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

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 информации. Именно поэтому невозможно переоценить, насколько важна 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 на их обилие, функциональность подобных интерактивных платформ зачастую сводится к решению узкоспециализированных практических 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 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 ухудшает академический прогресс.

Данная работа направлена на создание расширения, которое не только интегрирует СУБД PostgreSQL в Obsidian, но и реализует принцип *self-broadcasting* — локальное управление и синхронизацию данных без внешних зависимостей. Это позволит 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 полезной платформой 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 в Obsidian.

3. Анализ и интеграция тренажера SQL для выбранной СУБД

4. Проектирование архитектуры плагина

5. Разработка плагина.

6. Оценка проделанной работы и дальнейших возможных путей развития

1. Теоретические основы разработки расширений Obsidian.

1.1. 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 об-

ладает 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.

1.2. 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*» [3] и в конечном счете при запуске плагина компилируется в JS). Также в процессе разработки важную роль играет Node.js — серверная платформа, обеспечивающая работу 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 фундаментальных компонента, 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 плагина в экосистему Obsidian и содержит ключевую информацию о расширении. В первую очередь он включает в себя идентификационные данные – уникальный идентификатор, название расширения и его текущую версию. Далее можно указать технические требования и дополнительную информацию: к ним относятся, например, минимальная версия Obsidian, позволяющая интегрировать плагин, а также поля с 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: он обеспе-

чивает деактивацию плагина, избегая негативного влияние на производительность после завершения работы.

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

2. Анализ методов интеграции СУБД в Obsidian.

2.1. 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»*[6]. 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 ограничения: возможно использование только стандартных веб-API и по умолчанию рендеринг не поддерживает прямое TCP-подключение.

Тем временем PostgreSQL требует прямого доступа к TCP-сокетам через нативную библиотеку pg, поскольку согласно документации: *«только TCP-сокеты могут использоваться для подключения к серверу»* [8]. 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 обеспечить двустороннюю коммуникацию сервера и клиента в рамках ограничений Electron с поддержкой необходимых операций PostgreSQL, выбор решений закономерно сводится к двум вариантам: использование HTTP/HTTPS протокола 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.

2.2. 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, ссписками 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 обмениваться данными в интернете: это набор правил, определяющих, как клиент (например, браузер или плагин Obsidian) запрашивает информацию, а сервер её предоставляет. Его ключевая особенность — *stateless* (отсутствие состояния): каждый запрос обрабатывается изолированно (можно привести такую параллель - http это 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* — мощный фреймворк, который превращает рутинные задачи (маршрутизацию, парсинг данных, обработку ошибок) в интуитивные операции. Аналогично, 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 производительно-

сти, 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, «протокол двунаправленного обмена данными» [10] в реальном времени. Главная особенность этой технологии 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.

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. Пока каждое соединение с http — это новый «разговор с нуля» , WebSocket «помнит» 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. Все критерии рассмотрены с точки зрения уместности применения технологий для Obsidian и пользователя.

Таблица 1: *Сравнение выбранных подходов*

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	Высокая (localhost)	Высокая (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 возможность продолжить операцию
Сложность реализации и поддержания	Нормальная (не требует усложнений архитектуры)	Высокая (в т.ч. из-за необходимости наладить управление состоянием соединения)

Таким образом, учитывая, что в первую очередь, данное расширение – лишь инструмент для эффективной учебы, документации и в целом взаимодействия с PostgreSQL через редактор приложения, где хоть и сложные запросы (например, объемные транзакции) возможны, но маловероятны, мгновенное обновление данных с помощью WebSocket кажется нерелевантным: это может дать условные преимущества, но сложностей и вопросов все равно становится больше. Например, использования WebSocket в данном контек-

сте существенно увеличивает количество потребляемых ресурсов сервера для поддержания соединений, а также в целом усложняет архитектуру приложения, в то время как http и поднятие сервера с помощью Express.js позволит сохранить ресурсы и эффективно реализовать функционал.

Становится понятно, что в текущей ситуации использование второго варианта будет лишь избыточным, где сложность не оправдывает риски и лишь прибавляет недостатков, поэтому выбор в пользу Express — это выбор в пользу проверенного, гибкого и безопасного подхода, идеально соответствующего архитектуре разрабатываемого плагина.

3. Разработка расширения для Obsidian.

3.1. Реализация серверной части.

Исходя из проведенного анализа, для реализации был использован фреймворк Express.js. Его архитектура позволяет гибко обрабатывать входящие запросы: *express.json()* автоматически преобразует тело запросов в JavaScript-объекты, а *cors()* обеспечивает безопасное межсайтовое взаимодействие между плагином Obsidian и сервером.

В основе системы — два ключевых эндпоинта: *GET /tables* возвращает структуру базы данных, а *POST /query* безопасно преобразует запросы из клиентского формата в валидный SQL для PostgreSQL, предварительно проверяя входные данные на корректность.

Для работы с базой данных используется драйвер pg, реализующий пул соединений. Помимо этого, особое внимание уделено обработке ошибок: сервер перехватывает исключения и преобразует технические сообщения об ошибках в понятные уведомления не только в пользовательский интерфейс, но и в логи работы расширения.

Система запуска сервера, в свою очередь, информирует о готовности к работе и в целом предусматривает корректное завершение процессов: такой подход обеспечивает выполнение запросов, интеграцию СУБД в Obsidian, безопасность и изолированность процессов.

3.2. Внедрение интерактивного тренажера SQL.

Изучение SQL становится доступным благодаря ряду онлайн платформ, среди которых SQL Murder Mystery выделяется уникальным сценарием и подходом к обучению. Учитывая, что его оригинальная версия доступна исключительно онлайн, созданное решение, сочетающее в себе *принципы self-broadcasting* и полную интеграцию с PostgreSQL, является отличной возможностью для реализации данной платформы внутри Obsidian: это не просто перенос контента — это глубокая адаптация, сохраняющая суть оригинального тренажера, но дающая новые возможности пользователям.

С точки зрения интеграции, внедрение интерактивного тренажера доступно благодаря данным из п*убличного репозитория SQL Murder Mystery.* Помимо этого, важным пунктов является факт того, что оригинальная версия MurderMystery построена на SQLite – легковесной СУБД , в то время как разработанное расширение предназначено для пользователей PostgreSQL – мощной СУБД с более строгой системой, которая обладает расширенными возможностями безопасности. Именно поэтому важным аспектом этого этапа разработки стал анализ, перенос и адаптация базы данных.

Во-первых, процесс переноса базы данных потребовал работы с инструментом *SQLite Studio.* С помощью данной технологии, структура и данные из оригинальной SQLite-базы были экспортированы в формат SQL (из формата BD, который был предоставлен в публичном репозитории). Затем начался процесс адаптации:

1. Анализ схемы данных и выявление всех зависимостей между таблицами

2. Анализ и преобразование типов данных

3. Проверка и модификация SQL-синтаксиса

В последнем пункте в ходе анализа была замечена важная особенность в данном кейсе, которая заключалась в том, что SQLite использует подход, в котором возможно временное отключение проверки внешних ключей, что позволяет создавать таблицы в любом порядке, даже если они ссылаются друг на друга. Это возможно благодаря встроенному синтаксису SQLlite *PRAGMA*. Такой подход не свойственен PostgreSQL, поэтому, проанализировав возможные исходы, я приняла решение, что наиболее достоверным и безопасным вариантом будет реализация транзакции, в которой последовательно будет создаваться "скелет" базы данных (т.е. все таблицы без связей), и только потом с помощью синтаксиса *ALTER TABLE* и *ADD CONSTRAINT* будут добавлены внешние ключи. В данном случае весь процесс оборачивается в транзакцию для гарантии целостности.

Таким образом, в ходе переноса и интеграции БД, был проведен тщательный анализ различий между СУБД, который позволяет создать эффективное и практичное решение, сохраняющее все преимущества оригинального тренажера.

3.3. Интерфейсные доработки расширения.

Различные UI доработки однозначно могут стать мощным дополнением для любого расширения. По этой причине важным этапом разработки стало изучение и применение CSS (Cascading Style Sheets) и HTML – фундаментальных технологий веб-разработки, которые Obsidian также использует для отображения контента. HTML обеспечивает структуру, а CSS, в свою очередь, отвечает за визуальное представление.

В Obsidian есть возможность переопределять стандартные стили без модификации ядра самого приложения, что является крайне удобным для улучшения интерфейса в рамках плагина. Говоря конкретно о разрабатываемом расширении, важно обеспечить пользователям не только функциональность, но и удобное отображение результатов с читаемыми таблицами с данными, интуитивно понятными кнопками и гармоничными модульными окнами.

При разработке я предусмотрела сценарии, когда пользователю потребуется работать с большими объемами данных в таблицах: для эффективного отображения таблиц была реализована функция formatAsTable, *динамически* генерирующая HTML-структуру таблиц. Благодаря CSS в данной структуре обеспечена адаптивность: при большом количестве столбцов автоматически включается горизонтальная прокрутка, что сохраняет удобство взаимодействия даже с объемными данными.

С помощью CSS я также настроила отображение таблиц, изменила оформление callout-блоков и создала определенный стиль и цветовую палитру. Таким образом, я считаю, что благодаря грамотной работе с CSS и HTML я создала не просто функциональный инструмент, но и удобный в использовании продукт с интуитивно понятным интерфейсом.

4. Заключение.

В ходе выполнения данной курсовой работы я провела комплексное исследование архитектуры плагинов Obsidian и методов интеграции СУБД. В процессе работы я не только изучила ключевые принципы разработки расширений для Obsidian,но и самостоятельно реализовала уникальный плагин на JavaScript, позволяющий просто и эффективно взаимодействовать с реляционными базами данных.

Особую ценность представляет проведенный мной анализ методов интеграции PostgreSQL, в рамках которого я сначала изучила и выявила технические требования и архитектурные ограничения, а затем провела сравнительную оценку подходов для решения проблемы. Таким образом был определен оптимальный метод взаимодействия, учитывающий ограничения со стороны Obsidian и СУБД и обеспечивающий безопасное выполнение запросов.

Помимо этого, для успешной адаптации тренажера SQL MurderMystery был проведен краткий анализ SQLite и PostgreSQL, поскольку было важно безопасно и качественно перенести базу данных между двумя системами.

Актуальность выполненной работы подтверждается не только растущей популярностью Obsidian среди IT-специалистов, но и высокой практической ценностью, так как расширение предоставляет уникальную среду для конспектирования и отрабатывания навыков SQL оффлайн.

Проведенное исследование и практическая реализация проекта дали мне не только теоретические знания, но и ценные практические навыки в ряде аспектов: разработка плагинов для Obsidian, работа с Express.js, работа с миграцией данных. А полученные результаты, в свою очередь, открывают перспективы для дальнейшего развития проекта, например, добавление новых обучающих сценариев и оптимизацию работы с большими объемами данных.

Список литературы

[1] Overview // Obsidian URL: https://obsidian.md/ (дата обращения: 04.03.2025).

[2] Walker M. The Power of Obsidian: A Digital Approach to Note-Taking //U David Shaffer (ur.), More Than Words: Teaching for a Better World. – 2022. – С. 293-304.

[3] Bhattacharyya S., Nath A. Application of TypeScript Language: A Brief Overview //International Journal of Innovative Research in Computer and Communication Engineering. – 2007. – Т. 4. – С. 10585-10590.

[4] Plugins. Build a plugin. // Obsidian Docs URL: https://docs.obsidian.md/Plugins/Getting+started/Build+a+plugin (дата обращения: 04.03.2025).

[5] Plugins. Anatomy of a plugin. // Obsidian Docs URL: https://docs.obsidian.md/Plugins/Getting+started/Anatomy+of+a+plugin (дата обращения: 04.03.2025).

[6] Alymkulov D. Desktop Application Development Using Electron Framework: Native vs. Cross-Platform. – 2019.

[7] Security // Electronjs URL: https://www.electronjs.org/docs/latest/tutorial/security (дата обращения: 16.04.2025).

[8] PostgreSQL Documentation: Connections and Authentication // PostgreSQL URL: https://www.postgresql.org/docs/current/runtime-config-connection.html (дата обращения: 11.04.2025).

[9] Никонова Е. З., Королев Р. И. Анализ архитектурного стиля rest API //Современные вопросы устойчивого развития общества в эпоху трансформационных процессов. – 2023. – С. 176-179.

[10] Сухов К. WebSockets-стандарт современного веба. Часть 1 //Системный администратор. – 2014. – №. 5. – С. 49-53.

[11] МАДЕЕВА В. А. СРАВНЕНИЕ РЕЛЯЦИОННЫХ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ //Будущее науки-2020. – 2020. – С. 310-312.

[12] Routing // Expressjs URL: https://www.electronjs.org/docs/latest/tutorial/security (дата обращения: 27.04.2025).