ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет информатики, математики и компьютерных наук

Программа подготовки бакалавров по направлению
09.03.04 Программная инженерия

Титова Тамара Сергеевна

КУРСОВАЯ РАБОТА

Разработка расширения для Obsidian. Анализ аудитории на основе ML алгоритмов

Научный руководитель
старший преподаватель НИУ
ВШЭ - НН

Саратовцев Артем
Романович

Нижний Новгород, 2025г.

Оглавление

Введение

На сегодняшний день, IT технологии, несомненно, стремительно развиваются. Мы пользуемся Интернетом каждый день, используем приложения, для более эффективной работы или облегчения нашей жизни. Но несмотря на то, что программное обеспечение постоянно обновляется и развивается, не найдётся такого, где были бы всё нужные нам инструменты. Для выяснения необходимых инструментов, очевидно, нужно проводить исследование аудитории. Это можно сделать разными методами, например, через сбор статистики или проведение опросов. В рамках моей курсовой работы мне необходимо будет провести анализ аудитории вспомогательного программного обеспечения Obsidian с целью изучения использования расширения для выполнения SQL-запросов. Самый подходящий способ анализировать поведение пользователей в приложении это использовать машинные алгоритмы обучения, которые предсказывают дальнейшие действия, отслеживают статистику использования и ее тип.

Для анализа пользователей, которые используют расширение Obsidian для выполнения SQL-запросов, нужно производить сбор данных, то есть самих запросов, и создать функции, использующие ML-алгоритмы, для классификации запросов и отслеживания их использования. Для удобства просмотра результатов хорошей идеей будет добавление визуализации в виде графиков и диаграмм, которые позволят наглядно увидеть статистику использования расширения.

Задачи курсовой работы:

1. Добавить возможность логирования SQL-запросов в сервер.
2. Изучить ML-алгоритмы.
3. Создать инструмент для анализа логов запросов на основе ML-алгоритмов.

4. Добавить визуализацию результатов анализирования SQL-запросов.

5. Оценить результаты работы и дальнейшие пути улучшения и развития.

1. Теоретические основы Obsidian

1.1. Особенности архитектуры Obsidian и расширяемость системы

Obsidian — это современное кроссплатформенное приложение для ведения заметок на основе концепции Zettelkasten и принципов локальной базы данных. В отличие от других аналогов, Obsidian не работает с облачным хранилищем данных или централизованной базой данных: всё содержимое пользователя хранится в виде набора обычных Markdown-файлов, организованных в директории, называемой “vault” (хранилище).

Основные характеристики архитектуры:

- Файловая система как база данных. За счёт того, что приложение работает с файловой системой напрямую, повышается отказоустойчивость.
- Локальное хранение. Приложение не использует облачные хранилища, всё содержится на устройстве пользователя.
- Свободная структура. Обеспечивается гибкая система хранения данных — каждый пользователь может по-своему организовать структуру заметок.

Obsidian изначально строилось как модульная система с плагинной архитектурой. С момента релиза открылись широкие возможности для разработчиков, которые могут создавать пользовательские плагины и расширять функциональность приложения, в том числе за пределами стандартного набора настроек.

Преимущества расширяемости:

- Обширный API на JavaScript/TypeScript;
- Возможность доступа ко всем ключевым модулям ядра приложения;

- Активное сообщество и Marketplace для обмена плагинами;
- Возможность модификации интерфейса и взаимодействия с пользователем;
- Поддержка слушателей событий, что позволяет создавать реактивные плагины.

Таким образом, архитектура Obsidian предоставляет прочную основу для создания интеллектуальных, аналитических и визуальных инструментов на его базе.

2. Методы анализа данных, собранных с помощью расширения Obsidian

2.1. Источники данных и цели анализа

Данная работа направлена на разработку расширения для анализа поведения пользователей на основе SQL-запросов, формируемых ими в процессе работы с плагином Obsidian, с использованием ML-алгоритмов. Расширение анализирует все запросы, которые поступают с сервера. Серверная часть расширения написана на Node.js и PostgreSQL и сохраняет все SQL-запросы и результаты их выполнения в отдельный JSON-файл для последующего анализа.

Источники данных, по которым производится анализ:

- Временные метки и содержание SQL-запросов, посланных пользователем;
- Результаты выполнения (успешность или ошибки);
- Частота повторного выполнения одного и того же запроса;
- Статистика по таблицам, используемым в запросах.

Цель исследования — выявление паттернов поведения пользователей, определение активности по времени, определение популярных типов запросов и базовых аномалий.

2.2. Сравнительный анализ методов обработки SQL-запросов

Перед тем, как выбрать метод для анализа данных, нужно разобрать наиболее распространённые подходы к синтаксическому разбору и извлечению признаков. Важно найти метод, который бы соответствовал требованиям задачи, чтобы обеспечить максимально точное и эффективное представление структурных особенностей SQL-запросов.

Метод	Описание	Преимущества	Недостатки
Полный разбор AST (анализ абстрактного синтаксического дерева)	Используется в компиляторах и интерпретаторах для глубокого анализа кода или оптимизации. Разбирает семантику кода.	Высокая точность; возможность сложного анализа	Является избыточным для обработки простых SQL-запросов, требует использования мощных парсеров
Регулярные выражения	Простейший метод для извлечения ключевых конструкций запроса	Быстрота реализации	Хрупкость, неспособность корректно обрабатывать вложенные конструкции
Разбор через SQL-парсер (использовано в данной работе)	Используется библиотека для разбора SQL (например, `pgsql-ast-parser` или аналогичная), которая возвращает структурированный объект с типом запроса, таблицами, условиями и т.д.	Достаточная точность, лёгкость в использовании, подходит для статистического анализа	Не покрывает сложные случаи (например, нестандартные диалекты SQL), не строит полное дерево

Таблица 1: Сравнение методов анализа SQL-запросов

Исходя из характеристик методов и задач, которые нужно реализовать в анализе, я выбрала третий подход — разбор через SQL-парсер. В условиях данной курсовой работы этого метода достаточно. Он обеспечивает необходимую глубину анализа и при этом имеет разумную сложность реализации. Такой подход наиболее эффективен, так как в анализе будут обрабатываться простые SQL-запросы. Разбор через SQL-парсер позволяет:

- определить тип запроса (SELECT, UPDATE, DELETE и др.);
- извлечь названия таблиц и полей;
- отслеживать повторяемость одних и тех же шаблонов;
- вычислять частотные и статистические признаки запросов.

2.3. Классификация и предварительная обработка запросов

Прежде чем приступить к анализу, собранные SQL-запросы обрабатываются:

- Данные из JSON-строк преобразуются в объекты;
- Определяется тип запроса (SELECT, INSERT и т. д.) на основании первых ключевых слов;
- На основе регулярных выражений извлекается название используемой таблицы.

AST-дерево не строится, весь анализ происходит с использованием собственных эвристик и регулярных выражений.

2.4. Анализ ML-алгоритмов

«Алгоритмы машинного обучения — это алгоритмы, которые автоматически строят модель на основе наблюдаемых данных, с

целью прогнозирования или принятия решений без явного программирования этих действий»[3]

Определения основных алгоритмов машинного обучения:

Линейная регрессия: Модель, которая устанавливает линейную связь между входными признаками и числовым результатом, чтобы предсказать значение на основе этой зависимости.

Логическая регрессия: Классификатор, который с помощью логической функции вычисляет вероятность принадлежности объекта к определённому классу.

Дерево решений: Модель, разделяющая данные на группы по определённым признакам, формируя древовидную структуру решений.

Случайный лес: Совокупность нескольких деревьев решений, которые совместно голосуют за итоговый результат, повышая точность и снижая переобучение.

Метод опорных векторов (SVM): Алгоритм, который ищет оптимальную границу (гиперплоскость), максимально разделяющую классы в пространстве признаков.

Градиентный бустинг (XGBoost): Последовательный ансамбль слабых моделей, где каждая новая исправляет ошибки предыдущих, что улучшает качество предсказаний.

Нейронные сети: Модели, вдохновлённые работой мозга, способные выявлять сложные зависимости и закономерности в данных.

K-ближайших соседей (KNN): Метод, определяющий класс или значение объекта на основе сходства с ближайшими по признакам примерами из обучающей выборки.

Наивный байес: Простая вероятностная модель, предполагающая независимость признаков для классификации объектов.

ARIMA: Статистическая модель, предназначенная для анализа и прогнозирования временных рядов, учитывающая автокорреляцию и тренды.

Isolation Forest: Алгоритм для выявления аномалий, изолирующий выбросы через случайные разбиения данных.

DBSCAN: Метод кластеризации, который группирует точки, основываясь на плотности данных, позволяя выделять кластеры различной формы.

K-Means: Алгоритм кластеризации, разделяющий данные на заданное число групп, минимизируя разброс внутри кластеров.

PCA: Метод снижения размерности, преобразующий признаки в новый набор переменных, объясняющих основную вариативность данных.

Частотный анализ (EDA): Метод предварительного изучения данных, направленный на выявление закономерностей и особенностей с помощью визуализации и статистики.

При выборе алгоритмов нужно учитывать такие условия, как объём, структура данных, требуемая скорость и интерпретируемость.

Выбор алгоритма зависит от задачи (классификация или регрессия), объёма и структуры данных, требований к интерпретируемости и скорости.

Для сложных и больших данных лучше подходят ансамблевые методы (Random Forest, Gradient Boosting) и нейронные сети.

Для простых, линейных задач — линейные/логистические регрессии.

Если данные очень сложные или иерархичные — нейронные сети.

Для быстрой прототипной оценки можно использовать деревья решений и KNN.

Таблица 2: Сравнение алгоритмов анализа данных

Алгоритм	Плюсы	Минусы	Пример
Линейная регрессия	Простая, быстрая, интерпретируемая	Плохо с нелинейностями, чувствительна к выбросам	Прогноз числовых значений, например, оценка продаж или затрат
Логистическая регрессия	Быстрая, легко интерпретируется	Не подходит для сложных нелинейных данных	Классификация бинарных признаков (фильтрация по категориям)
Дерево решений	Интерпретируемость, работа с категориальными данными	Может переобучаться, чувствительно к шуму	Классификация и сегментация данных в аналитике
Случайный лес	Высокая точность, устойчив к шуму	Менее интерпретируем, ресурсоемок	Улучшенная классификация и регрессия на больших наборах данных
Метод опорных векторов (SVM)	Эффективен в высоких размерностях	Медленнее на больших данных, требует настройки	Классификация с четкими границами, например, аномалии

Продолжение на следующей странице

Алгоритм	Плюсы	Минусы	Пример
Градиентный бустинг (XGBoost)	Очень высокая точность	Сложнее в настройке, риск переобучения	Анализ табличных данных с высокой сложностью признаков
Нейронные сети	Моделируют сложные нелинейности	Требуют больших данных и ресурсов	Обработка текстов, временных рядов, сложных паттернов
К-ближайших соседей (KNN)	Простота, не требует обучения	Медленный на больших данных, чувствителен к метрике	Быстрая классификация по похожим данным
Наивный байес	Быстрый, прост в реализации	Предположение независимости признаков часто не выполняется	Классификация текстов, например, спам-фильтры
ARIMA	Хорош для стационарных временных рядов	Не подходит для сложных нелинейных данных	Прогнозирование временных рядов, например, продаж или трафика
Isolation Forest	Эффективен для аномалий	Менее точен при плотных данных	Обнаружение аномалий в логах и транзакциях

Продолжение на следующей странице

Алгоритм	Плюсы	Минусы	Пример
DBSCAN	Выделяет кластеры произвольной формы	Чувствителен к параметрам, плохо с разреженными данными	Кластеризация схожих записей без заранее заданного числа кластеров
K-Means	Быстрый, простой	Требует число кластеров, чувствителен к выбросам	Сегментация пользователей, товаров и т.п.
PCA	Снижает размерность, выявляет важные признаки	Потеря интерпретируемости	Уменьшение числа признаков для ускорения запросов и анализа
Частотный анализ (EDA)	Прост в применении, наглядный	Не автоматизирован	Предварительный анализ данных, выявление закономерностей и аномалий

2.5. Анализ функций анализатора и ML-алгоритмы

2.5.1 Функции анализатора

- classifyQueries(): группирует SQL-запросы по таблицам и типам, считает их частоту.

  Похож на кластеризацию по меткам или группировку. Это метод предобработки и агрегирования.

  Аналогия с ML: частотное кодирование категориальных признаков.

- predictActivity(): прогнозирует активность на следующий час на основе предыдущих значений.

  Похож на простые методы временных рядов: скользящее среднее, экспоненциальное сглаживание, AR.

  Аналогия с ML: ближе к статистике, но родственен ARIMA и простым регрессиям.

- detectAnomalies(): выявляет пары (тип + таблица) с числом запросов выше порога.

  Похож на: thresholding.

  Аналогия с ML: простейшее обнаружение выбросов, как в контрольных картах, но без обучения. В ML используется Isolation Forest, One-Class SVM.

- segmentByTables(): подсчёт частот запросов по типам для каждой таблицы.

  Похож на: агрегацию.

  Аналогия с ML: подготовка признаков, возможна сегментация на основе частот.

- getPopularQueryTypes(): подсчитывает и сортирует типы запросов по частоте.

  Похож на: EDA.

  Аналогия с ML: предварительный анализ распределения, не ML, но полезен на стадии EDA.

2.5.2 Итог

Функции анализатора в текущем виде представляют собой:

- Предварительную обработку и частотное кодирование;
- Простейший анализ временных рядов;

- Базовое обнаружение аномалий;
- Этапы подготовки признаков для последующего машинного обучения.

Таблица 3: Аналогии между функциями анализатора и ML-алгоритмами

Функция в анализаторе	Похожий ML-подход	Возможная замена / дополнение
`classifyQueries()`	Классификация на основе частот — Naive Bayes, Decision Tree	Автоклассификация типов запросов с учётом распределения по таблицам и типам
`predictActivity()`	Простые модели временных рядов — ARIMA, линейная регрессия, LSTM	Прогноз активности по часам с помощью моделей временных рядов
`detectAnomalies()`	Пороговые правила, Outlier Detection — Isolation Forest, DBSCAN	Более гибкие и адаптивные методы аномалий с обучением
`segmentByTables()`	Кластеризация — K-Means, PCA	Автоматическая сегментация таблиц на основе частот активности
`getPopularQueryTypes()`	Частотный анализ признаков — EDA	Визуализация и группировка через кластеризацию или PCA

3. Реализация серверной части и анализатора

В процессе реализации были использованы следующие библиотеки и модули:

Node.js

Node.js — это открытая платформа для выполнения JavaScript вне браузера. Она позволяет создавать серверные приложения, динамические веб-страницы и утилиты командной строки. В основе Node.js лежит событийно-ориентированная архитектура с неблокирующими операциями ввода-вывода, что обеспечивает высокую производительность и эффективность.

PostgreSQL

PostgreSQL — это бесплатная open-source система управления базами данных, построенная на объектно-реляционной модели. Она поддерживает сложные SQL-запросы, отличается надежностью и возможностью масштабирования.

Модуль pg

Модуль pg предназначен для взаимодействия с PostgreSQL из Node.js. С его помощью можно отправлять SQL-запросы к базе данных и получать результаты.

CORS и Express

CORS — это механизм, который позволяет веб-приложениям обращаться к ресурсам, размещённым на других доменах. В Express-приложениях для настройки CORS используется специальный middleware, который управляет соответствующими HTTP-заголовками.

Express — популярный фреймворк для создания веб-серверов и API.

Readline

Readline — встроенный в Node.js модуль, который облегчает ввод данных с консоли. Он позволяет интерактивно запрашивать информацию у пользователя, например, для ввода API-ключа Grafana перед отправкой запросов.

Axios

Axios — это библиотека для выполнения HTTP-запросов. Её применяют для отправки POST-запросов к Grafana API с настройками дашборда. Axios поддерживает промисы и удобную обработку ошибок, что помогает интегрировать API-вызовы в асинхронный код.

fs.promises

Встроенный модуль Node.js для работы с файловой системой в асинхронном режиме с использованием промисов.

Описание работы программы

Для обеспечения полной цепочки анализа от сбора данных до визуализации была разработана вспомогательная утилита на Node.js, реализующая следующие этапы:

- Чтение и предобработка логов SQL-запросов, сохранённых в формате JSON;
- Автоматическая классификация запросов по типу (SELECT, UPDATE и т.д.), а также извлечение имени целевой таблицы;

- Построение дополнительных признаков: час выполнения запроса, день недели, длина запроса;
- Обнаружение аномалий в поведении (например, чрезмерная нагрузка на отдельную таблицу);
- Сегментация запросов по таблицам и типам действий;
- Прогнозирование пользовательской активности на следующий час на основе временных паттернов;
- Выявление самых популярных SQL-команд в пользовательской истории;
- Интеграция с системой визуализации Grafana: построение автоматических дашбордов на основе логов.

В процессе работы утилита взаимодействует с пользователем через CLI-интерфейс: запрашивает API-ключ от Grafana и формирует соответствующие графики. Среди отображаемых панелей:

- Гистограмма распределения типов запросов;
- Линейный график активности по часам.

Для генерации запросов в Grafana используется её HTTP API и внутренняя поддержка SQL через источник данных PostgreSQL. Благодаря этому визуализация обновляется динамически на основе логов, полученных из расширения Obsidian.

3.1. Архитектура серверного приложения

Серверная часть расширения реализована на платформе Node.js с использованием фреймворка Express для обработки HTTP-запросов и подключения к базе данных PostgreSQL через библиотеку pg. Основные функции сервера:

- Прием SQL-запросов от клиента;
- Выполнение этих запросов в базе данных;
- Логирование всех запросов и результатов в файл в формате JSON;
- Предоставление API для получения списка таблиц.

3.2. Логирование SQL-запросов

Я дополнила реализацию сервера логированием SQL-запросов с помощью PostgreSQL и хранения в JSON-формате. Каждое событие записывается в лог с указанием текста запроса, временной метки, результата (успешно или с ошибкой).

Для анализа данные экспортируются в файл `sql_queries_log.json`, содержащий логи в формате JSON по одной записи на строку. Такой подход позволяет обеспечить как надёжное хранение, так и удобный доступ для обработки.

Запись в файл реализована асинхронно через модуль fs.promises, что позволяет эффективно сохранять поток данных без блокировки основного потока сервера.

3.3. Модуль анализа логов

Анализ логов реализован отдельным Node.js-приложением, задача которого — извлечь ключевые метрики и выявить закономерности в поведении пользователей.

Основные этапы анализа:

Предварительная обработка данных:

- Чтение логов из файла;
- Парсинг каждой строки в JSON-объект;

- Семантическое обогащение — извлечение из каждого запроса типа (SELECT, INSERT и др.), имени таблицы, временных характеристик (час, день недели).

Классификация запросов по типам и таблицам:

Для классификации используется простой подсчёт встречаемости запросов разных типов для каждой таблицы. Это позволяет выделить самые используемые таблицы и запросы, что служит основой для анализа нагрузки и поведения.

Прогнозирование активности:

Активность пользователя прогнозируется на основе временных данных (количество запросов по часам

3.4. Использованные алгоритмы и методы

В рамках реализованного приложения применяются следующие алгоритмы и методы:

Классификация на основе регулярных выражений

Извлечение типа запроса и имени таблицы из SQL текста реализовано с помощью регулярных выражений и строковых операций (функции `getQueryType` и `getTableName`).

Агрегация и подсчёт частот

Используется для группировки и подсчёта количества запросов по типам и таблицам (`classifyQueries`, `segmentByTables`, `getPopularQueryTypes`).

Простейшее прогнозирование временной активности

На основе подсчёта количества запросов по часам строится прогноз активности на следующий час (`predictActivity`).

Детекция аномалий через пороговое значение

Определение аномалий происходит путём фильтрации запросов с частотой, превышающей установленный порог (например, более 50 запросов) (`detectAnomalies`).

Подготовка данных для визуализации

Формирование данных для создания дашборда в Grafana с помощью SQL-запросов, автоматически генерируемых из обработанных логов (функция `createGrafanaDashboard`).

3.5. Интеграция с Grafana, визуализация

Чтобы показать результаты исследования, дополнение оснащено встроенным инструментом создания для приборной панели на платформе Grafana. Это позволяет пользователю видеть основные функции поведения в форме интерактивных диаграмм и графиков.

О Grafana

Grafana — это open-source приложение для отображения и изучения данных, которые могут ссылаться на различные источники данных, такие как PostgreSQL. Он предлагает простую в использовании платформу для создания панелей мониторинга, диаграмм, электронных таблиц и диаграмм, используя текущие или прошлые данные из обработки запросов.

Архитектура REST API

REST (Representational State Transfer) — это шаблон проектирования для распределенных систем, особенно веб-приложений. Он основан на ряде принципов, таких как:

- Архитектура клиент-сервер: клиент и сервер логически разделены, что обеспечивает независимость интерфейса от данных.
- Отсутствие состояния (Stateless): каждый запрос от клиента на сервер должен содержать всю необходимую информацию для его обработки.
- Единообразие интерфейса: взаимодействие осуществляется через стандартные методы HTTP (GET, POST, PUT, DELETE и так далее).
- Кэширование: ответы могут быть закэшированы, что повышает производительность.
- Иерархическая адресация ресурсов: каждый ресурс имеет уникальный URI.

Применение REST в Grafana

Grafana API реализует REST-стиль, что позволяет легко автоматизировать управление системой и интегрировать её с другими сервисами. Взаимодействие с API Grafana осуществляется через стандартные HTTP-запросы:

- `GET /api/dashboards/uid/:uid` — получить информацию о дашборде;
- `POST /api/dashboards/db` — создать или обновить дашборд;

- `DELETE /api/dashboards/uid/:uid` — удалить дашборд;
- `GET /api/datasources` — получить список подключённых источников данных.

Каждый такой запрос работает с определённым ресурсом (дашбордом, источником данных и т.п.), и имеет предсказуемое поведение, соответствующее HTTP-методу. Таким образом, Grafana служит мощным инструментом анализа и визуализации, который дополняет реализацию REST-сервиса и повышает удобство работы с системой.

Процесс интеграции

Пользователь при запуске анализатора вводит API-ключ Grafana. Этот ключ необходим для аутентификации и авторизации при отправке HTTP-запросов к REST API Grafana. Ключ запрашивается в интерактивном режиме через `readline`. API-ключ используется исключительно для отправки POST-запроса на создание дашборда и не сохраняется в системе. Это обеспечивает безопасность данных.

Структура дашборда

Сформированный дашборд состоит из двух основных панелей:

- Распределение типов запросов Представлено в виде столбчатой диаграммы (bar chart). Показывает количество запросов каждого типа (SELECT, INSERT, UPDATE, DELETE) за весь период анализа. Данные подготавливаются динамически на основе ранее обработанных логов.
- Активность по часам Представлена в виде линейного графика (line chart). Отображает распределение запросов по времени суток, что позволяет выявить пики пользовательской активности.

Отправка конфигурации

После формирования конфигурации она отправляется в Grafana по URL `http://localhost:3000/api/dashboards/db`. Запрос формируется с использованием `axios`.

Особенности реализации

- Назначение дашборда — «Автоматический дашборд», он всегда сохраняется в корневую папку (`folderId: 0`) и может быть перезаписан (`overwrite: true`);
- Все визуализируемые данные предварительно аггрегируются вручную, без использования SQL-базы Grafana как источника данных.

Заключение

В рамках данной курсовой работы была исследована проблема анализа SQL-запросов с целью повышения осознанности и эффективности взаимодействия пользователя с базой данных. Актуальность темы обусловлена широкой распространённостью СУБД и необходимостью разработки инструментов, помогающих пользователю отслеживать, анализировать свои запросы. С помощью визуализации через Grafana пользователю открываются новые возможности применения расширения, например, это может быть полезно для преподавателей, которые могут отслеживать графики активности студентов.

В ходе работы были дополнены серверные компоненты существующей системы, реализующие хранение и обработку пользовательских запросов.

Основным вкладом было создание инструмента для анализа запросов SQL, который может собирать данные, определять общие закономерности и строить графики на их основе. Методы синтаксического разбора и анализа, основанного на эвристике и ML-алгоритмах, были применены для классификации запросов, определения самых загруженных таблиц и групп, предсказаний.

Реализованное приложение обеспечивает полноценный цикл сбора, хранения и анализа логов SQL-запросов пользователя. Использование простых, но эффективных алгоритмов подсчёта и классификации запросов позволило получить ценные метрики поведения, а интеграция с Grafana — визуализировать данные и облегчить интерпретацию результатов.

Реализация обеспечивает быстрый и наглядный анализ поведения пользователя. Это делает систему прозрачной, управляемой и легко масштабируемой.

В ходе выполнения курсовой работы были приобретены следующие

практические навыки:

- разработка серверной логики и работа с существующим backend-кодом,
- синтаксический анализ SQL-запросов,
- проектирование и реализация простых аналитических инструментов,
- интеграциия новых компонентов в существующую архитектуру проекта,
- работа с REST API.

Таким образом, цели курсовой работы были достигнуты. Анализатор и серверная часть являются частью фундамента системы, которая реализует анализ SQL-запросов и визуализацию статистики.

Пути улучшения

Если нужно проводить более глубокий и точный анализ, можно использовать другие алгоритмы, например:

- Более подходящим было бы использовать модели, которые учатся выявлять аномалии на данных (Isolation Forest, Autoencoders).
- Для прогнозирования активности, бизнес-метрик или метрик качества можно использовать ARIMA, LSTM или другие модели временных рядов.
- Для распределения запросов по типам и таблицам, можно использовать классификаторы или кластеризацию.

Список литературы

1. Документация Obsidian API [Электронный ресурс]. — URL: `https://publish.obsidian.md/api` (дата обращения: 17.05.2025).

2. Документация PostgreSQL [Электронный ресурс]. — URL: `https://www.postgresql.org/docs/` (дата обращения: 17.05.2025).

3. Ахо А.В., Ульман Дж.Д. Теория синтаксического анализа, перевода и описания языков. — М.: Мир, 1978. — Т. 1. — 406 с.

4. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D. Compilers: Principles, Techniques, and Tools. — Addison-Wesley, 2007. — 1009 с.

5. Pratt, T.W., Zelkowitz, M.V. Programming Languages: Design and Implementati — Pearson Education, 2001. — 576 с.

6. SQLite Parser Generator Lemon [Электронный ресурс]. — URL: `https://sqlite.org/lemon.html` (дата обращения: 17.05.2025).

7. ANTLR – Parser Generator [Электронный ресурс]. — URL: `https://antlr.org/` (дата обращения: 17.05.2025).

8. Никитенко А.Г. Проектирование трансляторов вычислительных систем. — СПб.: СПбГУ ИТМО, 2010. — 120 с.

9. Митчелл Т.М. Введение в машинное обучение. — М.: Вильямс, 2000. — 400 с.

10. Бишоп К.М. Распознавание образов и машинное обучение [Pattern Recognition and Machine Learning]. — СПб.: Springer, 2006. — 738 с.

11. Джеймс Г., Уиттен Д., Хасти Т., Тибширани Р. Введение в статистическое обучение [An Introduction to Statistical Learning]. — СПб.: Springer, 2013. — 440 с.

12. Сухотин И.Е. Статистика и анализ данных. — М.: Наука, 2010. — 352 с.

13. Гудфеллоу И., Бенжио Й., Курвиль А. Глубокое обучение [Deep Learning]. — М.: MIT Press, 2016. — 800 с.

14. Жерон А. Погружение в машинное обучение с помощью Scikit-Learn, Keras и TensorFlow [Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow]. — М.: O'Reilly Media, 2019. — 812 с.

15. Максвелл Дж. Г., Маккин П. М. Анализ временных рядов [Time Series Analysis]. — М.: Wiley, 2007. — 520 с.

16. Тьюки Дж. В. Исследовательский анализ данных [Exploratory Data Analysis]. — Реддинг: Addison-Wesley, 1977. — 688 с.

17. Документация Node.js [Электронный ресурс]. — URL: `https://nodejs.org/en/docs/` (дата обращения: 17.05.2025).

18. Документация Grafana [Электронный ресурс]. — URL: `https://grafana.com/docs/` (дата обращения: 17.05.2025).