**Введение**

Уважаемая комиссия, представляю вашему вниманию дипломный проект – настольное приложение, разработанное в течение одного месяца в свободное от работы и семьи время. Тематика приложения могла различаться (управление складскими запасами, задачи сотрудников, автоматизация процессов на фабрике, магазин сладостей или университетские процессы), однако во всех случаях ключевой упор сделан на **разработку backend-части** системы. Цель проекта – на практике применить и укрепить навыки backend-разработки: спроектировать надежную базу данных, реализовать бизнес-логику на стороне сервера приложения и обеспечить устойчивую работу системы в целом. Такой подход позволил мне не только создать работающее приложение для выбранной предметной области, но и глубже понять принципы построения серверной части приложений.

**Выбор технологий и инструментов**

Для реализации проекта был использован следующий стек технологий:

* **PostgreSQL** – реляционная СУБД, выбранная из-за ее надежности и соответствия принципам ACID. PostgreSQL известна поддержкой сложных транзакций и строгим соблюдением ACID (Atomicity, Consistency, Isolation, Durability – атомарность, согласованность, изоляция и долговечность), что гарантирует целостность данных при любых сбоях. Мы создали в PostgreSQL все необходимые таблицы с продуманными схемами и связями, настроили индексы для ускорения важных запросов, реализовали хранимые функции и триггеры для автоматизации операций на уровне базы. Использование PostgreSQL позволило обеспечить стабильную работу с данными и удовлетворить требования надежности и консистентности, критичные для любого приложения – будь то складской учет или управление задачами.
* **PyQt6** – фреймворк для создания пользовательского интерфейса. Хотя дизайн интерфейса не был главным фокусом работы, PyQt6 предоставил быстрый способ разработать настольное приложение с оконным GUI, чтобы конечный пользователь мог взаимодействовать с системой. В проекте через интерфейс PyQt реализованы формы и окна для ввода и вывода данных (например, добавление новых записей, просмотр отчетов и т.д.). Таким образом, PyQt6 обеспечил необходимую визуальную оболочку для демонстрации функциональности backend-части.
* **Psycopg2** – драйвер PostgreSQL для Python. Он использован для подключения приложения к базе данных и выполнения SQL-запросов из кода Python. С помощью psycopg2 реализованы все операции чтения/записи: отправка SELECT, INSERT, UPDATE, DELETE запросов, вызов хранимых функций, управление транзакциями (BEGIN/COMMIT/ROLLBACK). Этот низкоуровневый подход (вместо ORM) позволил глубже понять работу с SQL и вручную контролировать оптимизацию запросов и транзакционную логику.
* **Модуль логирования (logging)** – для ведения журнала работы приложения. Я настроил системное логирование, благодаря чему все ключевые события и ошибки записываются в лог-файлы. В проекте применяются различные уровни логов (DEBUG, INFO, WARNING, ERROR), что позволяет отслеживать как обычный ход выполнения (например, успешное подключение к БД, запуск определенного модуля), так и исключительные ситуации. Логирование оказалось незаменимым при отладке: анализируя лог-файлы, можно было быстро найти причину сбоев или неправильных данных.
* **Исключения (exceptions)** – особое внимание уделено обработке ошибок. В коде реализована корректная обработка как системных исключений, так и собственных (кастомных) исключений. Например, предусмотрены try/except блоки при работе с базой данных (для отлавливания ошибок соединения или SQL-ошибок) и при работе с файлами. В случае непредвиденной ситуации приложение не «падает» бесследно: ошибка перехватывается, логируется с подробным traceback, пользователю выдаётся понятное сообщение о сбое. Также были созданы кастомные классы исключений для специфических ситуаций предметной области – это позволило более точно классифицировать ошибки и обрабатывать их индивидуально. Такой подход значительно повышает устойчивость приложения и облегчает поддержку кода.
* **Генерация документации (Sphinx)** – для оформления проекта применялась автоматическая генерация документации. По мере разработки я заполнял docstring-документированные комментарии в коде, описывая назначение модулей, классов и функций. С помощью Sphinx на основе этих docstring была сгенерирована полноценная документация проекта. Это дало ценную практику документирования своего кода и позволило получить удобный справочный материал, упрощающий передачу проекта другим разработчикам или поддержку в будущем.
* **Интеграция с Excel/PDF, графики** – в проект встроены инструменты для формирования отчетности. Использованы библиотеки для экспорта данных в Excel и PDF, а также библиотека **matplotlib** для построения графиков. Например, можно сформировать отчет о результатах работы: выгрузить таблицу данных в файл Excel или сгенерировать PDF-документ со сводной информацией. С помощью matplotlib строятся наглядные графики (диаграммы) – скажем, динамика остатков на складе или статистика выполненных задач, – которые затем включаются в отчеты или отображаются в интерфейсе. Реализация экспорта и графиков продемонстрировала умение интегрировать сторонние библиотеки и дала опыт в создании пользовательских отчетов, что часто требуется в реальных корпоративных приложениях.
* **Методология разработки (MVP)** – при разработке соблюдался принцип *Minimum Viable Product* (минимально жизнеспособного продукта). В сжатые сроки (4 недели) и при ограниченном времени важно было сначала реализовать базовый функционал, который решает главные задачи, а затем постепенно его расширять. Мы наметили минимальный набор функций, необходимый для рабочей версии приложения, и сконцентрировались на их реализации в первую очередь. Такой подход помог вовремя получить работающее решение, проверить основные гипотезы и затем уже браться за улучшения и дополнительные возможности.
* **Структура проекта** – код организован в понятную файловую структуру. Проект разделен на модули: отдельно вынесены компоненты работы с базой данных, отдельные модули отвечают за бизнес-логику, отдельная директория – за графический интерфейс, а общие утилиты и настройки вынесены в вспомогательные модули. Такой подход к структурированию кода упрощает навигацию по проекту: каждому компоненту соответствует свой файл или пакет, ясно разделены зоны ответственности. Кроме того, подобная организация кода соответствует промышленным стандартам, что важно для дальнейшей поддержки и масштабирования приложения.

**Реализация и технические аспекты**

Главная сложность и одновременно самая интересная часть проекта – это разработка серверной логики (backend). Хочу отметить несколько реализованных технических практик и решений, которые стали ключевыми в моем приложении:

* **Архитектура приложения**: хотя приложение является настольным, оно построено по принципу клиент-серверного взаимодействия. Графический интерфейс (клиентская часть на PyQt6) отправляет запросы и команды, которые обрабатываются логикой на Python, взаимодействующей с базой данных (серверная часть). Такая многослойная архитектура с разделением на UI, бизнес-логику и уровень работы с данными обеспечила гибкость и масштабируемость. Например, при необходимости интерфейс можно заменить веб-приложением без кардинального изменения логики работы с БД. Я фактически выступил в роли backend-разработчика, создавая прослойку между пользовательским вводом и базой данных, гарантирующую корректность и безопасность операций.
* **Работа с базой данных и транзакции**: взаимодействие с PostgreSQL организовано с соблюдением транзакционной логики. Все группы операций, которые должны выполняться атомарно, обёрнуты в транзакции (BEGIN ... COMMIT). Это означает, что если, скажем, в процессе выполнения последовательности SQL-запросов что-то пошло не так (например, один из запросов не выполнился), то все изменения откатываются, и база возвращается в исходное состояние. Таким образом соблюдается принцип атомарности. Помимо атомарности, обеспечена **согласованность** данных – перед проведением транзакций проверяются необходимые условия (бизнес-правила), чтобы не нарушить целостность базы. Использовался уровень изоляции транзакций по умолчанию PostgreSQL (Read Committed), чего было достаточно для данного проекта, чтобы параллельные операции не мешали друг другу. Наконец, свойство **долговечности** обеспечивается самим PostgreSQL – после коммита данные надежно сохраняются на диске. На практике работа с транзакциями дала понять, как важны эти принципы: например, при одновременном обновлении связанных таблиц (как в случае списания товара со склада и одновременного обновления журнала операций) транзакционный подход предотвратил расхождение данных между таблицами.
* **Оптимизация запросов**: при проектировании схемы БД были использованы индексы там, где ожидались частые выборки по определенным полям. Например, для таблиц сотрудников индексировалось поле отдела, чтобы быстро фильтровать задачи по отделам; в случае склада – индекс по коду товара для мгновенного поиска остатков. Индексы значительно ускорили работу приложения при больших объемах данных. Я убедился, насколько важно уметь анализировать запросы и понимать, где узкие места: инструменты EXPLAIN и простое профилирование помогли определить, какие запросы требуют оптимизации.
* **Хранимые функции и триггеры**: часть логики была перенесена непосредственно в базу данных с помощью хранимых процедур (FUNCTION) и триггеров. Например, в базе может быть функция перерасчета итогов по заказу или триггер, который автоматически создает запись в журнале изменений при обновлении данных. Этот механизм позволил разгрузить клиентский код и обеспечить выполнение бизнес-правил на уровне БД, что повышает надежность – правила работают даже если к базе обратиться вне приложения. Разработка функций и триггеров дала мне опыт в серверном программировании внутри СУБД и научила грамотно распределять ответственность между приложением и базой.
* **Объектно-ориентированное программирование**: в коде приложения активно использованы классы для представления основных сущностей и логики. Были разработаны классы-модели, например, можно выделить класс для работы с базой данных (объект, отвечающий за соединение, запросы и транзакции), класс для бизнес-объекта (скажем, товар на складе или задача сотрудника, со своими методами), а также классы для окон интерфейса. Применение ООП позволило сделать код более модульным и читаемым: данные и методы, относящиеся к одной сущности, инкапсулированы в соответствующем классе. Кроме того, мы использовали наследование и композицию там, где это упрощало решение – например, общий абстрактный класс для различных типов справочников, от которого унаследованы конкретные реализации. Такой подход продемонстрировал понимание принципов SOLID на базовом уровне и заложил фундамент для легкого расширения функционала в будущем, поскольку добавить новую сущность или модуль можно, создав новый класс по образцу уже существующих.
* **Лямбда-функции**: хотя большая часть логики реализована в методах классов, иногда для лаконичности использовались лямбда-выражения Python. В основном это имело место при обработке коллекций данных или настройке сигналов в интерфейсе. К примеру, при сортировке списка задач по определенному ключу применена функция sorted с лямбда-ключом. В PyQt6 также удобно использовать лямбда-функции для связывания сигналов GUI с определенными действиями, особенно когда нужно передать дополнительные параметры в слот. Это позволило упростить код, избежав лишних одноразовых функций, и познакомило с функциональным стилем в Python в разумных пределах.
* **Логирование и отладка**: как упоминалось, система логирования фиксирует все ошибки и важные события. Это дало мне возможность протестировать приложение вручную и отследить все проблемы. Поскольку автоматическое тестирование пока не было внедрено, я проводил ручное тестирование, проверяя каждый сценарий работы. Логи помогали понять, что именно произошло перед ошибкой. Например, если при выполнении определенной операции возникало исключение, в логе видно стек вызовов – где и почему проблема. Я старался проактивно обрабатывать возможные ошибки: проверял корректность входных данных, существование необходимых файлов, доступность сервера БД. Такой подход соответствует лучшим практикам backend-разработки – важно не только писать функциональный код, но и предусмотреть его поведение в случае некорректных ситуаций. В результате приложение получилось довольно устойчивым: при ошибках оно сообщает о проблеме, пишет информацию для разработчика и продолжает работу или корректно завершает транзакцию, не оставляя систему в подвешенном состоянии.

**Полученный опыт и значимость проекта**

За время работы над проектом я приобрел ценный практический опыт во многих аспектах backend-разработки. В теории многие из этих вещей были знакомы, но именно практика позволила прочувствовать их значение. В частности:

* Я значительно углубился в **работу с СУБД PostgreSQL**: спроектировал структуру базы с нуля, настроил ограничения (первичные и внешние ключи, уникальность), изучил на практике, как с помощью индексов и оптимизации запросов можно добиться заметного прироста производительности. Разобрался, как реализуются транзакции и как ACID-принципы выглядят в реальном приложении.
* Получил навыки интеграции Python-кода с базой данных через **psycopg2**, включая обработку соединений, параметризацию SQL-запросов (для защиты от SQL-инъекций) и обработку результатов. Теперь у меня есть понимание, как backend-приложение общается с базой, как обрабатывать курсоры, управлять транзакциями из кода приложения.
* Отточил умения в **обработке исключений и логировании**. Я увидел, как важна правильная обработка ошибок: без этих механизмов приложение было бы ненадежным и падало при любой мелочи. Теперь я уверенно использую конструкцию try/except, умею создавать свои исключения под нужды логики и понимаю, как вести журнала логов для мониторинга приложения. Эти навыки напрямую относятся к культуре промышленной разработки, где устойчивость сервиса – один из приоритетов.
* Научился создавать **документацию** к коду параллельно с его написанием. Sphinx показал себя удобным инструментом, а главное – я прочувствовал, насколько упрощается поддержка проекта, когда у каждого модуля есть описание, а у каждой функции – понятный комментарий о предназначении. В дальнейшем, если проект будет дорабатываться или передаваться другим разработчикам, такая документация сэкономит много времени.
* Приобрел опыт работы с **внешними библиотеками** для экспорта данных и визуализации. Интеграция генерации отчетов (Excel/PDF) и графиков дала понимание, как соединять backend-логику с созданием удобных выходных форм для конечного пользователя. Это ценный навык, потому что практически во всех реальных проектах данные нужно не только хранить, но и представлять в понятном виде.
* Улучшил понимание **структуры и архитектуры приложений**. Применение принципов модульности, ООП, проектирование файловой структуры – все это научило лучше организовывать код. Теперь, начиная новый проект, я смогу сразу разложить его по компонентам, что облегчит и разработку, и дальнейшее масштабирование. Я также прочувствовал ценность разработки через MVP: это уберегло от переработки и позволило увидеть результаты быстро, мотивируя к дальнейшим улучшениям.

Самое важное – проект помог связать воедино разные знания по backend-разработке и применить их на практике. Если ранее принципы ACID, транзакции, индексы, или, например, логирование были просто теорией из учебников, то теперь это мой личный опыт. Такой опыт крайне важен для становления как backend-разработчика: я лучше понимаю, как «под капотом» работает серверная часть приложений, как обеспечить надежность и эффективность системы. Этот фундамент позволит мне уверенно двигаться дальше в изучении более сложных технологий и подходов.

**Дальнейшее развитие проекта**

Проект получился работоспособным и уже решает поставленные задачи, однако я вижу множество путей для его развития, которые планирую реализовать в будущем:

* **Автоматическое тестирование**: следующий шаг – внедрение модульных и интеграционных тестов (например, с использованием unittest или pytest). Это позволит автоматизировать проверку работоспособности всех компонентов и предотвратить регрессии при внесении изменений. Пока что тестирование осуществлялось вручную, но для промышленного уровня качества необходимо покрыть ключевую логику автотестами.
* **Кэширование и Redis**: для повышения производительности я рассматриваю подключение системы кэширования. Изучение Redis (ин-memory хранилища) позволит хранить часто запрашиваемые данные в памяти и тем самым снизить нагрузку на базу PostgreSQL и ускорить ответы приложения. Например, в системе управления складом можно кэшировать справочники товаров или часто запрашиваемые отчеты, чтобы при повторном запросе не обращаться к базе.
* **Развертывание сервера**: сейчас приложение десктопное и работает локально, но я планирую перевести его на клиент-серверную архитектуру в полноценно-сетевом варианте. Это может быть реализация отдельного серверного приложения (например, REST API на Flask/FastAPI или другой backend-фреймворк), которое будет развернуто на удаленном сервере. Тогда клиентские приложения (возможно, даже веб-клиенты или мобильные) смогут подключаться к общему серверу. Такой переход потребует знаний деплоя на сервер, настройки среды (виртуальный сервер или облако) и обеспечит доступ к системе из любой точки, а не только с локального компьютера.
* **Вертикальное масштабирование и оптимизация**: по мере роста данных и нагрузки нужно быть готовым масштабировать систему. Вертикальное масштабирование подразумевает увеличение ресурсов сервера базы данных (мощности CPU, RAM, быстродействия дисков) по мере необходимости. Я планирую изучить инструменты мониторинга производительности PostgreSQL, чтобы знать, когда требуются дополнительные ресурсы или оптимизация запросов. Также можно будет рассмотреть и **горизонтальное масштабирование** (распределение нагрузки между несколькими узлами), хотя для настольного приложения это пока избыточно.
* **Резервное копирование БД**: важнейший аспект надежности – регулярное резервирование данных. В дальнейшем я настрою автоматическое резервное копирование базы данных, чтобы обезопасить информацию от потери. Это включает как создание бэкапов по расписанию, так и, возможно, репликацию базы на запасной сервер для отказоустойчивости. Реализуя это, я получу опыт и в области **администрирования баз данных**, что полезно для backend-инженера.
* **Интеграция с внешними сервисами (email, заявки)**: чтобы расширить функциональность приложения, планируется добавить взаимодействие с внешними системами. Например, **интеграция email** позволит автоматически отправлять письма – уведомления или отчеты – пользователям или администраторам. Внедрение работы с электронными почтовыми протоколами (SMTP) и формирование писем прямо из приложения повысит степень автоматизации. Также предполагается реализация модуля обработки заявок: например, для приложения университета это могут быть заявки студентов на курсы, для склада – заявки на закупку товаров. Позволив системе принимать и обрабатывать такие запросы, мы выходим за рамки чисто локальной программы и переходим к более комплексной информационной системе.
* **Использование технологий ИИ**: современным направлением, которое хочется освоить, является внедрение элементов искусственного интеллекта для автоматизации задач. В перспективе я планирую экспериментировать с тем, чтобы подключить к приложению интеллектуальные алгоритмы. Это может проявиться по-разному в зависимости от тематики: например, прогнозирование потребностей склада с помощью модели машинного обучения, интеллектуальный ассистент для сотрудников (чат-бот по типу GPT, помогающий находить информацию или генерировать ответы), либо система рекомендаций. Интеграция ИИ увеличит уровень автоматизации и «умности» системы, а для меня станет следующей ступенькой в профессиональном развитии.

Каждое из этих направлений развития не только сделает проект богаче функционально, но и даст мне возможность расширить компетенции – в тестировании, администрировании, масштабировании и современных технологиях backend-разработки.

**Заключение**

Подводя итоги, хочу подчеркнуть, что работа над этим дипломным проектом позволила мне закрепить ключевые знания и навыки backend-разработчика. От проектирования базы данных и эффективного кода бизнес-логики до обеспечения надежности через обработку ошибок и логирование – каждый аспект внес свой вклад в конечный результат. Проект не ограничился теорией, это был интенсивный практический опыт: сталкиваясь с реальными задачами и проблемами, я научился применять правильные подходы и решения.

Backend-разработка – это фундамент любой сложной системы, и теперь, пройдя через полный цикл создания серверной части приложения, я гораздо лучше понимаю этот фундамент. Уверен, что опыт, полученный в ходе данного проекта, послужит прочной базой для моей дальнейшей карьеры. В будущем я смогу смелее браться за сложные проекты, зная, как реализовать их ядро, обеспечить целостность данных, масштабируемость и интеграцию с различными технологиями.

Спасибо за внимание! Я готов ответить на ваши вопросы.