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

Факультет информатики, математики и компьютерных наук

Программа подготовки бакалавров по направлению
38.03.05 Бизнес-информатика

Чуркина Екатерина Николаевна

КУРСОВАЯ РАБОТА

«Разработка расширения для Obsidian. Архитектура и разработка расширения»

Научный руководитель
старший преподаватель НИУ
ВШЭ - НН

Саратовцев Артем Романович

Нижний Новгород, 2025г.

Содержание

1. Введение

На сегодняшний день невозможно представить успешное функционирование различных компаний и организаций без развитой информационной системы и баз данных, которые позволяют автоматизировать сбор и обработку данных. Именно поэтому невозможно переоценить, насколько важно в современном мире работа с реляционными базами данных и SQL, которая возможна благодаря системным аналитикам и разработчикам.

Существует множество онлайн-платформ, тренажеров, где начинающие разработчики и аналитики могут практиковать свои навыки в SQL в интерактивном формате, например: Stepik, LeetCode, SQL MurderMystery. Тем не менее, несмотря их обилие, функциональность подобных интерактивных платформ зачастую сводится к решению узкоспециализированных задач без поддержки полноценной документации, визуализации и интерактивного конспектирования. Кроме того, перечисленные инструменты доступны исключительно с доступом к интернету, что накладывает некоторые ограничения на образовательный процесс.

С другой стороны, существуют мощные инструменты для систематизации информации (Notion, Evernote, Obsidian), но их применение в контексте работы с базами данных остается недостаточно функциональным и полным из-за отсутствия интеграции с СУБД и возможностей развития практических навыков пользователя. Как следствие, пользователи вынуждены прибегать к фрагментированному воркфлоу, используя отдельные приложения для практики и ведения документации, что значительно снижает эффективность обучения и уменьшает академический прогресс.

Данная работа направлена на создание расширения, которое не только позволит интегрировать СУБД PostgreSQL в Obsidian, но и предоставит возможность пользователю создавать и вести конспекты и документацию, визуализировать данные и практиковать свои навыки с помощью автономного SQL-тренажера без подключения к интернету, что выделяет это расширение на фоне других сервисов. Таким образом, актуальность работы продиктована

необходимостью создания специализированного решения, которое позволяет пользователям объединить функционал нескольких инструментов, создавая единую эффективную оффлайн среду обучения, которая станет полезным инструментом не только для студентов, только изучающих основы реляционных баз данных, но и для опытных разработчиков, нуждающихся в удобном инструменте для проектирования и документирования сложных структур.

Целью данной курсовой работы является изучение архитектуры плагинов и разработка расширения для Obsidian, которое позволит пользователям эффективно взаимодействовать с реляционными базами данных и развивать навыки работы с SQL. В процессе работы будут исследованы технологии разработки расширений для Obsidian, а также методы интеграции с PostgreSQL.

Для достижения поставленной цели необходимо решить следующие задачи:

1) Изучение теоретических основ разработки расширений для Obsidian.

2) Поиск и анализ существующих решений в области интеграции баз данных и Obsidian.

3) Проектирование архитектуры плагина и описание функционала.

4) Разработка плагина.

5) Оценка дальнейших возможных оптимизаций и развития работы.

2. Теоретические основы разработки расширений Obsidian.

2.1. Приложение Obsidian и его ключевые особенности.

Obsidian – это кроссплатформенное приложение для работы с информацией, основанное на принципах локальных файлов Markdown. Obsidian не просто реализует базовую поддержку Markdown — фреймворк приложения целиком построен на этом языке разметки, обеспечивая интуитивно понятное и продвинутое форматирование текста пользователем. Более того, благодаря архитектуре, основанной на локальном хранении файлов, все данные хранятся непосредственно на жестком диске устройства пользователя, обеспечивая полную локализацию и безопасность данных, что таким образом исключает риски облачных решений (например, утечки данных). Также приложение позволяет создавать не только заметки, но и целостную систему взаимосвязанных знаний благодаря его функциональной особенности – создание двусторонних ссылок между файлами. Таким образом, пользователь имеет возможность создать не только систему взаимосвязанных заметок, но и визуализировать свои идеи с помощью mind-map, а также построить полноценную базу знаний, где каждая часть информации будет логично и удобно связана с другими. Приложение доступно на ряде операционных систем (Windows, macOS, Linux, iOS, Android) и работает без доступа к сети.

Несомненно, одна из наиболее востребованных особенностей приложения – это мощнейшая система плагинов и развитая система взаимодействия разработчиков с пользователями. Создатели приложения активно работают над его экосистемой и всячески поощряют вклад пользователей: Obsidian обладает расширяемой архитектурой, поддерживая как официальные, так и сторонние плагины, которые активно разрабатываются вовлеченной аудиторией. Пользователи совершенствуют базы знаний с помощью ряда расширений, позволяющих адаптировать приложение для конкретных целей, что делает Obsidian незаменимым инструментом в различных сферах.

2.2. Основные принципы разработки плагинов Obsidian.

Говоря о техническом стеке, архитектуре и основных положениях, которые касаются разработки расширений в Obsidian, есть несколько ключевых аспектов.

Во-первых, разработка плагинов для Obsidian, согласно официальным рекомендациям создателей приложения, происходит на языках программирования JavaScript или TypeScript (который по своей сути является строго типизированным надмножеством JavaScript и в конечном счете при запуске плагина компилируется в JS). Также в процессе разработки важную роль играет Node.js — серверная платформа, обеспечивающая работу npm для управления зависимостями (включая официальный пакет obsidian с API) и автоматизацию workflow (тестирование, сборку, публикацию). Готовые плагины, в свою очередь, публикуются и распространяются на GitHub.

Во-вторых, в основе любого плагина лежит класс, наследующий от базового класса Plugin – он предоставляет доступ ко всем необходимым типам и API для интеграции с редактором. Благодаря такому подходу, сторонние плагины, разрабатываемые вовлеченной аудиторией, реализуют стандартизированный интерфейс взаимодействия с ядром Obsidian, а также обеспечивают четкий жизненный цикл плагина, соответствующий рекомендациям по разработке. Другими словами, данный класс играет роль полноценного адаптера между кодом и сложной внутренней архитектурой приложения.

Плагины в Obsidian представляют из себя модульные расширения, построенные по принципам компонентно-ориентированной архитектуры. В их структуре можно выделить три основополагающих элемента, определяющих способ взаимодействия с основным приложением. К ним относятся:

1) Инициализация плагина с помощью файла manifest.json. Этот структурированный файл является начальной точкой для интеграции разрабатываемого расширения в экосистему Obsidian и содержит ключевую информацию о нём. В первую очередь он включает в себя идентификационные данные

– уникальный идентификатор, название расширения и его текущую версию. Далее необходимы технические требования и охарактеризовывающая информация: к ним относятся, например, минимальная версия Obsidian, позволяющая интегрировать плагин, а также поле Description, author и т.д. с прочими данными о функционале и разработчике.

2) Главный модуль плагина – main.js, содержащий и реализующий основную логику и функционал разрабатываемого расширения.

3) Ресурсы и интерфейсные доработки плагина: стили, UI-шаблоны или дополнительные скрипты (чаще всего задаются с помощью файла styles.css), позволяющие кастомизировать внешний вид расширения.

Говоря о динамических аспектах работы плагина, можно отметить, что жизненный цикл любого плагина строго регламентирован, состоит из нескольких фаз и управляется как минимум двумя основными методами, которые гарантируют стабильность работы расширения и предотвращают ошибки и утечки памяти.

• onload() – метод, отвечающий за инициализацию плагина. В момент его вызова (т.е. включения плагина) расширение регистрирует свои команды и горячие клавиши, добавляет предусмотренные UI-элементы и инициализирует состояние плагина. После успешной инициализации плагин переходит в рабочее состояние, в котором реагирует на пользовательские действия и обрабатывает системные события.

• onunload() – метод, отвечающий за корректное завершение и деактивацию плагина.

Таким образом, типичный процесс разработки плагина включает в себя:

1) Настройку среды (Node.js, git, редактор кода)

2) Инициализацию проекта на основе шаблона (стандартизированного интерфейса и жизненного цикла расширений)

3) Написание логики на JavaScript/TypeScript через Obsidian API
4) Локальное тестирование
5) Сборку production-версии
6) Публикацию через GitHub

3. Анализ методов интеграции СУБД в Obsidian.

3.1. Архитектурные ограничения и технические требования

Поскольку разрабатываемое расширение предусматривает в первую очередь взаимодействие с базами данных PostgreSQL через Obsidian, важным аспектом данной работы является исследование способов интеграции выбранной СУБД в плагин. Необходимо рассмотреть технические особенности Obsidian и необходимых библиотек для их корректной работы и коллаборации внутри расширения.

Говоря о технических особенностях Obsidian, важно подчеркнуть, что приложение разработано на Electron. Electron – это фреймворк с открытым исходным кодом, позволяющий разрабатывать десктопные GUI-приложения, используя frontend- и backend- компоненты. Для первого Electron использует движок Chromium для отображения интерфейса (иными словами – для рендер-процесса), что в свою очередь предоставляет полную совместимость с веб-технологиями (JavaScript, HTML и т.д.). Однако в контексте использования и интеграции некоторых технологий накладывает определенные ограничения: возможно использование только стандартных веб-API и по умолчанию рендеринг не имеет доступа к бинарным протоколам и не поддерживает прямое TCP-подключение.

Тем временем, PostgreSQL требует прямого доступа к TCP-сокетам через нативную библиотеку pg, которая, в свою очередь, использует бинарные протоколы для обмена данными с СУБД. Таким образом, PostgreSQL и Obsidian очевидно не могут работать напрямую. Более того, для обеспечения безопасности передачи данных важно изолировать процесс выполнений запросов в СУБД от действий пользователя внутри Obsidian.

Основным вариантом для решения данной проблемы является создание и использование локального сервера, который по сути будет играть роль моста между Obsidian (а именно его частью Chromium) и библиотекой pg. Тем не менее, в контексте разработки данного плагина, учитывая необходимость обеспе-

чить двустороннюю коммуникацию сервера и клиента в рамках ограничений Electron и с поддержкой необходимых операций PostgreSQL, выбор решений закономерно сводится к двум вариантам: использование HTTP/HTTPS протокола в сочетании с технологией Express.js или использование WebSocket. Оба подхода полностью соответствуют техническим требованиям и могут обеспечить стабильную работу расширения. Чтобы сделать оптимальный выбор, необходимо детально проанализировать их ключевые характеристики и сопоставить с функциональными задачами проекта.

3.2. Сравнительный анализ подходов и обоснование выбора

Для выбора оптимального решения необходимо изучить несколько аспектов и выделить ключевые моменты, важные для разработки плагина. Во-первых, необходимо проанализировать и структурировать информацию о том, как именно сервер будет взаимодействовать с основной частью кода и как часто будет необходимо обращение к СУБД через сервер. Далее, опираясь на полученные доводы, определить критические функции сервера, а также в общем разобрать два подхода и в конце концов прийти к однозначному выводу: выбору решения, соответствующего архитектуре и функционалу сервера.

Вкратце, разрабатываемое расширение для Obsidian предполагает форматирование SQL-запросов и их выполнение с помощью PostgreSQL. Помимо этого, плагин должен уметь работать с метаданными из СУБД (например, с списками таблиц и пр.), а также визуализировать данные с помощью таблиц. Таким образом, можно отметить несколько ключевых особенностей взаимодействия плагина:

- Локальность работы, присущая Obsidian в целом, остается базисом в архитектуре данного расширения. Иными словами, все компоненты (Obsidian, сервер, БД) работают локально на одной машине.
- Можно предположить неконстантный, непостоянный характер запросов: ожидается, что пользователь в процессе документирования или конспектирования будет периодически с помощью горячих клавиш получать результаты запросов от СУБД.
- Аналогично можно предположить, что данные будут преимущественно статичны во время выполнения запроса.

Как было упомянуто выше, сервер играет роль “моста” между СУБД и Obsidian. Тем не менее, можно также выделить несколько критических задач сервера в архитектуре плагина. Во-первых, конечно, он обеспечивает выполнение запросов в PostgreSQL и обмен данными с клиентом. Помимо этого,

сервер играет ключевую роль в управлении соединением с БД: он не только обрабатывает аутентификацию (логин, пароль и прочее), но и создает и управляет пулом подключений к БД. Сервер также обрабатывает некоторые ошибки, которые могут возникнуть при использовании расширения. Изучив и проанализировав теоретические аспекты функционала плагина и роли сервера, необходимо рассмотреть выбранные решения.

HTTP (HyperText Transfer Protocol) — это протокол, который позволяет компьютерам обмениваться данными в интернете. Иными словами, в цифровом мире, где приложения обмениваются данными, протокол HTTP выступает универсальным языком общения. Это набор правил, определяющих, как клиент (например, браузер или плагин Obsidian) запрашивает информацию, а сервер её предоставляет. HTTP — это «архитектурный стиль взаимодействия распределенных систем», будь то фронтенд и бэкенд или несколько серверных приложений. Его ключевая особенность — stateless (отсутствие состояния): каждый запрос обрабатывается изолированно, словно библиотекарь, который забывает о предыдущих вопросах сразу после ответа. Это упрощает масштабируемость, но требует передачи всех необходимых данных в каждом запросе.

Для работы с HTTP в среде Node.js существует Express — минималистичный, но мощный фреймворк, который превращает рутинные задачи (маршрутизацию, парсинг данных, обработку ошибок) в интуитивные операции. Express можно сравнить с опытным переводчиком: он принимает «сырые» HTTP-запросы, извлекает из них нужные параметры и форматирует ответ в понятном клиенту виде (JSON, HTML или простой текст).

Связка HTTP и Express — это классика веб-разработки, которая сочетает в себе универсальность протокола с удобством фреймворка. Подобный подход не только поддерживает кэширование, которое играет важную роль для повышения производительности, но и в целом объективно более легкая задача для реализации корректного сервера, благодаря которой можно будет минимизировать ошибки во время взаимодействия Obsidian и PostgreSQL.

Вторым возможным решением в данной ситуации является использование WebSocket. WebSocket – технология двустороннего общения в реальном времени. Такая технология революционна в некотором смысле (по сравнению с http) и имеет свои особенности: главная из них заключается в том, что при установке соединения клиент и сервер «договариваются» о подключении через handshake (рукопожатие). Успешно подключившись, соединение остается открытым до тех пор, пока одна из сторон (клиентская или серверная) не закроют его. Таким образом, такое подключение буквально работает в режиме реального времени. Более того, в общем случае сервер может присылать данные клиенту и без его запроса: например, если пришло какое-то уведомление не серверной стороне. Можно точно сказать, что WebSockets также предоставляют эффективный и простой способ общения клиента и сервера.

В отличие от HTTP, где клиент должен постоянно «стучаться» к серверу за обновлениями, WebSocket устанавливает постоянное двустороннее соединение, позволяя серверу и клиенту обмениваться сообщениями в любой момент без лишних запросов. Возвращаясь к примеру с "библиотекарем говоря о данном подходе, можно по аналогии интерпретировать это так: WebSocket – это "библиотекарь"с отличной памятью. Пока каждое соединение с http — это новый "разговор с нуля WebSocket "gомнит"предыдущие взаимодействия в рамках одной сессии. За счет этого значительно снижается количество handshake-запросов, что в глобальном контексте может существенно уменьшить нагрузку и улучшить эффективность сервера.

Для наиболее наглядного сравнения и анализа выбранных двух подходов в контексте разрабатываемого расширения, все данные собраны в следующей таблице. В ней все критерии рассмотрены с точки зрения уместности применения технологий для Obsidian и пользователя.

Критерий	Использование HTTP + Express.js	Использование WebSocket
Модель взаимодействия	Запрос-> ответ-> закрытие соединения	Двустороннее соединение
Ресурсы	Небольшое количество, поскольку соединение разрывается каждый раз	Может быть большое количество, т.к. долгие соединения потребляют память
Безопасность	Локальный localhost	Локальный localhost
Поддержка в Electron	Полная (через fetch в Express.js)	Полная (через WebSocket API)
Длительные операции	Возможны таймауты	Может работать непрерывно долгое время
Стабильность и прогресс	В случае сброса соединения происходит потеря данных	В случае сброса соединение есть вохможность продолжить операцию
Сложность реализации и поддержания	Нормальная	Высокая (в т.ч. из-за необходимости наладить управление состоянием соединения)

Таблица 1: Сравнение выбранных подходов

4. Разработка расширения для Obsidian.

4.1. Реализация серверной части.

4.2. Внедрение интерактивного тренажера SQL.

4.3. Интерфейсные доработки расширения.

4.4. Реализация функционала.

5. Заключение.

6. Список использованной литературы.