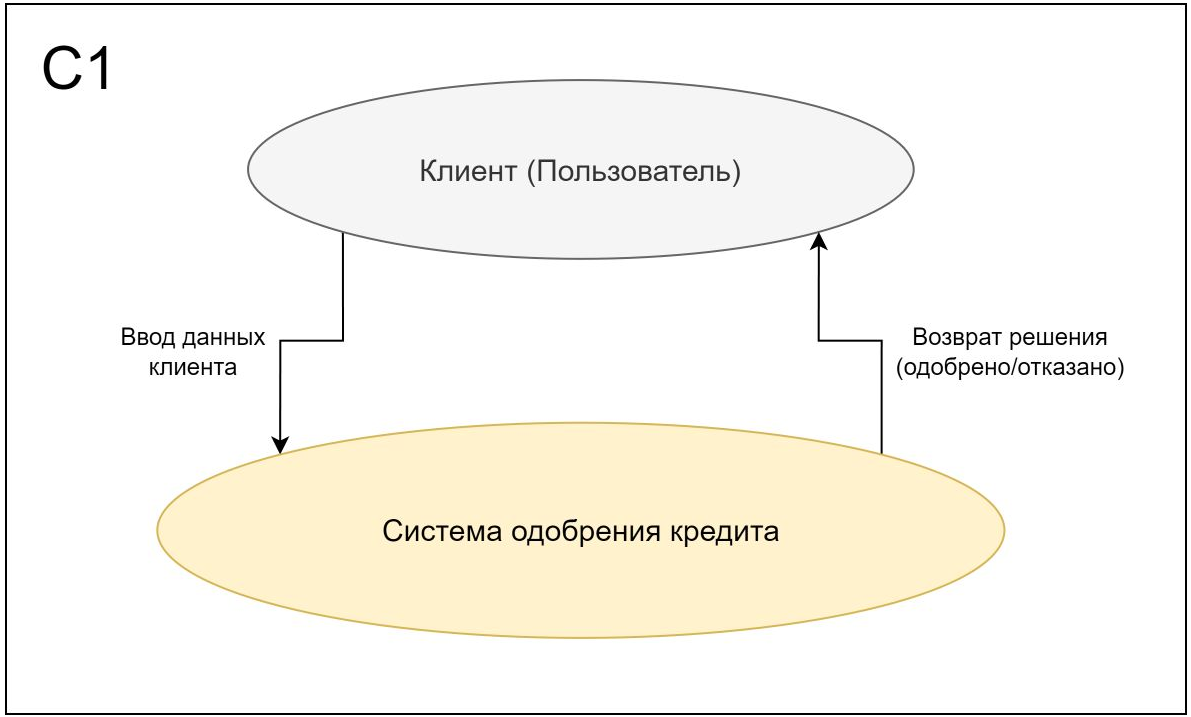
Целью классификации в данном случае является:

1. Максимизация точности предсказания одобрения кредитных карт для подходящих клиентов.
2. Минимизация ошибок (ложных положительных и ложных отрицательных), чтобы сократить риск выдачи кредитов неплатежеспособным клиентам.

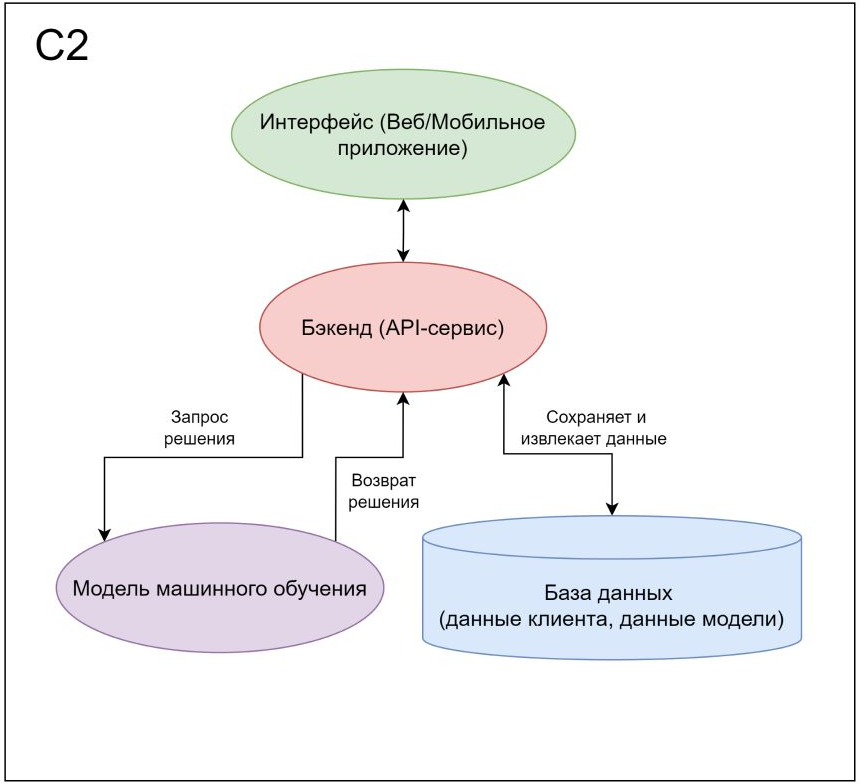
Архитектура (схема C4)



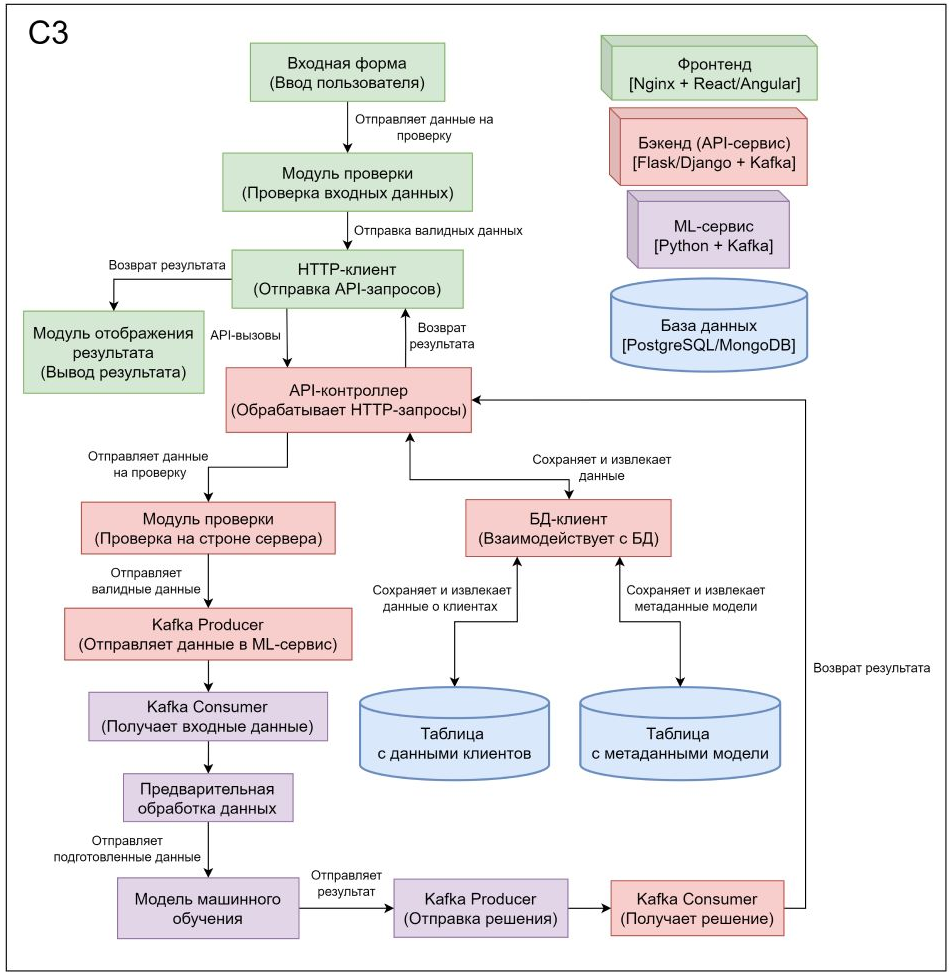
C1


C1
Клиент (Пользователь)
Ввод данных клиента
Возврат решения (одобрено/отказано)
Система одобрения кредита

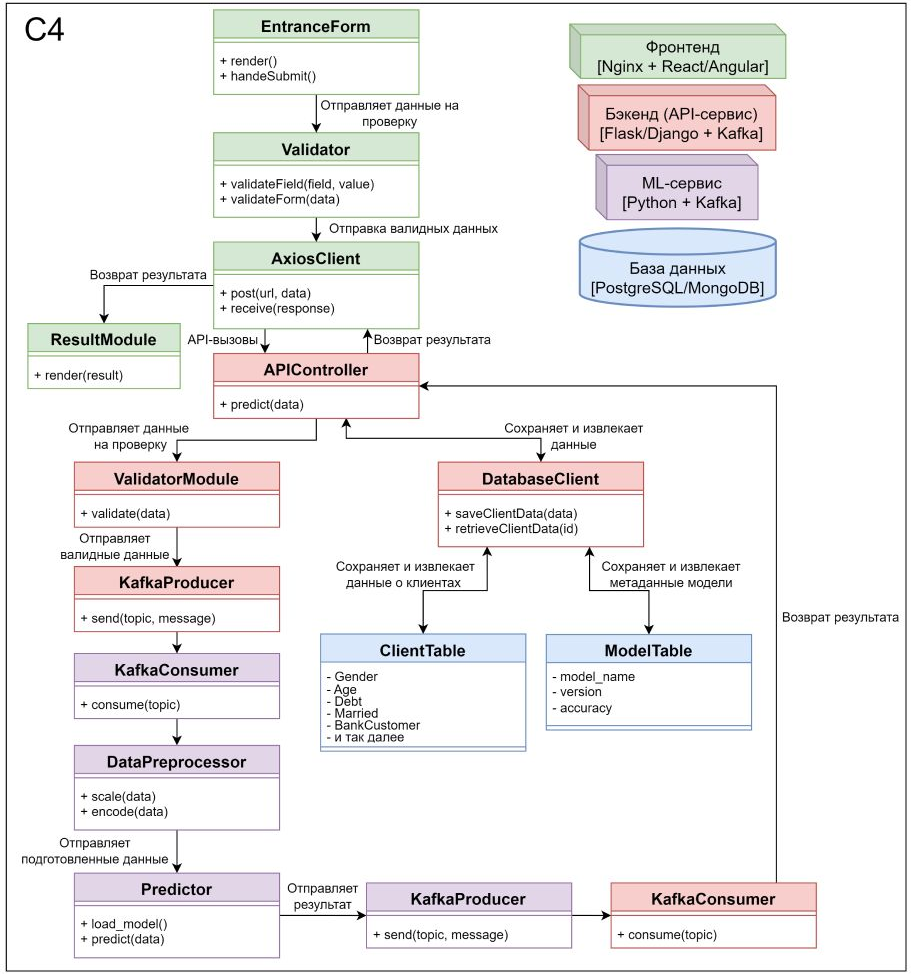
C2


C2
Интерфейс (Веб/Мобильное приложение)
Бэкенд (API-сервис)
Запрос решения
Возврат решения
Сохраняет и извлекает данные
Модель машинного обучения
База данных
(данные клиента, данные модели)

С3


С3
Входная форма
(Ввод пользователя)
Отправляет данные на проверку
Модуль проверки
(Проверка входных данных)
Отправка валидных данных
Возврат результата
HTTP-клиент
(Отправка API-запросов)
Модуль отображения результата
(Вывод результата)
API-вызовы
Возврат результата
API-контроллер
(Обрабатывает HTTP-запросы)
Отправляет данные на проверку
Сохраняет и извлекает данные
Модуль проверки
(Проверка на строне сервера)
БД-клиент
(Взаимодействует с БД)
Отправляет валидные данные
Сохраняет и извлекает данные о клиентах
Сохраняет и извлекает метаданные модели
Kafka Producer
(Отправляет данные в ML-сервис)
Kafka Consumer
(Получает входные данные)
Таблица
с данными клиентов
Таблица
с метаданными модели
Предварительная обработка данных
Отправляет подготовленные данные
Модель машинного обучения
Отправляет результат
Kafka Producer
(Отправка решения)
Kafka Consumer
(Получает решение)
Возврат результата
Фронтенд
[Nginx + React/Angular]
Бэкенд (API-сервис)
[Flask/Django + Kafka]
ML-сервис
[Python + Kafka]
База данных
[PostgreSQL/MongoDB]

C4


C4
EntranceForm
+ render()
+ handeSubmit()
Отправляет данные на проверку
Validator
+ validateField(field, value)
+ validateForm(data)
Отправка валидных данных
AxiosClient
+ post(url, data)
+ receive(response)
Возврат результата
ResultModule
+ render(result)
API-вызовы
Возврат результата
APIController
+ predict(data)
Отправляет данные на проверку
Сохраняет и извлекает данные
ValidatorModule
+ validate(data)
DatabaseClient
+ saveClientData(data)
+ retrieveClientData(id)
Отправляет валидные данные
KafkaProducer
+ send(topic, message)
Сохраняет и извлекает данные о клиентах
Сохраняет и извлекает метаданные модели
ClientTable
- Gender
- Age
- Debt
- Married
- BankCustomer
- и так далее
ModelTable
- model_name
- version
- accuracy
KafkaConsumer
+ consume(topic)
DataPreprocessor
+ scale(data)
+ encode(data)
Возврат результата
Отправляет подготовленные данные
Predictor
+ load_model()
+ predict(data)
Отправляет результат
KafkaProducer
+ send(topic, message)
KafkaConsumer
+ consume(topic)
Фронтенд [Nginx + React/Angular]
Бэкенд (API-сервис) [Flask/Django + Kafka]
ML-сервис [Python + Kafka]
База данных [PostgreSQL/MongoDB]

Обоснование архитектуры

Архитектура проекта основана на клиент-серверной модели. Модель машинного обучения разрабатывается и обучается на языке программирования Python с использованием библиотеки Scikit-learn, средства которой позволяют быстро разработать модель. Для предсказания одобрения кредита выбрана модель случайного леса, как она хорошо справляется с несбалансированными данными и собеспечивает высокую точность результатов.

Данные клиента обрабатываются в три этапа: сбор, предобработка (масштабирование и кодирование категориальных признаков) и хранение в PostgreSQL. Такой выбор обусловлен необходимостью долговременного хранения и возможности выполнения сложных аналитических запросов. REST API реализован с помощью Flask, что позволяет легко интегрировать модель в какую-либо существующую инфраструктуру.

Уровень C1

На этом уровне описывается взаимодействие пользователя (клиента) с системой одобрения кредитов:

Клиент (пользователь) вводит данные через интерфейс (например, веб-приложение или мобильное приложение). Система одобрения кредита обрабатывает эти данные и возвращает решение: одобрить кредит или отказать.

Уровень C2

Этот уровень показывает разбиение системы на ключевые компоненты:

Интерфейс (веб или мобильное приложение) собирает данные от клиента и направляет их в бэкенд. Интерфейс отвечает за удобство пользователя, позволяя легко вводить данные.

Бэкенд (API-сервис) выполняет обработку данных, взаимодействует с базой данных и вызывает модель машинного обучения. Бэкенд обеспечивает централизованную обработку запросов, что делает систему масштабируемой и удобной для интеграции.

Модель машинного обучения принимает входные данные, делает предсказание (0 или 1) и возвращает результат. Модель ML встроена как отдельный компонент, что позволяет легко обновлять или заменять её в будущем.

База данных хранит данные клиентов и метаинформацию о модели. База данных необходима для хранения информации о клиентах и модели, что обеспечивает устойчивость и повторяемость процесса.

Уровень С3

Этот уровень детализирует взаимодействие между компонентами:

Frontend (интерфейс) вводит данные через форму (например, с использованием React/Angular) и передаёт данные через API-запросы (AxiosClient).

API-контроллер (бэкенд) получает запросы, проверяет данные (модуль валидации), отправляет валидные данные через Kafka в ML-сервис.

ML-сервис выполняет предобработку данных (DataPreprocessor), делает предсказание (Predictor) и отправляет результат обратно.
База данных хранит клиентские данные (ClientTable) и метаинформацию о модели (ModelTable).

Использование Kafka для передачи данных между бэкендом и ML-сервисом делает систему асинхронной и устойчивой к нагрузкам. Модульная структура (валидация, предобработка, предсказание) упрощает поддержку и масштабирование. Хранение метаданных модели в отдельной таблице позволяет отслеживать версию модели и её точность.

Объяснение выбора стека

1. **Фронтенд — ввод и отображение данных**

Фреймворки React или Angular — для создания динамичного пользовательского интерфейса, обеспечивающего интерактивность и отзывчивость приложения.

Библиотека Axios — для отправки HTTP-запросов (GET/POST) с клиентской стороны на сервер.

Веб-сервер Nginx — для хостинга фронтенд-приложения и обработки статических файлов.

Объяснение выбора стека

2. **Бэкенд — обработка данных и взаимодействие с другими сервисами.**

Язык программирования Python используется благодаря своей простоте и большому количеству библиотек для интеграции с ML-моделями и работы с данными.

Фреймворк для создания REST API — Flask, лёгкий и масштабируемый фреймворк для обработки HTTP-запросов и построения API.

Асинхронные сообщения — Kafka (Apache Kafka) для передачи данных между компонентами (например, от API к ML-сервису). Kafka обеспечивает надёжную асинхронную коммуникацию и масштабируемость.

Объяснение выбора стека

3. Машинное обучение (ML-сервис)

Язык программирования: Python — основной язык для разработки ML-сервиса. Scikit-learn: библиотека для разработки, обучения и тестирования моделей машинного обучения. В данном проекте используется модель случайного леса (Random Forest).

Обработка данных: Pandas для работы с табличными данными, NumPy для вычислений и работы с числовыми данными, Kafka (в ML-сервисе) для приёма входных данных и отправки предсказаний обратно в API.

Объяснение выбора стека

4. **База данных — хранение данных клиентов и метаинформации**

База данных: PostgreSQL — реляционная база данных для хранения клиентских данных (таблица ClientTable) и метаинформации о модели (таблица ModelTable). MongoDB (опционально) может использоваться для неструктурированных данных, если проект предполагает хранение дополнительной информации (например, логов).

Объяснение выбора стека

5. Системные компоненты

Серверные технологии:

Nginx — веб-сервер для маршрутизации запросов и балансировки нагрузки.
Docker — для контейнеризации всех компонентов проекта (фронтенд, бэкенд, ML-сервис, Kafka, база данных). Это упрощает развертывание и переносимость системы.
Kubernetes (опционально) — для оркестрации контейнеров в случае масштабирования системы.

6. Дополнительные технологии

Контроль версий:

Git для управления версиями кода.
GitHub/GitLab для хостинга репозиториев и управления проектом.

Мониторинг и логирование:

Prometheus/Grafana для мониторинга производительности системы.
ELK-стек (Elasticsearch, Logstash, Kibana) для хранения и анализа логов.

Описание сетей

1. **DMZ (Demilitarized Zone)** предназначена для минимизации риска атаки на внутреннюю сеть. Здесь располагается Frontend (Nginx/React/Angular) и API шлюз. Содержит компоненты, которые должны быть доступны из интернета.

2. **Secure Zone (зона безопасности)** предназначена для обработки данных, которые уже прошли первичную валидацию. Здесь располагаются Backend API и Validation Logic. Также здесь находится компонент, отправляющий данные в Kafka.

3. **Internal Zone (внутренняя зона)** предназначена для компонентов, обеспечивающих хранение данных и выполнение машинного обучения. Здесь находятся: база данных (PostgreSQL/MongoDB), Kafka Cluster, ML Service.

4. **Monitoring & Management Zone (зона мониторинга)** — изолированная зона для управления и мониторинга всей инфраструктуры. Размещает инструменты мониторинга, такие как Prometheus, Grafana, а также Kafka UI или Kibana для анализа логов.