# Проектная документация: «Карго Менеджер Лисёнок»

## 1. Подробное техническое задание (ТЗ)

### Ключевые функции системы

* **Парсинг товаров с 1688:** Автоматический сбор данных о товаре по ссылке с платформы 1688.com. Система извлекает название, описание, фотографии, характеристики, цену и минимальную партию товара. Дополнительно парсер собирает информацию о поставщике (рейтинг, отзывы, время работы на рынке) и автоматически конвертирует цену из юаней в рубли по актуальному курсу. Извлечённые данные могут использоваться для быстрого создания заказа или коммерческого предложения.
* **Управление заказами:** Ведение базы заявок на грузоперевозки. В каждой заявке хранится информация о клиенте, поставщике, описании груза, весе, объёме, таможенном коде (ТН ВЭД), условиях Инкотермс, датах отправки и прибытия, статусе доставки и общей стоимости заказа[[1]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,REAL%20NOT%20NULL%2C%20hs_code%20TEXT)[[2]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=self.root.title%28%22Cargo%20Manager%20%D0%9B%D0%B8%D1%81%D1%91%D0%BD%D0%BE%D0%BA%20,1000x700). Менеджер может создавать новые заявки (вручную или на основе парсинга), редактировать их, изменять статус (например, «Новый», «В обработке», «В пути», «Доставлен» и т.д.), а также просматривать список всех заказов с возможностью фильтрации по статусу или клиенту.
* **Документооборот:** Генерация и хранение документов по заявкам. Система должна автоматически формировать **коммерческие предложения**, **контракты**, **инвойсы**, **таможенные декларации**, **накладные** и другие документы на основе данных заказа. Реализовано автозаполнение шаблонов документов данными из заявки (чтобы исключить дублирование ввода)[[3]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%A5%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8%20%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9%20%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%B2). Предусмотрена проверка корректности заполнения документов (например, все ли обязательные поля заполнены) и возможность электронной подписи документов. Система хранит историю версий документов, чтобы можно было отследить изменения и при необходимости восстановить предыдущую версию[[3]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%A5%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8%20%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9%20%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%B2).
* **Таможенный калькулятор:** Модуль расчёта таможенных платежей и логистических расходов. Позволяет автоматически определять код ТН ВЭД по описанию товара (при помощи встроенной базы знаний или алгоритмов поиска ключевых слов) и рассчитывать ориентировочную пошлину, НДС и другие налоги для ввоза товара[[4]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%20%D1%83%D0%BF%D1%80%D0%BE%D1%89%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BE%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F). Также калькулятор вычисляет параметры доставки – например, тарифный вес (с учётом объёма и фактического веса), стоимость транспортировки разными способами (авиа, Ж/Д, море) и сроки доставки. Менеджер может сравнить варианты доставки: система покажет разницу в цене и сроках и может дать рекомендацию (например, «Экономия времени: 15 дней, переплата: 93 300 ₽» при переходе с Ж/Д на авиа-доставку[[5]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%96%2F%D0%94)). Модуль интегрирован с базой знаний, содержащей актуальные ставки пошлин, списки запрещённых к ввозу товаров и рекомендации по оформлению сложных категорий грузов[[6]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%A0%D0%B5%D0%BA%D0%BE%D0%BC%D0%B5%D0%BD%D0%B4%D0%B0%D1%86%D0%B8%D0%B8%20%D0%BF%D0%BE%20%D0%BE%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D1%8E%20%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D1%8B%D1%85%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B9).
* **Отслеживание логистики:** Функционал мониторинга перемещения груза. Интеграция с API транспортных и курьерских компаний (например, Cainiao, SF Express, DHL и др.) для автоматического получения статуса посылки по трекинг-номеру[[7]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%92%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20%D0%BC%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B0%20%D0%BD%D0%B0%20%D0%BA%D0%B0%D1%80%D1%82%D0%B5). Система отображает текущий статус доставки (в пути, на таможне, прибыло в страну назначения, на складе и т.д.) и визуализирует маршрут на карте. При изменении статуса система генерирует уведомление для менеджера, а также может отправлять оповещение клиенту через Telegram. Предусмотрены прогнозирование возможных задержек и сравнение плановых сроков доставки с фактическими[[8]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%20%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D1%85%20%D0%B8%20%D0%BF%D0%BB%D0%B0%D0%BD%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D1%85%20%D1%81%D1%80%D0%BE%D0%BA%D0%BE%D0%B2).
* **Интеграция с мессенджерами (Communication Hub):** Единый коммуникационный модуль, связывающий систему с внешними мессенджерами для работы с клиентами и поставщиками. В частности, поддерживаются:
* **Telegram (для клиентов):** Используется Telegram Bot API для взаимодействия с клиентами[[9]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%2A%2A%D0%9A%D0%B0%D0%BA%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%B5%D1%82%3A%2A%2A%20,%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%B0%D0%BC%D0%B8%20%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B8%D1%80%D1%83%D1%8E%D1%82%D1%81%D1%8F%20%D1%81%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B9%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Система создаёт собственного бота, через которого происходит общение. Все сообщения от клиентов через бота автоматически связываются с соответствующими заказами (по предварительной привязке клиента к заявке)[[10]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=3,%D0%BC%D0%B3%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%20%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B8%D1%82%D1%8C%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8E%20%D1%83%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%B0). Менеджер видит и отвечает на сообщения непосредственно в интерфейсе приложения – отпадает необходимость переключаться в само приложение Telegram[[11]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=4.%20%D0%9F%D1%80%D0%BE%D1%84%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%B8%D0%BC%D0%B8%D0%B4%D0%B6%20,%D0%9B%D0%B8%D1%81%D1%91%D0%BD%D0%BE%D0%BA%20%D0%B2%D1%81%D0%B5%D0%B3%D0%B4%D0%B0%20%D0%B2%20%D0%BA%D1%83%D1%80%D1%81%D0%B5%20%D0%B4%D0%B5%D0%BB). Реализованы шаблоны сообщений для типовых ситуаций (например, подтверждение получения запроса, уведомление об отправке груза, сообщении о прибытии). Система присылает уведомления о новых сообщениях с высоким приоритетом, чтобы менеджер не пропустил важный запрос. Также через бота клиенту могут автоматически отправляться документы (PDF коммерческого предложения, счёт и пр.) и уведомления о статусе его заказа.
* **WeChat (для поставщиков):** Интеграция с WeChat осуществляется через официальное API WeChat Work либо посредством настольного приложения WeChat (в случае использования личного аккаунта поставщика). Предусмотрен специальный механизм подключения: менеджер сканирует QR-код в меню «Подключение WeChat», после чего система связывается с его WeChat Desktop для мониторинга необходимых чатов[[12]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%B8%D0%B9%202%3A%20%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0%20WeChat%20%D0%B4%D0%BB%D1%8F,%D0%BE%D1%82%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%BE%D0%B2%20%D0%BF%D0%BE%D1%8F%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F%20%D0%B2%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B8). Все входящие сообщения от китайских поставщиков отображаются в разделе сообщений приложения, и менеджер может на них сразу отвечать через интерфейс системы[[13]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%BC%D0%B3%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%20%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B8%D1%82%D1%8C%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8E%20%D1%83%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%B0). Система сохраняет контент переписки в базу данных: текст, изображения, файлы – и привязывает их к карточке соответствующего поставщика или заявки. Например, если поставщик прислал фотографию товара в WeChat, менеджер может нажать «Вставить фото» в карточке товара, и система сохранит изображение, привяжет его к заказу и отметит источник (например, «Фото от Shenzhen Electronics через WeChat»)[[14]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1,%D0%B1%D1%8B%D1%81%D1%82%D1%80%D0%BE%D0%B3%D0%BE%20%D0%B4%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)[[15]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,WeChat%20%D0%BE%D1%82%20Shenzhen%20Electronics). Благодаря интеграции, все коммуникации хранятся централизованно, а у менеджера создаётся профессиональный имидж – больше не нужно переключаться между разными приложениями, всё происходит в одной системе[[11]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=4.%20%D0%9F%D1%80%D0%BE%D1%84%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%B8%D0%BC%D0%B8%D0%B4%D0%B6%20,%D0%9B%D0%B8%D1%81%D1%91%D0%BD%D0%BE%D0%BA%20%D0%B2%D1%81%D0%B5%D0%B3%D0%B4%D0%B0%20%D0%B2%20%D0%BA%D1%83%D1%80%D1%81%D0%B5%20%D0%B4%D0%B5%D0%BB).
* **База знаний и справочники:** Вспомогательный модуль, хранящий нормативно-справочную информацию, необходимую для работы. В него входят: список актуальных курсов валют (обновляемых в реальном времени через внешний API), таблицы тарифов перевозчиков (для расчёта логистики), базы данных кодов ТН ВЭД с описаниями, ставки таможенных пошлин, перечень запрещённых к ввозу товаров, шаблоны сообщений и документов, типовые справочники (например, список доступных статусов заказа, типов доставки, терминов Инкотермс и т.д.). **Курсы валют** обновляются автоматически по расписанию (например, раз в день или при значительном колебании) посредством интеграции с внешним сервисом (например, API Центрального банка или другой финансовой API). При обновлении курсов система пересчитывает все открытые заявки в фоновом режиме и уведомляет менеджера, если изменение курса существенно влияет на стоимость по текущим просчётам[[16]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1,%E2%86%92%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0%20%D1%88%D0%BB%D0%B5%D1%82%20%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F). Менеджер может в один клик разослать уведомления клиентам о перерасчёте цен из-за изменения курса[[16]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1,%E2%86%92%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0%20%D1%88%D0%BB%D0%B5%D1%82%20%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F) – это исключает ручной труд по пересчёту в Excel. База знаний обеспечивает единообразие данных во всех модулях системы.

### Роли пользователей и права доступа

* **Администратор:** Полный доступ к системе. Администратор управляет учетными записями пользователей, назначает роли, настраивает справочники (добавляет новых перевозчиков, обновляет ставки, курсы валют, шаблоны документов), а также отвечает за общие настройки системы и интеграций. Имеет право просматривать и изменять любые данные, выполнять резервное копирование и восстановление базы. Следит за журналом действий пользователей и общей безопасностью.
* **Карго-менеджер:** Основной пользователь приложения (роль Лисёнка). Имеет доступ ко всем рабочим модулям: может создавать и редактировать заказы, пользоваться парсером 1688, калькулятором, генерировать документы, общаться с клиентами и поставщиками через интегрированные мессенджеры. Карго-менеджер видит только связанные с его работой данные (в небольшом бизнесе менеджер может быть один и видеть всё; если система внедряется в более крупной компании – может потребоваться разграничение по филиалам или своим клиентам). Карго-менеджер не управляет пользователями системы, но может добавлять новых клиентов и поставщиков в соответствующие справочники.
* **Клиент:** В контексте системы клиент – это контрагент, которому оказываются услуги доставки. В текущей реализации клиенты не имеют прямого доступа к интерфейсу системы, однако на них заведены учетные карточки (название компании, контактные лица, реквизиты, история заказов). Коммуникация с клиентом происходит через Telegram или телефон, но в перспективе возможен **личный кабинет** клиента: ограниченный веб-интерфейс или мобильное приложение, где клиент сможет отслеживать статус своего груза, просматривать и загружать документы, оплачивать счета и задавать вопросы менеджеру. В рамках данной ТЗ роль клиента нужна для структуры данных и генерации персонализированных уведомлений.
* **Поставщик:** Аналогично клиенту, поставщики (китайские фабрики или торговые компании) хранятся в системе как сущности с контактной информацией, реквизитами и историей взаимодействий. Прямого доступа к системе у них нет, связь идёт через WeChat. Роль поставщика используется системой для автоматического определения входящих сообщений (если сообщение пришло через WeChat, система помечает его как сообщение от конкретного поставщика, идентифицированного по аккаунту)[[13]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%BC%D0%B3%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%20%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B8%D1%82%D1%8C%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8E%20%D1%83%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%B0). В будущем возможно создание **портала для поставщиков** (например, для загрузки документов или подтверждения отгрузок), но это вне первичного объёма работ.

### Пользовательские интерфейсы

Приложение ориентировано на настольное применение (Desktop/Web), с единым окном, разделённым на функциональные разделы. Основные интерфейсные компоненты:

* **Главное окно с навигацией:** После входа в систему (с вводом логина/пароля и, при включённой опции, подтверждением 2FA) пользователь попадает на основной экран – **рабочий стол карго-менеджера**. Интерфейс спроектирован как набор вкладок (в веб-версии – разделов меню), между которыми можно быстро переключаться без потери контекста. Главные разделы: **Заказы**, **Калькулятор**, **Документы**, **Отслеживание**, **Справочники**, а также **Сообщения**:
* **Раздел «Заказы»:** здесь отображается реестр всех заявок (табличным списком) и форма для создания нового заказа. В таблице заказов отображаются ключевые поля: № заявки, клиент, поставщик, описание груза, вес, статус[[17]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,headings). Предусмотрена строка поиска/фильтра по клиенту, поставщику или номеру заявки, а также фильтр по статусу. При выборе конкретной заявки можно просмотреть детальную информацию: открывается карточка заказа (либо во всплывающем окне, либо на отдельной вкладке) с полной информацией, включая историю изменений статусов и переписку. В этой же вкладке менеджер может редактировать поля заявки, добавлять примечания. Для создания нового заказа – форма ввода данных: поля «Поставщик», «Клиент», «Описание груза», «Вес», «Объем», «Код ТН ВЭД», «Инкотермс», «Дата отправки», «Планируемая дата прибытия» и выпадающий список **статуса**[[18]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,7)[[19]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,w). Часть полей может заполняться автоматически (например, текущая дата для поля даты отправки подставляется по умолчанию[[20]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,d)). После ввода данных менеджер сохраняет заказ, и он появляется в списке. К каждому заказу привязано контекстное меню или кнопки действий: «Просмотреть/Редактировать», «Изменить статус», «Сгенерировать документы», «Отследить груз» и т.д. Эти действия либо переводят пользователя в соответствующие разделы, фильтруя данные по данной заявке, либо открывают диалоги.
* **Раздел «Калькулятор»:** интерактивный модуль для расчётов. Здесь менеджер может выполнить таможенные и логистические расчёты как для конкретной заявки (данные могут подтягиваться автоматически из выбранного заказа), так и произвести прикидочные расчёты для нового запроса. Интерфейс калькулятора включает поля для ввода таможенной стоимости, объёмного и фактического веса, выбора кода ТН ВЭД (с автоподстановкой по базе), выбора маршрута и способа доставки. После ввода данных система отображает рассчитанные пошлины, НДС, таможенные сборы, а также стоимость доставки каждым доступным способом. Если в справочниках присутствуют тарифы нескольких перевозчиков или маршрутов, калькулятор может отображать несколько вариантов с указанием цены и срока. Менеджер может сразу выбрать оптимальный вариант, и тогда эти данные прикрепятся к соответствующей заявке. Например, если в заявке изменить маршрут доставки, система пересчитает стоимость и время и мгновенно отобразит разницу прямо в интерфейсе (как показано во **Примере 3**: при смене Ж/Д на Авиа стоимость изменилась с 85 200 ₽ до 178 500 ₽, срок сократился на 15 дней, и система отображает эту разницу на экране)[[5]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%96%2F%D0%94).
* **Раздел «Документы»:** библиотека документов и шаблонов. Здесь хранятся все сгенерированные по заявкам документы. Интерфейс позволяет выбрать заявку и увидеть список документов (коммерческое предложение, счёт, упаковочный лист, и т.д.). Реализованы кнопки: **«Создать новый документ»** (на основе шаблона и данных заявки), **«Экспорт всех документов»** (например, скачивание архива или пакетная выгрузка) и **«Отправить клиенту»** (отправка выбранного документа клиенту по интегрированному каналу связи, например, вложением в чат Telegram)[[21]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%F0%9F%8C%90%20%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%B8%20%D0%B8%20%D0%B2%D0%BD%D0%B5%D1%88%D0%BD%D0%B8%D0%B5%20%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D1%8B)[[22]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%2A%2A%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8%3A%2A%2A%20,%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%B0%20%D0%BD%D0%B0%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F%20%D1%87%D0%B5%D1%80%D0%B5%D0%B7%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Документ открывается для предварительного просмотра прямо в системе (например, встроенный PDF-просмотрщик) перед отправкой. Также можно отредактировать некоторые поля шаблона перед финальной генерацией (например, особые условия в контракте). Версионность: при каждом изменении документа старая версия сохраняется, и ее можно просмотреть или восстановить при необходимости.
* **Раздел «Отслеживание»:** здесь отображается информация о доставке для текущих заказов. Менеджер может ввести или сканировать трек-номер отправления и привязать его к заявке. Если настроены API-интеграции с перевозчиками, система автоматически подтянет статус отправления и отобразит его. В таблице отслеживания видны: трек-номер, связанная заявка, ответственный перевозчик, текущий статус, дата последнего обновления, ожидаемая дата доставки. Нажав на конкретное отправление, менеджер получает подробности – историю перемещений (этапы с датами и местоположением). Если включена карта, можно увидеть текущую позицию или маршрут груза. В случае задержек или отклонений от графика система может подсветить проблемные отправления и выдать прогноз (например, «Задержка на таможне, ожидаемое прибытие +2 дня»). Из этого же раздела доступно изменение статуса заказа вручную, если необходимо (например, пометить заказ как «Доставлен» после получения подтверждения).
* **Раздел «Сообщения» (Коммуникации):** единый центр общения, интегрированный с Telegram и WeChat. Интерфейс схож с мессенджером: слева список диалогов (контрагентов), справа – текущий выбранный чат. Диалоги с клиентами и поставщиками помечены иконкой соответствующего канала (например, Telegram или WeChat). Каждый диалог, в идеале, привязан к клиенту/поставщику и к конкретной заявке (система может группировать сообщения по заявкам или по контактам). Вверху окна чата отображается контрагент и, если возможно, ссылка на связанную заявку. Менеджер может печатать сообщения, отправлять файлы или фотографии прямо из приложения – они уйдут клиенту через соответствующий бот или поставщику через WeChat API. Когда приходят новые сообщения, система показывает всплывающее уведомление. В настройках подключений (раздел «Подключения» или «Настройки интеграций») менеджер заранее настраивает бота Telegram (вводит токен)[[23]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%B8%D0%B9%201%3A%20%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0%20Telegram%20%D0%B4%D0%BB%D1%8F,%D0%BE%D1%82%20%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%BE%D0%B2%20%D0%BF%D0%BE%D1%8F%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F%20%D0%B2%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B8) и привязывает WeChat (сканирует QR для подключения)[[12]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%B8%D0%B9%202%3A%20%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0%20WeChat%20%D0%B4%D0%BB%D1%8F,%D0%BE%D1%82%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%BE%D0%B2%20%D0%BF%D0%BE%D1%8F%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F%20%D0%B2%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B8). После настройки все соответствующие сообщения начинают стекаться в «Сообщения». Таким образом, **Лисёнок видит все коммуникации в одном окне и может сразу реагировать**, не переключаясь между приложениями[[11]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=4.%20%D0%9F%D1%80%D0%BE%D1%84%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%B8%D0%BC%D0%B8%D0%B4%D0%B6%20,%D0%9B%D0%B8%D1%81%D1%91%D0%BD%D0%BE%D0%BA%20%D0%B2%D1%81%D0%B5%D0%B3%D0%B4%D0%B0%20%D0%B2%20%D0%BA%D1%83%D1%80%D1%81%D0%B5%20%D0%B4%D0%B5%D0%BB). Это ускоряет работу и протоколирует всю историю взаимодействия – например, любое полученное от клиента сообщение сразу сохраняется в базе и при необходимости может быть найдено из карточки заказа.
* **Раздел «Справочники» (База знаний):** здесь менеджер (или админ) управляет вспомогательными данными. Интерфейс состоит из нескольких вкладок/подразделов: «Клиенты», «Поставщики», «Тарифы перевозчиков», «Товары/Категории» (опционально), «Курсы валют», «Шаблоны» и «Прочее». В **«Клиентах»** хранится список клиентов с реквизитами, контактами, примечаниями – оттуда же можно добавлять новых или редактировать. Аналогично для **«Поставщиков»** – список поставщиков. **«Тарифы перевозчиков»** содержит информацию от логистических компаний: тарифные сетки для разных направлений, нормативы по весу/объёму, сроки. Эти данные используются калькулятором и модулем отслеживания. **«Курсы валют»** – здесь можно просмотреть текущие курсы, историю изменений и при необходимости вручную обновить или задать фиксированный курс для расчётов (если в компании принята, скажем, фиксированная внутренняя ставка). **«Шаблоны»** – библиотека шаблонов документов и сообщений: пользователь может редактировать текст типовых писем, форматы документов, подписи. **«Прочее»** – прочие справочные таблицы: список статусов заказа (для настройки бизнес-процесса), список доступных пользователей системы с ролями (для администратора), настройки интеграций (API-ключи, токены ботов) и т.д. Вся информация справочников связана с основными модулями: например, добавив нового перевозчика и его тарифы, менеджер сразу сможет выбирать его в калькуляторе, а модуль логистики – учитывать его при отслеживании.

### Системные требования

* **Платформа и развёртывание:** Решение может быть развернуто в двух вариантах – как локальное десктоп-приложение или как веб-приложение (клиент-серверная архитектура). В случае десктоп-версии требуется современная ОС (Windows 10+, macOS 11+ или совместимая Linux-среда) и установка всех необходимых зависимостей. Приложение может быть упаковано для удобства (например, с помощью PyInstaller для Python или Electron для веб-интерфейса). Веб-версия предполагает наличие сервера (или облачного хостинга) для бэкенда и базы данных, а пользовательская часть открывается в браузере. Для клиента достаточно любого современного браузера (Chrome, Firefox, Edge и т.д.) на ПК; интерфейс адаптируется под разрешение экрана.
* **Оборудование:** Специальных мощностей не требуется – достаточно стандартного офисного компьютера. Рекомендуемая конфигурация сервера (или ПК): процессор 2+ ГГц, 8 ГБ ОЗУ (для комфортной работы, особенно если используется браузерный парсер 1688 и хранится много данных), дисковое пространство от 1-5 ГБ под базу данных и хранилище документов (в перспективе больше, в зависимости от объёма данных и сохраненных вложений). Наличие стабильного интернет-соединения **обязательно** для работы интеграций (мессенджеры, курсы валют, API отслеживания).
* **Программные компоненты:** На сервере/ПК должны быть установлены: интерпретатор Python 3.10+ (если выбран Python-бэкенд), либо Node.js 18+ (если бэкенд на Node.js), веб-сервер (например, uvicorn для FastAPI или Node/Express) и СУБД (рекомендуется PostgreSQL 14+ для промышленной эксплуатации). В десктоп-варианте все необходимые библиотеки включаются в сборку.
* **Зависимости для парсера 1688:** Необходим headless-браузер или аналог (например, Chrome/Chromium + Selenium/Playwright) для корректного парсинга динамичного контента 1688. Это потребует драйвер браузера (например, ChromeDriver) либо библиотеку Playwright. Также необходим доступ к 1688: возможно, потребуется VPN или китайский прокси, если парсинг осуществляется за пределами Китая (учитывая ограничения Alibaba).
* **Прочие требования:** Для PDF-генерации – установка библиотек (например, ReportLab для Python или PDFKit). Для работы с Telegram – наличие доступа к Telegram Bot API (достаточно просто интернета), для WeChat – установленное приложение WeChat на ПК менеджера (если интеграция через настольный клиент).
* **Мультиязычность:** Интерфейс приложения по умолчанию на русском языке. Поддержка китайского языка заключается в корректном отображении и хранении символов (например, названий товаров, адресов) – система должна использовать кодировку Unicode (UTF-8) и шрифты, поддерживающие иероглифы. Если потребуется полноценный перевод интерфейса на китайский для поставщиков, архитектура позволяет добавить языковые файлы локализации.

### Правила безопасности

* **Аутентификация и авторизация:** Реализована система пользовательских аккаунтов с разграничением прав (роли). Пароли хранятся в базе данных в виде хеша (например, алгоритм BCrypt). Поддерживается двухфакторная аутентификация (например, TOTP через приложение Google Authenticator или одноразовый код по email/SMS) для администратора и менеджеров. При аутентификации через внешние сервисы (Telegram/WeChat) используются токены и API-ключи, которые хранятся в зашифрованном виде.
* **Разграничение доступа:** Пользователь с ролью «клиент» или «поставщик» (если такие роли будут задействованы в портале) имеет доступ только к своему контенту. Менеджеры видят данные только своего отдела, если внедряется разделение. Администратор может всё. Все критичные действия (удаление записи, изменение статуса, экспорт данных) требуют подтверждения действия, чтобы избежать случайных ошибок.
* **Журнал операций:** Система ведет логирование важных действий: вход/выход пользователей, создание/удаление заказов, изменение статусов, отправка документов, изменения в справочниках. Журнал (audit log) хранит кто и когда совершил действие. Это помогает в внутренней безопасности и отладке.
* **Безопасность данных:** База данных регулярно архивируется (автоматическое резервное копирование, например, раз в день). Архивы хранятся в защищённом хранилище. Соединение веб-клиента с сервером должно быть по HTTPS с валидным сертификатом, чтобы защитить данные (особенно персональные данные клиентов и переписку) при передаче. Внутри системы конфиденциальные сведения (например, финансовая информация, паспортные данные клиентов, если хранятся) шифруются на уровне БД или приложения.
* **Интеграция мессенджеров:** Telegram-бот хранит токен доступа в настройках приложения (лучше вынести в конфиг-файл вне кода). WeChat-интеграция, если использует внешний сервис или эмуляцию – ограничивается только необходимыми правами. Все входящие сообщения проверяются на отсутствие вредоносного содержимого (например, если поставщик прислал файл, система может проверить его антивирусным движком перед сохранением).
* **Защита от сбоев:** При недоступности внешних сервисов (например, API отслеживания или курсы валют) система не должна «падать» – реализованы таймауты и повторные попытки, а в случае невозможности подключения – уведомление менеджера о том, что сервис временно недоступен.
* **Модуль обновлений:** Предусмотрено, что приложение сможет получать обновления (например, если это веб – выкладкой новой версии на сервер; если десктоп – через автообновление с сервера). При обновлении сохраняется целостность данных, а перед серьёзными изменениями структуры делается бэкап.

### Масштабируемость и резерв развития

* **Архитектура, готовая к росту:** Система спроектирована модульной – каждый крупный компонент (заказы, парсер, сообщения, и т.д.) отделён логически и технически. Это позволяет распределять нагрузку и в будущем выносить модули в отдельные сервисы при высокой нагрузке. Например, модуль парсинга 1688 может быть развёрнут как отдельный сервис (микросервис), чтобы при необходимости запускать несколько парсеров параллельно для обработки множества запросов. Аналогично, модуль интеграции с мессенджерами может быть вынесен отдельно для независимого поддержания подключения к API.
* **База данных:** Начальная реализация может использовать SQLite (это упрощает старт разработки и тестирование), но для **масштабируемости** рекомендуется переход на PostgreSQL или MySQL. Эти СУБД могут обслуживать одновременную работу нескольких пользователей, обеспечивать бо́льшую сохранность данных и лучше масштабируются на выросших объемах (можно настроить репликацию, резервные копии и т.д.). При росте числа заказов до десятков тысяч и более, используется индексация полей, оптимизация SQL-запросов, возможно, шардинг или разделение БД по модулям (например, вынести исторические данные в архивную БД).
* **Многопользовательский режим:** Если планируется, что системой будут пользоваться несколько менеджеров одновременно (например, в компании несколько логистов), веб-архитектура уже это поддерживает. Важно учесть блокировки на уровне записей (например, чтобы два менеджера случайно не редактировали одну заявку одновременно – можно реализовать механизмы блокировки или обновления в реальном времени с оповещением через WebSocket). Приложение изначально проектируется под конкурентный доступ, чтобы добавление новых пользователей не потребовало кардинальной переработки.
* **Вертикальное и горизонтальное масштабирование:** Приложение может быть развернуто в облаке. Для увеличения производительности серверной части достаточно масштабировать сервер (вертикально – увеличив ресурсы CPU/RAM, либо горизонтально – запустить несколько экземпляров за балансировщиком). Так как состояние в основном хранится в БД, несколько экземпляров бэкенда могут работать параллельно, обрабатывая разные запросы. Статические файлы и документы можно выносить на CDN или облачные хранилища для разгрузки сервера.
* **Мобильная адаптация в перспективе:** Уже на этапе проектирования заложена поддержка мобильных клиентов. Это означает, что фронтенд реализуется адаптивным (responsive) – им можно пользоваться на планшете, и базовый функционал доступен. В дальнейшем, при необходимости, можно разработать нативное мобильное приложение (например, на базе React Native или Flutter) или PWA (прогрессивное веб-приложение), которое подключается к тому же серверу через API. Таким образом, расширение на мобильные платформы не потребует изменения ядра системы – достаточно разработать новый интерфейс, используя уже существующие API для операций.
* **Добавление новых интеграций и функций:** Благодаря модульности, система может быть расширена без больших переделок. Например, при необходимости подключить новый мессенджер (WhatsApp, Viber) – добавляется новый подпроект интеграции, использующий их API, и встраивается в Communication Hub по аналогии с Telegram/WeChat. Если появятся новые требования (например, расчет складского хранения или финансовый учет), можно дополнять систему новыми модулями, взаимодействующими с ядром. Обеспечивается **слабо связанная архитектура**: модули взаимодействуют через четко определенные интерфейсы (API ядра, общая БД с ключами и пр.), что минимизирует влияние изменений в одном модуле на остальные.

## 2. Архитектура модулей системы

Система разделена на модули, каждый из которых отвечает за свой участок функциональности, при этом все связаны через центральное ядро. Ниже приведена схема взаимодействия основных компонентов:

[Парсер 1688] → [Центральное ядро] ← [Интеграция Telegram]  
 ↓ ↑ ↓  
[Управление заказами] ← [Центральное ядро] → [Документооборот]  
 ↓ ↑ ↓  
[Таможенный калькулятор] ← [Центральное ядро] → [Отслеживание груза]  
 ↓  
 [База знаний]

*(Стрелки обозначают обмен данными между модулями через ядро.)*

**Центральное ядро (Core):** Является «сердцем» системы. Включает в себя **централизованную базу данных** и слой бизнес-логики[[24]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1.%20%D0%AF%D0%B4%D1%80%D0%BE%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B%20%28Core%29%20,%D0%A0%D0%B5%D0%B7%D0%B5%D1%80%D0%B2%D0%BD%D0%BE%D0%B5%20%D0%BA%D0%BE%D0%BF%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%B8%20%D0%B2%D0%BE%D1%81%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Все остальные модули взаимодействуют с ядром для получения или записи данных. В ядре реализовано: - Единое хранилище всех сущностей: заявки, клиенты, поставщики, товары, сообщения, документы и пр., связанные связями (foreign keys или аналог)[[24]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1.%20%D0%AF%D0%B4%D1%80%D0%BE%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B%20%28Core%29%20,%D0%A0%D0%B5%D0%B7%D0%B5%D1%80%D0%B2%D0%BD%D0%BE%D0%B5%20%D0%BA%D0%BE%D0%BF%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%B8%20%D0%B2%D0%BE%D1%81%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). - Управление пользователями и доступом (включая проверку прав при выполнении операций). - Общие сервисы: логирование, уведомления, вычисления, справочники. - API для внешних модулей: ядро предоставляет функции или интерфейсы, которые вызываются парсером, калькулятором, внешними интеграциями. Например, метод create\_order(data) для создания заявки (который может вызываться как UI, так и парсером 1688 при автосоздании черновика заказа), или метод update\_status(order\_id, new\_status) который может вызываться как вручную менеджером, так и модулем отслеживания, и внутри ядра вызовет, например, триггер на добавление записи в историю статусов и отправку уведомления клиенту. - Связующее звено: ядро обеспечивает, что при событии в одном модуле другие обновляются. Например, если калькулятор пересчитал стоимость – ядро обновит общую стоимость в заявке; если пришло сообщение от клиента – ядро пометит заявку, связав её с непрочитанным сообщением.

**Модуль управления заказами:** Отвечает за всё, что связано с заявками (orders). Включает в себя UI для работы с заказами и логику обработки заявок. Через ядро модуль получает список заказов, сохраняет новые, обновляет поля. Взаимодействия: - Получает данные от **парсера 1688** (например, когда менеджер парсит товар, модуль заказов принимает от парсера черновик заказа со всеми полями, чтобы менеджер его подтвердил или отредактировал). - Передаёт данные в **документооборот** (когда нужно сгенерировать документы по заказу) и **калькулятор** (для расчёта итоговых платежей). - Получает сигналы от **модуля отслеживания** (например, изменение статуса доставки) и обновляет статус заказа[[25]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%9E%D0%B1%D0%BD%D0%BE%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%20%D0%BE%D0%B1%D1%89%D0%B8%D0%B9%20%D1%80%D0%B0%D1%81%D1%87%D0%B5%D1%82%20%D0%B2%20%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B5). - Использует **базу знаний** для автоподстановки некоторых данных (например, подсказывает форму адреса или предлагает коды ТН ВЭД по базе при вводе). - Обменивается информацией с **коммуникационным модулем**: например, при создании новой заявки менеджер может сразу из интерфейса написать клиенту, или если из чата Telegram пришёл новый запрос – модуль заказов может создать новую заявку на его основе.

**Модуль парсинга 1688:** Внешний по отношению к ядру компонент, задача которого – по запросу менеджера (ввести URL товара или ID на 1688) собрать информацию о товаре. Технически это может быть реализовано как отдельная служба (например, скрипт Python с Selenium) или встроено в приложение (например, скрытый браузерный виджет). Модуль парсинга взаимодействует с ядром следующим образом: - По команде «Спарсить» – запускает процесс получения данных с указанной страницы 1688. Использует инструменты: headless-браузер с поддержкой JS, обходит блокировки (имитация реального браузера, ожидание загрузки динамических данных)[[26]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%9F%D0%BE%D0%B4%D0%B4%D0%B5%D1%80%D0%B6%D0%BA%D0%B0%20%D0%BA%D0%B8%D1%82%D0%B0%D0%B9%D1%81%D0%BA%D0%BE%D0%B3%D0%BE%20%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0%20%D0%B8%20%D1%82%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8). Также учитывает языковые нюансы – данные на китайском языке могут сразу транслитерироваться или переводиться для удобства, хотя бы названия. - Получив данные о товаре, парсер обращается к API ядра для создания **черновика** нового заказа или товара: например, вызывает метод создания заказа и передает собранные поля (поставщик, описание товара, цена, вес и т.д.). Ядро сохраняет запись в базу, возвращает ID заявки. - Далее фронтенд может автоматически перенаправить менеджера на экран редактирования созданного черновика заказа, где он проверяет/дополняет данные и сохраняет окончательно. Интеграция плотная: если парсер нашёл несколько вариантов поставщиков одного товара, он может предложить менеджеру выбор прямо в интерфейсе, и затем с выбранным поставщиком создать заказ. - Парсер также может сохранять **изображения**: фотографии товара скачиваются и прикрепляются к заказу (сохраняются на диск или в базу, а ссылки – в полях заказа). - Если требуется периодический парсинг (например, обновление цен с 1688), модуль может работать и в фоновом режиме: по запросу из ядра он заново заходит на страницу товара и обновляет цену, тогда ядро пересчитывает стоимость заказа и уведомляет менеджера об изменении.

**Модуль документооборота:** Компонент, ответственный за шаблоны и генерацию документов: - Хранит шаблоны документов (например, в формате HTML+CSS, DOCX, ODT или собственном шаблонном языке). При запросе на генерацию получает из ядра все необходимые данные заявки (и связанных сущностей: клиент, список товаров/груза, расчёты из калькулятора и т.п.), подставляет их в шаблон и выдаёт документ (PDF или другой формат). - Может работать в обе стороны: из ядра документ-сервису передаются данные для генерации; обратно в систему модуль возвращает готовый файл, который сохраняется в базу (либо файловое хранилище, а ссылка – в базу). - Документооборот связан с **коммуникациями**: после генерации можно сразу отправить документ клиенту, вызвав API Telegram или электронной почты. - Также взаимодействует с **базой знаний**: например, подтягивает актуальные курсы валют, чтобы проставить в документе примечание (как в примере – «Цены актуальны на 16.09.2024, курс 1 CNY = 12.60 ₽»[[27]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%A3%D0%B2%D0%B5%D0%B4%D0%BE%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%3A%20,%D0%9F%D0%B5%D1%80%D0%B5%D1%81%D1%87%D0%B8%D1%82%D0%B0%D1%82%D1%8C%20%D0%B2%D1%81%D0%B5%20%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5%20%D0%B7%D0%B0%D1%8F%D0%B2%D0%BA%D0%B8)). - Документооборот регистрирует в журнале кто и когда генерировал или отправлял документ (через ядро).

**Таможенный калькулятор:** Модуль, который может быть реализован как часть серверной логики (напрямую в ядре) или как отдельный сервис. Он использует данные из **базы знаний** (ставки пошлин, правила расчёта НДС и т.д.) для вычислений. Взаимодействие: - Из ядра получает запрос на расчёт (например, когда менеджер нажимает «Рассчитать» или меняет параметры). В запросе – все необходимые параметры (стоимость товара, код ТН ВЭД, вес, страна отправления, желаемый пункт назначения и т.п.). - Возвращает в ядро рассчитанные значения: сумма пошлины, НДС, другие сборы, а также список необходимых документов для этого товара (при особых категориях). - Интегрирован с **модулем заказов** – рассчитанные данные сохраняются в заявку (общая стоимость, в том числе разбивка по таможенным платежам, чтобы менеджер видел структуру стоимости). - Интегрирован с **отслеживанием** – при изменении маршрута доставки или транспортной компании калькулятор может быть вызван повторно для пересчёта логистических тарифов[[25]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%9E%D0%B1%D0%BD%D0%BE%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%20%D0%BE%D0%B1%D1%89%D0%B8%D0%B9%20%D1%80%D0%B0%D1%81%D1%87%D0%B5%D1%82%20%D0%B2%20%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B5). - Возможно, имеет собственный UI (вкладка «Калькулятор») для ручной работы, но основные вычисления происходят на уровне бэкэнда.

**Модуль отслеживания груза (трекер):** Компонент, связанный с внешними логистическими системами: - После того как в заказе указаны данные отправления (номер накладной, перевозчик), трекер через API этих перевозчиков (или через универсальный агрегатор типа 17track) периодически опрашивает статус доставки. Можно реализовать как фоновые задачи (cron-job, celery task и т.д.) или push-уведомления (некоторые сервисы сами умеют слать вебхуки при обновлении статуса). - Получив обновление статуса, трекер обращается к ядру, чтобы: а) обновить статус доставки в заявке (например, изменить статус заявки на «В пути» или «Прибыло в РФ» и т.п.), б) записать событие в историю перемещения груза (отдельная сущность «ShipmentEvent» например), в) сгенерировать уведомление. Далее ядро может уведомить клиента (через Telegram) и всегда уведомляет менеджера внутри приложения. - Если менеджер вручную изменяет статус заявки (например, закрывает её по завершении доставки), трекер может остановить слежение по данному трек-номеру. - Трекер тесно связан с **калькулятором** и **заказами**: он использует данные заказа (вес, габариты, тип товара) чтобы определить какой перевозчик выбран и как часто проверять статус. Например, авиадоставка обычно короче – проверять чаще, морская – реже. При изменении маршрута (как в примере со сменой на авиадоставку) трекер тоже обновляет свою стратегию отслеживания. - Может предоставлять визуализацию маршрута: для этого либо хранит координаты/города из статусов, либо запрашивает у API карту. Это чисто фронтенд-функция, но данные дает трекер.

**Коммуникационный модуль (интеграция с мессенджерами):** Здесь объединены интеграции с Telegram, WeChat (и потенциально другими каналами). Архитектурно может состоять из субмодулей: - **Интеграция Telegram:** Представляет собой, по сути, бота, работающего через Telegram Bot API[[9]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%2A%2A%D0%9A%D0%B0%D0%BA%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%B5%D1%82%3A%2A%2A%20,%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%B0%D0%BC%D0%B8%20%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B8%D1%80%D1%83%D1%8E%D1%82%D1%81%D1%8F%20%D1%81%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B9%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), и обработчик сообщений. При входящем сообщении от клиента бот вызывает webhook на сервер (либо длинное подключение), который обрабатывается нашим приложением. Сообщение идентифицируется (например, по from\_user.id находит клиента в базе)[[28]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=def%20handle_telegram_message%28update%2C%20context%29%3A%20,id), затем сохраняется в таблицу сообщений через ядро[[29]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,id). Параллельно может отправляться уведомление в UI. Для исходящих сообщений приложение вызывает метод API Telegram sendMessage/sendDocument с токеном бота. Кроме текстовых, поддерживается отправка файлов (в боте – документ или фото). - **Интеграция WeChat:** Если используется WeChat Work (корпоративный WeChat), то через соответствующее API (включая webhook для входящих сообщений). Если же личный WeChat через десктоп-клиент – то интеграция более сложная: возможно, запускается скрытый модуль, который отслеживает файлы истории чатов WeChat или использует неофициальный API. В нашем случае, судя по сценарию с QR-кодом, реализовано через WeChat Desktop: менеджер сканирует QR, приложение получает доступ к сессии. Далее модуль отслеживает новые сообщения: это может быть сделано через чтение лог-файлов или эмулирование клиентского приложения. Входящее сообщение определяется – находим в базе поставщика по ID или имени – сохраняем в общую таблицу сообщений. Исходящие – через механизм отправки сообщения от имени пользователя (возможно, имитируя нажатие клавиш в WeChat Desktop через API Windows, либо через автоматизацию). Детали реализации могут быть скрыты от пользователя: главное, что в системе сообщения отображаются и синхронизируются[[30]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%2A%2A%D0%9A%D0%B0%D0%BA%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%B5%D1%82%3A%2A%2A%20,%D1%84%D0%B0%D0%B9%D0%BB%D1%8B%20%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B8%D1%80%D1%83%D1%8E%D1%82%D1%81%D1%8F%20%D1%81%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B9%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). - **Связь с ядром:** Коммуникационный модуль не хранит данные сам, он обращается к ядру для чтения контактов (например, узнать, есть ли уже клиент с таким Telegram ID) и для сохранения сообщений. Также, при настройке интеграции, модуль через ядро записывает параметры (токены, QR-ключи) в таблицу настроек. - **Единый интерфейс:** Для верхнего уровня системы коммуникационный модуль предоставляет унифицированный интерфейс «Сообщения», где независимо от канала, диалоги представлены похожим образом. Это достигается тем, что все сообщения в БД хранятся с указанием канала (telegram/wechat/другой) и идентификаторов собеседника, а UI уже группирует их.

**База знаний (справочники):** Это не столько модуль с логикой, сколько набор данных и утилит для доступа к ним: - Взаимодействует с ядром для обновления/чтения справочных данных. - Используется **калькулятором** (ставки валют, пошлин), **документооборотом** (шаблоны, тексты), **заказами** (коды ТН ВЭД, справочники адресов, статусов), **трекером** (список перевозчиков и их API). - Имеет административный UI для пополнения данных. - В архитектуре база знаний обеспечивает централизованное хранение того, что может использоваться в разных модулях, чтобы не было дублирования. Например, обновление курса валют происходит один раз в справочнике, и все модули берут оттуда актуальные значения, вместо того чтобы каждый подключался к внешнему сервису сам.

Таким образом, архитектура построена вокруг **ядра и единой базы данных**, к которой подключены специализированные модули. Это позволяет сохранить целостность данных (все данные в одной БД с правильными связями) и одновременно разделить ответственность по модулям. Интеграции с внешними сервисами (мессенджеры, 1688, логистика, валюты) вынесены в отдельные компоненты, которые можно модифицировать или масштабировать независимо. Например, если 1688 изменит формат страниц, можно обновить модуль парсинга, не затрагивая остальные части системы.

## 3. Визуальная схема интерфейса и логика пользовательского взаимодействия

Опишем пошагово сценарий работы в системе «Карго Менеджер Лисёнок» – **от получения запроса клиента до закрытия сделки**, с точки зрения пользователя (карго-менеджера Лисёнка):

**Шаг 1. Получение запроса от клиента.** Предположим, российский клиент отправляет Лисёнку сообщение с просьбой просчитать стоимость доставки партии товара из Китая. Это сообщение приходит через Telegram – либо на личный аккаунт менеджера, либо, как планируется, на Telegram-бота компании. Благодаря интеграции, это сообщение сразу появляется в разделе **«Сообщения»** системы: всплывает уведомление «Новое сообщение от клиента Иван П.» с текстом запроса. Менеджер открывает вкладку «Сообщения» и видит чат с этим клиентом. Если это новый клиент, система предложит создать для него карточку (заполнить имя, контакты) и новую заявку; если клиент уже существующий и, возможно, уже есть открытая заявка, сообщение привяжется к ней. Лисёнок читает запрос: клиент, например, прислал ссылку на товар на 1688 и спросил ориентировочную стоимость доставки 100 штук до Москвы.

**Шаг 2. Создание новой заявки.** Из интерфейса чата менеджер может сразу создать заявку: нажимает кнопку «+ Создать заявку на основе сообщения». Система переключает его на вкладку **«Заказы»** с открытой формой новой заявки, куда уже подставлены известные данные: клиент – Иван П., описание груза – взято из сообщения (например, «электронные компоненты, 100 шт»), возможно, ссылка на товар сохранена в поле примечания. Лисёнок заполняет недостающие поля: выбирает (или добавляет) поставщика, если известен конкретный (пока нет – можно оставить пустым или «требуется выбрать»), количество, примерный вес/объём если знает. Поля «статус» ставит «Новый» или «Запрос просчёта». Сохраняет заявку. В списке заказов появляется новая строка, ей присваивается уникальный номер (например, #2025-001).

**Шаг 3. Парсинг товара с 1688.** Теперь задача – получить точные данные о товаре из присланной ссылки. Менеджер копирует ссылку из сообщения (если она не была автоматически распознана). Переходит во вкладку **«Калькулятор»** или в форму внутри заявки (в UI может быть кнопка «Парсить с 1688» прямо в карточке заказа). Вставляет ссылку в поле парсера и жмёт «Старт». На экране может показываться индикатор загрузки. Через несколько секунд система заполняет поля товара: название (на русском, если доступен перевод, либо на английском с транслитерацией), описание, изображения. Если товар многовариантный, возможно, система уточнит параметры. Менеджер видит, что появилось окно предварительного просмотра товара с картинкой, характеристиками и ценой, например: «Минимальный заказ: 100 шт, Цена: 10 ¥ за шт». Система также сразу рассчитала ориентировочную логистику: предположим, определила вес одной штуки 200 г, итого вес партии ~20 кг, объем небольшой – рассчитала тарифный вес 20 кг. По умолчанию выбран текущий базовый маршрут (например, морем, 30 дней). Калькулятор показывает: пошлина 0% (если, допустим, электронные компоненты беспошлинно), доставка морем 5 000 ₽, итоговая стоимость ~ **5 000 ₽** + стоимость товара (10¥*100=1000¥, ~12 600 ₽ по курсу). Общая оценка: ~17 600 ₽. Менеджер может переключить способ доставки на авиа, чтобы быстро сравнить – результаты мгновенно обновляются: авиа, 7 дней, но 90 000 ₽, что сильно дороже. Лисёнок оставляет морской вариант. Все эти данные уже подставлены в* *заявку*\* – система обновила поля: вес = 20 кг, объем, ориентировочная стоимость, маршрут, срок.

**Шаг 4. Подготовка коммерческого предложения.** Менеджер доводит коммерческое предложение до готовности. Во вкладке **«Документы»** он выбирает шаблон «Просчёт для клиента» (коммерческое предложение). Система автоматически генерирует документ: подставляет в него данные клиента, описание груза, выбранный маршрут, стоимость товара (может быть указано в ¥ и рублях), стоимость доставки, итоговую сумму в рублях, а также условия сделки (например, предоплата, сроки доставки, включенные услуги). Документ открывается для проверки. Лисёнок просматривает и видит, что всё верно. В документе также пометка мелким шрифтом: «Курс валют: 1 CNY = 12.60 ₽, расчет верен на дату 31.08.2025» (эти данные система взяла из базы знаний автоматически). Менеджер нажимает «Отправить клиенту», выбирает канал Telegram – система сама прикрепляет PDF документа и отправляет в нужный чат с клиентом. Клиент оперативно получает сообщение от бота с файлом и благодарит. Весь разговор виден Лисёнку в системе.

**Шаг 5. Согласование и заказ товара у поставщика.** Клиент изучает коммерческое предложение и через некоторое время в Telegram сообщает, что его устраивает цена и сроки, просит выставить счёт. Это сообщение снова приходит в систему, и менеджер видит уведомление. Заявка переводится в статус «Ожидается оплата/подтверждение». Далее Лисёнок выставляет счёт: во вкладке **«Документы»** генерирует **инвойс** на предоплату или договор. Система заполняет реквизиты клиента и суммы, менеджер добавляет номер счёта. Отправляет документ клиенту, возможно, дублирует на email (если предусмотреть такую функцию – например, кнопкой «отправить на email»). После получения оплаты (это вне системы, но менеджер фиксирует событие вручную) заказ переходит в статус «Оплачено, заказывается у поставщика».

Теперь менеджеру нужно связаться с поставщиком на 1688 (китайская сторона). Через систему он копирует детали заказа на китайском (возможно, парсер 1688 уже предоставил название товара на китайском и контакт поставщика). Он открывает **WeChat** – интеграция может позволять отправить сообщение прямо из системы: например, выбрать действие «написать поставщику» в карточке поставщика, что открывает встроенный чат WeChat. Если такой функционал реализован, Лисёнок пишет на русском или английском, при необходимости переводит на китайский (возможно, через встроенный переводчик, но это опционально), и отправляет сообщение поставщику: заказ подтверждён, прошу оформить на 100 шт. Поставщик отвечает в WeChat, например: «Ок, отправим в течение 5 дней». Это ответное сообщение опять же фиксируется системой. Менеджер при необходимости прикрепляет в систему счет от поставщика или скриншот оплаты, но это скорее дополнительная опция (можно хранить файлы в привязке к заявке).

**Шаг 6. Доставка и отслеживание.** Поставщик отправляет товар, предоставляет трек-номер или накладную. Лисёнок вносит эти данные: открывает заявку, вводит полученный трек-номер и выбирает перевозчика (например, SF Express). Заявка переходит в статус «Отправлено, в пути». Модуль отслеживания начинает мониторить этот трек. Через некоторое время система обновляет статус: «Прибыло в г. Манчжурия, экспорт из Китая». Менеджер и клиент получают уведомления (клиенту уходит сообщение в Telegram: «Ваш груз #2025-001 покинул Китай, ожидается в РФ через 5 дней»). Клиент может также сам запросить статус, отправив команду боту, но чаще система проактивно уведомляет по ключевым этапам. В интерфейсе **«Отслеживание»** менеджер может видеть на карте путь груза и текущую точку. Также он видит прогноз: на основе данных система сообщает, что ориентировочная дата прибытия – например, 25 сентября (плюс-минус 2 дня).

**Шаг 7. Изменения и непредвиденные ситуации (пример).** Пока груз в пути, курс юаня значительно вырос – это может повлиять на себестоимость. Поскольку система автоматически отслеживает курсы, она заметила рост, пересчитала ориентировочную прибыль/расходы по активным заявкам и выявила, что по данной заявке прибыль снизится. Менеджеру приходит уведомление: «Курс CNY вырос на 3%, стоимость по заявке #2025-001 увеличилась на 2.5%»[[16]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1,%E2%86%92%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0%20%D1%88%D0%BB%D0%B5%D1%82%20%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F). Появляется кнопка «Уведомить клиента». Если нажать, система отправит шаблонное сообщение в чат с клиентом, например: «Уведомляем, что произошло изменение валютного курса, итоговая стоимость может незначительно измениться». Лисёнок решает, нужно ли это делать, и при необходимости отправляет (либо игнорирует, если считает, что это некритично).

Другой пример: вдруг клиент решил увеличить объем заказа – он пишет об этом. Лисёнок редактирует заявку: изменяет количество на 150 шт. Сразу же система предлагает обновить расчёты – менеджер подтверждает, и калькулятор пересчитывает вес, стоимость, пошлины. Документы (коммерческое предложение, счет) обновляются – их можно снова отправить клиенту для подтверждения доплаты.

**Шаг 8. Завершение сделки.** Груз прибыл в Россию, прошёл таможню и доставлен клиенту. Менеджер отмечает заказ как **«Доставлен»** – в системе статус меняется на финальный, заявка закрывается. Клиент автоматически получает финальное сообщение благодарности (по желанию компании, можно настроить шаблон «Спасибо за сотрудничество, ждём вас снова»). Вся информация по сделке – переписка, документы, расчёты, события отслеживания – теперь сохранена и доступна для последующего анализа.

**Шаг 9. Пост-аналитика и обслуживание:** Администратор может выгрузить данные для отчетности – например, сколько заявок выполнено, финансовый результат. Если клиент возвращается через месяц с повторным заказом, менеджер уже видит историю предыдущего взаимодействия: открыв карточку клиента, доступны все прошлые заявки и сообщения. Система позволила значительно ускорить работу: многие шаги, которые ранее Лисёнок делала вручную (копирование данных между Excel, чатами и документами), теперь автоматизированы. Например, **в сценарии выше экономия времени** очевидна: парсинг товара и заполнение данных – секунды вместо 15 минут ручной работы, обновление курсов – автоматическое вместо ручного перерасчёта, отправка статусов – одним кликом вместо множества ручных сообщений[[31]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%9E%D0%B1%D0%BD%D0%BE%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D1%8B%20%D0%B3%D1%80%D1%83%D0%B7%D0%B0)[[32]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1,%E2%86%92%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0%20%D1%88%D0%BB%D0%B5%D1%82%20%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F).

Визуально интерфейс старался быть **максимально дружественным**: единый рабочий стол со всеми инструментами под рукой. Менеджер может выполнять несколько задач параллельно (например, пока парсер загружает данные по одному товару, можно просматривать другую заявку – модульность и отсутствие перезагрузок вкладок этому способствуют). Настольное приложение позволяет открывать окна поверх (например, окно чата поверх списка заказов для быстрого ответа клиенту). Благодаря адаптивности, часть работы можно вести на планшете, что удобно, например, на складе или в дороге – интерфейс подстроится под сенсорный экран.

Вся **логика взаимодействия** нацелена на то, чтобы исключить дублирование действий: информация введена один раз далее течёт по системе. Запрос клиента породил заявку, из нее родились документы и сообщения, статусы обновлялись автоматически, а менеджеру оставалось принимать решения и контролировать. Таким образом, «Карго Менеджер Лисёнок» обеспечивает сквозной процесс от первого сообщения клиента до финальной доставки, **наглядно** отображая всю нужную информацию на каждом этапе и позволяя мгновенно реагировать на события.

## 4. Перечень технологий и инструментов с обоснованием выбора

Для реализации проекта выбраны современные, проверенные технологии, обеспечивающие надежность, удобство разработки и соответствие требованиям десктопного и веб-приложения:

* **Backend (серверная часть):**
* **Python 3.10+ с FastAPI** – предпочтительный выбор для бэкенда. *Обоснование:* Python имеет богатую экосистему для работы с сетевыми запросами, парсингом и интеграциями. Библиотека FastAPI позволяет быстро создавать RESTful API и WebSocket-соединения для realtime-обновлений (например, нотификации о новых сообщениях или статусах груза) с отличной производительностью. Кроме того, на Python проще реализовать сложную бизнес-логику (например, расчет пошлин) благодаря читаемому синтаксису и большим возможностям в научных библиотеках. Многие части системы (парсер 1688, интеграция с Telegram) уже хорошо поддерживаются в Python: есть библиотеки **Selenium/Playwright** для парсинга веб-страниц, **python-telegram-bot** для Telegram Bot API, неофициальные библиотеки для WeChat. Python также знаком команде (судя по тому, что ранее использовался tkinter и sqlite), что снизит порог поддержки. Альтернативой мог бы быть Node.js (например, фреймворк NestJS или Express) – он тоже хорошо подходит для real-time (через Socket.io) и интеграций. Но Python выбран из-за простоты реализации специфических вещей (PDF-генерация, вычисления). Дополнительно: на Python легко реализовать кроссплатформенный настольный вариант (например, с библиотекой PyInstaller упаковать сервер+UI).
* **Frontend (клиентская часть):**
* **React 18+ с Redux Toolkit** – для построения одностраничного приложения (SPA). *Обоснование:* React – популярная библиотека для создания динамичного UI. Она позволяет разбить интерфейс на многократно используемые компоненты (например, компонент таблицы заказов, компонент чат-окна). Вместе с Redux мы получаем предсказуемое хранение состояния приложения – удобно держать данные заказов, сообщений, текущих фильтров в едином хранилище, что облегчает взаимодействие разных частей интерфейса. Выбор SPA означает, что переходы между разделами (заказы, калькулятор, сообщения) будут быстрыми, без полной перезагрузки страницы – это соответствует требованию «единый рабочий стол». Также React хорошо поддерживает адаптивность под мобильные устройства. Кроме того, команда уже пробовала связку React + Vite (судя по предоставленному package.json) – эту же связку целесообразно продолжить использовать. Vite обеспечивает быструю сборку и запуск в Dev-режиме, что ускоряет разработку.
* **Tailwind CSS** – CSS-фреймворк для стилизации. *Обоснование:* Tailwind позволяет быстро и единообразно оформлять приложение с помощью утилитарных классов, не отвлекаясь на написание сырого CSS. Он хорошо интегрируется с React. С помощью Tailwind можно легко реализовать адаптивность (например, скрыть боковое меню на мобильном, изменить компоновку) и поддерживать чистый дизайн. В итоговом интерфейсе для пользователя важна **читабельность и удобство**, Tailwind помогает достичь современного лаконичного UI без больших затрат.
* **Electron (опционально)** – фреймворк для упаковки веб-приложения в десктоп. *Обоснование:* Если требуется полноценное настольное приложение (работающее офлайн и имеющее доступ, например, к файловой системе), можно использовать Electron. Он позволит упаковать наш фронтенд (HTML/JS/CSS) и бэкенд-сервер в единый исполняемый файл, который пользователь запустит на своем компьютере как обычную программу. Electron хорош тем, что код переиспользуется – веб-версия и десктоп будут на одном React. Минус – некоторая ресурсоёмкость. Данный инструмент рассматривается для удобства пользователей, уже привыкших работать с локальными программами.
* **База данных:**
* **PostgreSQL 14+** – основная СУБД для боевой эксплуатации. *Обоснование:* PostgreSQL – промышленно признанная реляционная база данных, известная надежностью и соответствием стандартам SQL. Она отлично работает с сложными связями, поддерживает хранимые процедуры, триггеры (например, можно использовать триггер для автоматического логирования изменений статусов или синхронизации, как демонстрировалось в проекте) и масштабируется до больших объемов данных. Для нашей системы, где много связанных таблиц (заказы, товары, клиенты, сообщения, события) – реляционная модель подходит идеально. PostgreSQL также поддерживает JSONB-поля, что может пригодиться для хранения, к примеру, произвольных структур (вложений сообщений или версий документов). На этапе MVP, как упоминалось, допустимо использовать **SQLite** (встроенная легковесная БД) – она не требует установки сервера и проще при разработке[[33]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%98%D0%BD%D0%B8%D1%86%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20%D0%B1%D0%B0%D0%B7%D1%8B%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%20def%20init_db,cursor). Но при переходе в продакшн с несколькими пользователями и высокими требованиями к надёжности, лучше сразу PostgreSQL. Возможна также **MySQL/MariaDB** – но PostgreSQL предпочтительнее за счёт более строгого соответствия ACID и расширенных возможностей.
* **Redis (опционально):** Если понадобится реализовать очереди задач или хранение сессий, можно использовать Redis – например, для фоновой очереди обработки парсинга (чтобы несколько процессов получали задачи парсинга из очереди) или для хранении JWT-токенов/сессий пользователей для масштабирования на несколько серверов.
* **Интеграция 1688 (парсинг):**
* **Playwright или Selenium** – инструменты для автоматизированного парсинга сайтов. *Обоснование:* Платформа 1688, будучи динамической и защищенной, требует полноценного браузера для корректного получения данных. **Playwright** (Microsoft) или **Puppeteer** (Headless Chrome) – хорошо подходят, поддерживают эмуляцию пользователя, ожидание загрузки элементов, работу с CAPTCHA (в какой-то мере) и т.д. Python имеет Playwright API, Node.js – Puppeteer. Выбор Python + Playwright выглядит привлекательно, т.к. позволит легко интегрировать с остальным Python-кодом. В реализации парсера будет использоваться либо безголовый браузер, либо, для экономии ресурсов, **API Alibaba**. Возможно, у Alibaba есть API для некоторых данных (например, Taobao Open Platform), но для 1688 открытых API нет, поэтому **web-scraping – основное решение**.
* Для парсинга и обработки данных потребуется библиотека для HTML-анализа, например **BeautifulSoup4** (Python) для разбора статического HTML, или использование Playwright’s page methods для получения текста узлов. Также пригодится **googletrans** или аналог для перевода китайского описания на русский (если нужно автоперевод).
* **Интеграция Telegram:**
* **Telegram Bot API** – будет использоваться для получения/отправки сообщений. *Обоснование:* Telegram предоставляет удобный HTTP API для ботов, что упрощает интеграцию. В системе будет создан бот (через @BotFather), и его токен сохранён в настройках. Для обработки удобно использовать готовую библиотеку, например, **python-telegram-bot** (если бэкенд на Python) либо **Telegraf** (если Node.js). Эти библиотеки обеспечивают прослушивание обновлений (update) и маршрутизацию команд. Telegram бот может работать в режиме webhook (требует доступного из интернета URL) или long polling. Для надёжности можно реализовать webhook через сервис типа ngrok на этапе разработки, а в продакшне – развернуть в облаке с HTTPS. Использование Bot API оправдано тем, что клиенты уже привыкли к мессенджеру, а бот – официальный и безопасный способ взаимодействия.
* **Telegram API (MTProto, optional):** Альтернативно, можно использовать более низкоуровневый Telegram API через библиотеку **Telethon** (Python). Это позволило бы получать сообщения не только, адресованные боту, но и перехватывать, например, личные сообщения менеджеру. Однако это усложняет архитектуру и требует хранения телефона/кода. Поэтому основной путь – бот.
* Со стороны фронтенда, для real-time обновления чата, задействуем **WebSockets**. Бэкенд при получении нового сообщения от Telegram будет слать событие через WebSocket соединение на клиент, где Redux-store обновится и отобразит новое сообщение в UI (пример соответствующего кода уже был предусмотрен)[[34]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=3,from%20%27..%2Fstore)[[35]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=switch%20%28data.type%29%20,%29%3B%20break).
* **Интеграция WeChat:**
* **WeChat Official API / WeCom** – если удастся убедить поставщиков общаться через официальный канал. *Обоснование:* У WeChat (Weixin) есть API для официальных аккаунтов и для WeChat Work (корпоративный WeChat). Если Лисёнок зарегистрирует корпоративный аккаунт, можно через API получать сообщения от добавленных контактов и отправлять им. Это надежный и поддерживаемый способ. Библиотек не так много, но можно работать через прямые HTTP запросы к WeChat API.
* **Неофициальные методы (через WeChat Desktop):** Поскольку в реальности поставщики, скорее всего, общаются через обычный WeChat, может потребоваться обходной путь. Например, использовать библиотеку **ItChat** (Python) – она позволяет через Web WeChat (сканирование QR) залогиниться и получать события сообщений программно. Однако, WeChat периодически блокирует такие попытки, поэтому нужно быть осторожным. Другой путь – написать небольшое Windows-приложение, которое с использованием WinAPI читает окна WeChat Desktop (например, используя Accessibility API) или мониторит файлы. В предоставленном описании говорится о «мониторинге папки WeChat»[[36]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=4.%20%D0%A1%D0%BA%D0%B0%D0%BD%D0%B8%D1%80%D1%83%D0%B5%D1%82%20QR,%D0%BE%D1%82%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%BE%D0%B2%20%D0%BF%D0%BE%D1%8F%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F%20%D0%B2%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B8) – видимо, выбрана стратегия отслеживать директорию, куда WeChat Desktop сохраняет медиафайлы, и, возможно, parse sqlite-файл истории чатов WeChat (который клиент хранит локально). Это достаточно хакерский метод, но вполне реализуемый.
* Как бы то ни было, технология может быть: Python + ItChat, или C# + формирование Windows-хука. В контексте нашего проекта проще взять Python-скрипт с ItChat: он запускается при настройке, авторизуется, и далее шлёт события в наше приложение (например, через REST API ядра или через запись в БД). Для отправки сообщений поставщику из системы скрипт тоже вызывает соответствующий метод.
* Безопасность: нужно хранить куки сессии WeChat, чтобы не требовать сканирования QR при каждом запуске. Храним в зашифрованном виде на диске пользователя.
* **Дополнительные библиотеки и инструменты:**
* **SQLAlchemy (Python)** – ORM для удобной работы с базой данных. Ускорит разработку, позволив оперировать объектами (Order, Client и т.д.) вместо ручного SQL, при этом поддерживает PostgreSQL, SQLite без изменений кода.
* **Alembic** – для миграций схемы БД (чтобы легко обновлять структуру таблиц между версиями).
* **Celery** – если понадобится задача фоновой обработки (например, парсинг, который лучше вынести в отдельный поток/процесс, или периодическая проверка статусов доставки).
* **ReportLab или WeasyPrint** – генерация PDF. ReportLab позволяет програмно рисовать PDF (подходит для форм инвойсов, где много табличных данных). WeasyPrint генерирует PDF из HTML/CSS – удобно, если шаблоны писаться как HTML.
* **OpenAPI/Swagger:** FastAPI автоматически генерирует документацию API. Это поможет при разработке фронтенда и для возможного будущего открытия API для клиентов.
* **JWT (JSON Web Tokens):** для авторизации пользователей в веб-версии. После логина сервер выдаёт токен, фронтенд хранит его и шлет с каждым запросом. Это стандарт для SPA.
* **Git** – система контроля версий для ведения проекта, **GitHub/GitLab** – для совместной работы и хранения кода. Continuous Integration (например, GitHub Actions) можно настроить для прогона тестов при каждом коммите.
* **Docker** – контейнеризация для облегчения деплоя. Создадим Dockerfile для бэкенда (Python + зависимости), возможно, отдельный контейнер для БД, и Docker Compose для локального запуска всех компонентов. Это даст портативность: развернуть на сервере можно запустив соответствующие контейнеры.
* **Тестирование:** PyTest (для Python) – для юнит-тестов логики, Postman или тесты API – для проверки интеграций (например, эмуляция webhook от Telegram). Selenium WebDriver – для end-to-end тестов фронтенда (например, прогнать сценарий: создать заказ -> сгенерировать документ).
* **Monitoring/Logging:** для продакшна можно внедрить Sentry (отслеживание ошибок фронтенда/бэкенда), и базовый мониторинг сервера (метрики CPU/RAM, pinger для API интеграций).

Каждый выбор технологии обоснован сочетанием **опыта, доступности библиотек и масштабируемости**. Использование популярных фреймворков (React, FastAPI) гарантирует, что проект будет поддерживаемым и понятным другим разработчикам, а также легко найдет решения для возникших задач (за счет большого сообщества). В тоже время, специфические инструменты (парсер 1688, WeChat-бот) выбраны из практических соображений, исходя из реальных условий работы Лисёнка – они обеспечат интеграцию с нужными сервисами даже там, где нет официальных удобных API.

## 5. Дорожная карта (Roadmap) проекта

Для успешной реализации проекта «Карго Менеджер Лисёнок» разработан план работ, разбитый на этапы. Каждый этап включает в себя определенный набор задач и результатов:

1. **Этап планирования и анализа (0–1 месяц):** На этом этапе собираются и уточняются все требования.
2. 📌 *Задачи:* Провести дополнительные интервью с пользователем (Лисёнком) для выяснения деталей рабочих процессов, собрать примеры используемых сейчас Excel-файлов, шаблонов документов, переписок – чтобы ничего не упустить. Проанализировать эти материалы и уточнить функциональные требования (при необходимости скорректировать ТЗ). Проработать UX/UI наброски: нарисовать схематически экраны приложения, расположение элементов, чтобы убедиться, что все удобно. Определить технические границы: например, решить, будет ли сначала реализована десктоп-версия или сразу веб, какие интеграции включать в MVP. Составить модель данных – ER-диаграмму основных сущностей и связей. Проектировать архитектуру (более детально, если нужно, расписать интерфейсы между модулями). Выбрать стек технологий окончательно, утвердить его.
3. 💾 *Результаты:* Утвержденное **техническое задание** (данный документ), прототипы интерфейсов (на бумаге или в Figma – низкой детализации, для понимания структуры), диаграмма базы данных, список пользовательских историй (user stories) для реализации. Дорожная карта (этот план) согласована по срокам и этапам.
4. **Этап разработки MVP (1–3 месяц):** Создание минимально жизнеспособного продукта, включающего основные функции.
5. 📌 *Задачи:* Настроить среду разработки (репозиторий, CI, базовую инфраструктуру). Начать с **базы данных и бэкенда**: спроектировать схемы таблиц и реализовать ORM-модели (таблицы клиентов, поставщиков, заказов, товаров, сообщений, документов, пользователей и пр.). Реализовать ключевые API эндпоинты: авторизация пользователей, CRUD для заказов, получение списков справочников. Затем создать **фронтенд каркас**: настроить React + Redux + Tailwind, реализовать навигацию между разделами, сверстать простые версии страниц (список заказов, форма заказа, простейший чат-интерфейс, пустые заглушки для калькулятора, документов). В MVP включаем **самый необходимый функционал**:
   * Создание/редактирование заявок вручную (пока без парсера, данные вводятся руками).
   * Просмотр списка заявок.
   * Добавление клиентов и поставщиков в справочники.
   * Генерация одного-двух простых документов (например, коммерческое предложение) с минимальным шаблоном.
   * Интеграция с Telegram *на базовом уровне*: получение сообщений от бота и отображение их в разделе «Сообщения», отправка текстовых ответов из системы (без отправки файлов пока).
   * Обновление курсов валют: можно либо заглушить фиксированным курсом, либо сразу подключить простой API и записывать в справочник.
   * Базовый калькулятор: формула для расчета объёмного веса и стоимости доставки по одному зафиксированному тарифу, чтобы просчет работал.
   * Отслеживание: в MVP можно сделать вручную – поле для статуса, который менеджер сам меняет (авто-API подключим на следующем этапе).
6. 🤖 *Тестирование:* Написать unit-тесты для модели заказа (например, проверка расчета тарифного веса), протестировать API эндпоинты (Postman коллекция). Убедиться, что фронтенд успешно вызывает бэкенд (можно использовать мок-данные сначала).
7. 💾 *Результаты:* К концу этапа должен получиться работающий прототип: **MVP-версия приложения**. Менеджер может: войти в систему, внести через форму новый заказ, сохранить его в базу, увидеть его в списке; добавить нового клиента/поставщика; вручную вбить расчет данных; сгенерировать PDF коммерческого предложения (хотя бы с логотипом и таблицей цены); через Telegram-бота отправить сообщение и увидеть его внутри приложения (и ответить на него). Этот MVP, хотя и урезанный, уже заменяет часть работы в Excel и демонстрирует концепцию. Важный момент – собрать **фидбек от пользователя**: Лисёнок пробует MVP на реальных данных параллельно с основной работой, отмечает, что удобно, а что нужно улучшить, какие поля или функции добавить.
8. **Этап расширенной интеграции и доработка функций (3–6 месяц):** На этом этапе MVP превращается в полноценный продукт – добавляются все запланированные функции и интеграции, на основе обратной связи исправляются недостатки UX.
9. 📌 *Задачи:*
   * **Парсер 1688:** интегрировать модуль парсинга. Реализовать серверный скрипт (например, с Playwright) и соответствующий фронтенд-UX: поле ввода ссылки и кнопка, индикатор выполнения, затем заполнение полей. Протестировать на разных примерах ссылок 1688, доработать алгоритм извлечения (включая поиск веса, размеров – по китайским ключевым словам)[[37]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,self.status_var.get%28%29)[[38]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=messagebox.showerror%28,return). Добавить возможность редактирования полученных данных перед сохранением заказа.
   * **Продвинутый калькулятор:** закончить функционал таможенного калькулятора. Подключить базу ставок пошлин (можно импортировать из открытых данных или вбить вручную для часто используемых категорий). Сделать подбор кода ТН ВЭД: например, автозаполнение по ключевому слову или выпадающий список категорий. Формулы для НДС, таможенных сборов – заложить согласно актуальному законодательству. Интерфейс улучшить: отдельная вкладка «Калькулятор» – доработать ее, чтобы работала и автономно, и связно с заказом (в заявке кнопка «в калькулятор» подтягивает данные).
   * **Документы:** расширить список документов. Реализовать шаблоны для счет-фактуры, договора, упаковочного листа, и др. Прикрутить библиотеку генерации PDF (если не сделано) – убедиться, что кириллица, китайские иероглифы корректно отображаются в PDF (подключить нужные шрифты). Добавить электронную подпись: вероятно, как прикрепление изображения подписи и печати в PDF. Версионность: сделать так, чтобы при повторной генерации документ не перезаписывался, а создавалась новая запись (можно с номером версии).
   * **Интеграция Telegram (полная):** Дополнить бот функционалом: чтобы новые клиенты могли сами инициировать заявку (например, команда /start регистрирует нового клиента), чтобы бот мог отвечать на запрос статуса (/status по номеру заявки). Но главное – реализовать отправку файлов (документов) через бота прямо из системы. Подключить обработку изображений (если клиент прислал фото – сохранять его). Настроить приоритизацию уведомлений: например, если сообщение содержит слово «срочно», отметить его особым значком.
   * **Интеграция WeChat:** Внедрить выбранный метод. Если это ItChat: поднять отдельный процесс, подключающийся к WeChat. Настроить получение сообщений и команду отправки. Убедиться, что китайский текст отображается правильно в интерфейсе сообщений. Возможно, реализовать простейший переводчик (кнопка «Перевести» рядом с сообщением от поставщика, чтобы менеджеру вывести перевод на русский – можно задействовать API перевода).
   * **Отслеживание грузов (авто):** Подключить реальные API. Например, зарегистрироваться в Cainiao/Alibaba API для трекинга посылок или использовать 17track (у них есть API). Реализовать в системе хранение для отправлений: таблица Shipment с полями (номер, перевозчик, заявка, статус, последние обновление и пр.). Написать фоновые задачи или webhook-слушатели для обновления статусов. Тестирование: взять тестовые трек-номера и пройтись по процессу. Добавить визуальную индикацию на интерфейсе (можно использовать Google Maps API или OSM для отображения мест по координатам).
   * **Уведомления и система событий:** К этому этапу много различных автоматических событий – изменение статуса, новое сообщение, изменение курса. Нужно централизовать работу с уведомлениями. Реализовать на бэкенде механизм отправки оповещений фронтенду (через WebSocket). На фронте – небольшой модуль показа уведомлений (всплывающих или в центре уведомлений). Проверить, что все необходимые случаи покрыты: новый чат, новое сообщение, статус груза, ошибка интеграции (например, если бот отключился).
   * **Улучшение UI/UX:** Отшлифовать интерфейс по результатам тестирования. Возможно, добавить **поиск** по всем заявкам и сообщениям (global search bar). Добавить фильтры, сортировки в таблицах. Реализовать удобные мелочи: например, кнопку «Очистить форму» при создании заказа[[39]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=ttk.Button%28btn_frame%2C%20text%3D,command%3Dself.clear_order_form%29.pack%28side%3Dtk.LEFT%2C%20padx%3D5), подтверждения при удалении записей, показывать спиннер при загрузке данных.
   * **Мультиязычность интерфейса (при необходимости):** если планируется китайский интерфейс – реализовать систему переводов (React Intl или i18next). Если нет, хотя бы убедиться, что все строки вынесены в настройки, чтобы потом было проще.
   * **Безопасность и устойчивость:** Провести внутренний аудит безопасности. Исправить замеченные уязвимости (например, убедиться, что API защищены, нельзя получить данные чужой заявки, вводимые данные проверяются и экранируются, и т.д.). Настроить обработку ошибок: если парсер 1688 не смог получить данные – выводить понятное сообщение пользователю, а не падать.
10. 🤖 *Тестирование:* Этот этап включает существенное **интеграционное тестирование**. Нужно проверить сквозные сценарии:  
    *Сценарий A:* Создание заказа -> Парсинг -> Расчет -> Документ -> Telegram отправка -> изменение статуса -> доставка -> закрытие.  
    *Сценарий B:* Входящее сообщение новый клиент -> авто-создание заявки -> ответ клиенту.  
    *Сценарий C:* Отказ от интеграций: проверить, что если Telegram или WeChat не настроены, система не падает, а предлагает настроить.  
    Также провести нагрузочное тестирование базовых операций (например, загрузить 1000 заказов в базу, посмотреть, не тормозит ли интерфейс). Если есть возможность, привлечь нескольких тестовых пользователей (например, коллега Лисёнка) для имитации одновременной работы.
11. 💾 *Результаты:* Полностью функциональная версия **Beta 1.0**. Она включает все заявленные в ТЗ возможности: и автоматизации, и интеграции. По окончании этого этапа приложение передается Лисёнку для тщательного испытания в боевых условиях.
12. **Этап бета-тестирования и отладки (6–7 месяц):** Внедрение системы в реальную работу Лисёнка на ограниченное время и аудит результата.
13. 📌 *Задачи:* В течение согласованного периода (например, 2–4 недели) Лисёнок ведет все новые заказы параллельно в системе и в своём старом процессе, сравнивая результаты. Собираются все возникшие проблемы: баги, неудобства, недостающий функционал. Регулярно (каждые несколько дней) проводится сессия сбора обратной связи. Разработчики оперативно исправляют критические ошибки. Оптимизируют приложение по итогам: например, ускоряют медленные места, исправляют вылеты. Проводится дополнительное обучение пользователя, если требуется (хотя интерфейс старались сделать интуитивным, могут потребоваться пояснения для новых процессов, например, как именно подключить бота или что означает тот или иной статус).
14. Особое внимание уделяется **надежности интеграций**: чтобы Telegram-бот не отключался, чтобы парсер 1688 стабильно работал (возможно, добавить резервный вариант парсинга, если основной метод не сработал). Проверяются граничные случаи: очень длинные сообщения, файлы необычных форматов, нестандартные товары.
15. Если во время беты выявляются небольшие недочеты в функционале (что-то упущено, например, нужен дополнительный тип документа или поле в заявке) – по возможности добавляем на лету.
16. 🤖 *Тестирование:* Именно на этом этапе происходит **приемочное тестирование** (UAT) пользователем. Разработчики могут составить контрольный лист, по которому Лисёнок пройдет и подтвердит работу всех ключевых функций (чек-лист по критериям успеха: запуск без ошибок, создание записи, интеграция с каждым мессенджером, обновление курса, генерация документов, права доступа и т.п. – все должно быть пройдено)[[40]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=4.%20%2A%2A%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%2A%2A%3A%20,%D0%B2%D0%B0%D0%BB%D1%8E%D1%82%20%D0%B2%20%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%20%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8)[[41]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%9F%D1%80%D0%BE%D0%BC%D1%82%20%D0%BF%D0%BE%D0%B4%D1%80%D0%BE%D0%B1%D0%BD%D0%BE%20%D0%BE%D0%BF%D0%B8%D1%81%D1%8B%D0%B2%D0%B0%D0%B5%D1%82%3A%201,%D0%9A%D0%B0%D0%BA%20%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D0%B8%D1%82%D1%8C%20%D0%BA%D0%B0%D0%B6%D0%B4%D1%83%D1%8E%20%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8E).
17. 💾 *Результаты:* **Release Candidate** версия приложения. После исправления всех найденных багов и полировки интерфейса система готова к промышленному запуску. От пользователя получено письменное или устное подтверждение, что система выполняет все необходимое и может использоваться как основное рабочее средство.
18. **Этап развёртывания и релиза (7–8 месяц):** Официальный запуск «Карго Менеджер Лисёнок» в эксплуатацию.
19. 📌 *Задачи:* Развернуть серверную часть на выбранной площадке (например, облачный VPS или локальный офисный сервер). Настроить резервное копирование БД на уровне сервера (например, ежедневный дамп на удаленное хранилище). Установить десктоп-приложение на компьютер Лисёнка (если нужен оффлайн-клиент) – либо настроить удобный доступ через веб-браузер. Все конфигурации (токены, API-ключи, пароли) безопасно внести в систему.
20. Провести **обучение и документацию:** хотя Лисёнок уже знаком с системой, подготовить краткое руководство пользователя: например, PDF с описанием основных операций, на случай расширения штата или чтобы она могла освежить в памяти редкие функции. Также передать техническую документацию (структура БД, описание модулей) – это будет полезно для поддержки и развития (возможно, документация в формате README или Wiki в репозитории).
21. **Резервный план:** Обговорить, что делать в случае сбоев – кто техподдержка, какой канал связи. Настроить мониторинг (например, оповещение разработчиков, если сервер лег или бот отключился).
22. Официально переключить процессы: теперь все новые заказы ведутся только через «Карго Менеджер Лисёнок». Старые Excel можно импортировать (если есть функционал миграции, можно выполнить, либо хранить старые как архив).
23. 💾 *Результаты:* **Версия 1.0 – релиз.** Система запущена, пользователь работает в ней ежедневно. Проект считается выполненным.
24. *(Опционально)* **Этап поддержки и дальнейшего развития (пост-релизный):** После релиза обычно следует период сопровождения.
25. На протяжении, например, 1–3 месяцев разработчики поддерживают систему: быстро чинят возникшие неполадки (если вдруг вылезут при масштабном использовании), отвечают на вопросы пользователя.
26. Параллельно можно планировать **новые фичи**: мобильное приложение для Лисёнка (чтобы отслеживать грузы и общаться на ходу), личный кабинет для клиентов (чтобы они сами могли создавать заявки и смотреть статусы), интеграция с платежными системами (например, автоматическое подтверждение оплаты счёта) и прочее. Эти улучшения оформляются отдельным планом (версии 1.1, 2.0 и т.д.) и реализуются отдельными итерациями.

Данный план рассчитан на ~8 месяцев до полноценного релиза. Он гибкий: по ходу проекта возможны корректировки (Agile-подход с итерациями разработки на этапе 2 и 3, с регулярным демонстрацией результата пользователю). Важнейшим показателем успеха будет то, что к финалу Лисёнок полностью переходит с разрозненных инструментов (Excel, переписки) на единую систему и ощущает реальную экономию времени и рост эффективности. Этот результат достижим при следовании изложенному плану и тесном сотрудничестве с конечным пользователем на всех этапах проекта. [[42]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%9E%D0%B1%D0%BD%D0%BE%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D1%8B%20%D0%B3%D1%80%D1%83%D0%B7%D0%B0)[[32]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1,%E2%86%92%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0%20%D1%88%D0%BB%D0%B5%D1%82%20%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F)

[[1]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,REAL%20NOT%20NULL%2C%20hs_code%20TEXT) [[2]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=self.root.title%28%22Cargo%20Manager%20%D0%9B%D0%B8%D1%81%D1%91%D0%BD%D0%BE%D0%BA%20,1000x700) [[3]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%A5%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8%20%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9%20%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%B2) [[4]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%20%D1%83%D0%BF%D1%80%D0%BE%D1%89%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%BE%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) [[5]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%96%2F%D0%94) [[6]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%A0%D0%B5%D0%BA%D0%BE%D0%BC%D0%B5%D0%BD%D0%B4%D0%B0%D1%86%D0%B8%D0%B8%20%D0%BF%D0%BE%20%D0%BE%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D1%8E%20%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D1%8B%D1%85%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B9) [[7]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%92%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20%D0%BC%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B0%20%D0%BD%D0%B0%20%D0%BA%D0%B0%D1%80%D1%82%D0%B5) [[8]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%20%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D1%85%20%D0%B8%20%D0%BF%D0%BB%D0%B0%D0%BD%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D1%85%20%D1%81%D1%80%D0%BE%D0%BA%D0%BE%D0%B2) [[9]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%2A%2A%D0%9A%D0%B0%D0%BA%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%B5%D1%82%3A%2A%2A%20,%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%B0%D0%BC%D0%B8%20%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B8%D1%80%D1%83%D1%8E%D1%82%D1%81%D1%8F%20%D1%81%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B9%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [[10]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=3,%D0%BC%D0%B3%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%20%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B8%D1%82%D1%8C%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8E%20%D1%83%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%B0) [[11]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=4.%20%D0%9F%D1%80%D0%BE%D1%84%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%B8%D0%BC%D0%B8%D0%B4%D0%B6%20,%D0%9B%D0%B8%D1%81%D1%91%D0%BD%D0%BE%D0%BA%20%D0%B2%D1%81%D0%B5%D0%B3%D0%B4%D0%B0%20%D0%B2%20%D0%BA%D1%83%D1%80%D1%81%D0%B5%20%D0%B4%D0%B5%D0%BB) [[12]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%B8%D0%B9%202%3A%20%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0%20WeChat%20%D0%B4%D0%BB%D1%8F,%D0%BE%D1%82%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%BE%D0%B2%20%D0%BF%D0%BE%D1%8F%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F%20%D0%B2%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B8) [[13]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%BC%D0%B3%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%20%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%B8%D1%82%D1%8C%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8E%20%D1%83%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%B0) [[14]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1,%D0%B1%D1%8B%D1%81%D1%82%D1%80%D0%BE%D0%B3%D0%BE%20%D0%B4%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) [[15]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,WeChat%20%D0%BE%D1%82%20Shenzhen%20Electronics) [[16]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1,%E2%86%92%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0%20%D1%88%D0%BB%D0%B5%D1%82%20%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F) [[17]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,headings) [[18]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,7) [[19]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,w) [[20]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,d) [[21]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%F0%9F%8C%90%20%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%B8%20%D0%B8%20%D0%B2%D0%BD%D0%B5%D1%88%D0%BD%D0%B8%D0%B5%20%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D1%8B) [[22]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%2A%2A%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8%3A%2A%2A%20,%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%B0%20%D0%BD%D0%B0%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F%20%D1%87%D0%B5%D1%80%D0%B5%D0%B7%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) [[23]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%B8%D0%B9%201%3A%20%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0%20Telegram%20%D0%B4%D0%BB%D1%8F,%D0%BE%D1%82%20%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%BE%D0%B2%20%D0%BF%D0%BE%D1%8F%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F%20%D0%B2%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B8) [[24]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1.%20%D0%AF%D0%B4%D1%80%D0%BE%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B%20%28Core%29%20,%D0%A0%D0%B5%D0%B7%D0%B5%D1%80%D0%B2%D0%BD%D0%BE%D0%B5%20%D0%BA%D0%BE%D0%BF%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%B8%20%D0%B2%D0%BE%D1%81%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [[25]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%9E%D0%B1%D0%BD%D0%BE%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%20%D0%BE%D0%B1%D1%89%D0%B8%D0%B9%20%D1%80%D0%B0%D1%81%D1%87%D0%B5%D1%82%20%D0%B2%20%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B5) [[26]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%9F%D0%BE%D0%B4%D0%B4%D0%B5%D1%80%D0%B6%D0%BA%D0%B0%20%D0%BA%D0%B8%D1%82%D0%B0%D0%B9%D1%81%D0%BA%D0%BE%D0%B3%D0%BE%20%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0%20%D0%B8%20%D1%82%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8) [[27]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%A3%D0%B2%D0%B5%D0%B4%D0%BE%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%3A%20,%D0%9F%D0%B5%D1%80%D0%B5%D1%81%D1%87%D0%B8%D1%82%D0%B0%D1%82%D1%8C%20%D0%B2%D1%81%D0%B5%20%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5%20%D0%B7%D0%B0%D1%8F%D0%B2%D0%BA%D0%B8) [[28]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=def%20handle_telegram_message%28update%2C%20context%29%3A%20,id) [[29]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,id) [[30]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%2A%2A%D0%9A%D0%B0%D0%BA%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%B5%D1%82%3A%2A%2A%20,%D1%84%D0%B0%D0%B9%D0%BB%D1%8B%20%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B8%D1%80%D1%83%D1%8E%D1%82%D1%81%D1%8F%20%D1%81%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B9%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [[31]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%9E%D0%B1%D0%BD%D0%BE%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D1%8B%20%D0%B3%D1%80%D1%83%D0%B7%D0%B0) [[32]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=1,%E2%86%92%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0%20%D1%88%D0%BB%D0%B5%D1%82%20%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F) [[33]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%98%D0%BD%D0%B8%D1%86%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20%D0%B1%D0%B0%D0%B7%D1%8B%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%20def%20init_db,cursor) [[34]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=3,from%20%27..%2Fstore) [[35]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=switch%20%28data.type%29%20,%29%3B%20break) [[36]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=4.%20%D0%A1%D0%BA%D0%B0%D0%BD%D0%B8%D1%80%D1%83%D0%B5%D1%82%20QR,%D0%BE%D1%82%20%D0%BF%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D1%89%D0%B8%D0%BA%D0%BE%D0%B2%20%D0%BF%D0%BE%D1%8F%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F%20%D0%B2%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B8) [[37]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,self.status_var.get%28%29) [[38]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=messagebox.showerror%28,return) [[39]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=ttk.Button%28btn_frame%2C%20text%3D,command%3Dself.clear_order_form%29.pack%28side%3Dtk.LEFT%2C%20padx%3D5) [[40]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=4.%20%2A%2A%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%2A%2A%3A%20,%D0%B2%D0%B0%D0%BB%D1%8E%D1%82%20%D0%B2%20%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%20%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8) [[41]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=%D0%9F%D1%80%D0%BE%D0%BC%D1%82%20%D0%BF%D0%BE%D0%B4%D1%80%D0%BE%D0%B1%D0%BD%D0%BE%20%D0%BE%D0%BF%D0%B8%D1%81%D1%8B%D0%B2%D0%B0%D0%B5%D1%82%3A%201,%D0%9A%D0%B0%D0%BA%20%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D0%B8%D1%82%D1%8C%20%D0%BA%D0%B0%D0%B6%D0%B4%D1%83%D1%8E%20%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8E) [[42]](file://file-P2yxhRfVZVuKd4uWDL5Eba#:~:text=,%D0%9E%D0%B1%D0%BD%D0%BE%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D1%8B%20%D0%B3%D1%80%D1%83%D0%B7%D0%B0) chat-Cargo Management Solution.txt

<file://file-P2yxhRfVZVuKd4uWDL5Eba>