МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Разработка программного обеспечения
(направление (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка программного обеспечения для оптимизации работы сотрудников курьерской службы»

Обучающийся Д.В. Камышанов _____
(Инициалы Фамилия) (личная подпись)

Руководитель к.т.н., доцент, О.В. Аникина
ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

В данной работе рассматривается задача разработки программного обеспечения для автоматизации и оптимизации процессов курьерской службы. Курьерские службы сталкиваются с необходимостью оперативного и точного выполнения задач, связанных с доставкой посылок, планированием маршрутов и координацией работы сотрудников. Разработка программного обеспечения, ориентированного на решение этих задач, позволяет значительно повысить производительность, минимизировать ошибки и улучшить качество обслуживания клиентов.

Актуальность исследования обусловлена потребностью в повышении эффективности работы курьерских служб, особенно в условиях растущего объема заказов и возросших требований к скорости и качеству доставки. Основной целью работы является создание автоматизированной информационной системы (АИС), которая облегчит и ускорит выполнение повседневных задач сотрудников, таких как управление заказами и отслеживание маршрутов.

В структуру работы входит:

- введение, где раскрываются актуальность темы, формулируется цель исследования и задачи разработки программного обеспечения;
- функциональное моделирование предметной области, где производится технико-экономическое описание предметной области, концептуальное моделирование области, анализ существующих аналогов на предмет соответствия требования и постановка задачи на разработку проекта создания АИС;
- логическое проектирование АИС, в которой производится выбор технологии логического моделирования, построение логической модели АИС и проектирование БД;

– физическое проектирование АИС, где производится выбор архитектуры АИС, выбор технологи разработки ПО, разработка физической модели данных и непосредственно самой АИС, а также тестирование готового продукта;
– заключение, содержащее выводы о достигнутых результатах, обоснование эффективности решения и предложения по дальнейшему развитию системы.

Работа состоит из 70 страниц, включает 6 таблиц и 24 рисунка, 7 приложений к ВКР. В процессе разработки были использованы инструменты для программирования, такие как Kotlin и C# для клиентских приложений, Node.js для серверной части и MongoDB для хранения данных.

В ходе работы были решены следующие задачи:

– разработана архитектура АИС на основе клиент-серверной модели;
– спроектирована база данных для хранения информации о заказах и клиентах;
– реализовано программное обеспечение для автоматизации работы сотрудников, включая мобильное приложение для создания заказов и офисное приложение для обработки заявок;
– проведено тестирование и проверка работоспособности системы.

Основным результатом работы является программный продукт, способный автоматизировать задачи, связанные с доставкой посылок, что позволяет сократить время обработки заказов и повысить точность выполнения доставок.

Содержание

Введение

В условиях стремительного роста объемов перевозок и услуг по доставке актуальной задачей для курьерских служб становится оптимизация процессов обработки заказов, планирования маршрутов и координации действий сотрудников. Разработка специализированного программного обеспечения для автоматизации работы сотрудников курьерской службы является необходимым условием повышения их производительности и сокращения времени выполнения задач.

Актуальность выбранной темы обусловлена потребностью в создании эффективных инструментов для автоматизации курьерских служб, что позволит сократить ручной труд, повысить точность обработки заказов, а также обеспечить более высокое качество обслуживания клиентов. Внедрение современных информационных технологий в процессы доставки также способствует увеличению скорости и эффективности работы предприятия.

Объектом исследования является деятельность курьерской службы, связанная с обработкой заказов и взаимодействием с клиентами. Предмет исследования — автоматизация процессов курьерской службы через внедрение программного обеспечения для управления заказами и маршрутизацией.

Целью работы является разработка автоматизированной информационной системы (АИС), которая обеспечит оптимизацию работы сотрудников курьерской службы. Для достижения этой цели необходимо решить следующие задачи:

- изучить современные технологии и средства разработки информационных систем;
- проанализировать и выбрать оптимальные инструменты для реализации АИС;
- спроектировать и разработать базу данных для хранения информации о заказах и клиентах;

– реализовать клиентскую часть системы для управления заказами;
– провести тестирование программного обеспечения и оценить его эффективность.

Основные решения, выносимые на защиту, включают выбор архитектуры клиент-серверного приложения с использованием технологий MongoDB, Node.js и Kotlin, а также разработку базы данных, серверной части и клиентского приложения для мобильных устройств и настольных компьютеров.

Методами исследования в работе использовались объектно-ориентированный анализ и проектирование, методология разработки клиент-серверных приложений, а также методы системного проектирования информационных систем.

Практическая значимость заключается в разработке программного обеспечения, которое позволяет автоматизировать ключевые процессы курьерской службы, включая управление заказами, маршрутизацию и отслеживание состояния доставок. Это программное обеспечение может быть внедрено в курьерские компании для повышения эффективности их работы.

Первый раздел посвящен функциональному моделированию предметной области.

Во втором разделе было выполнено логическое проектирование АИС.

Третий раздел посвящен реализации программного обеспечения и выполнению тестирования разработанной АИС.

Заключение содержит обобщенные выводы по проделанной работе и направления для дальнейшего совершенствования системы.

Таким образом, данная работа представляет собой полноценный проект разработки и внедрения программного обеспечения для курьерской службы, направленного на автоматизацию рабочих процессов и повышение эффективности работы сотрудников.

1 Функциональное моделирование предметной области

1.1 Технико-экономическая характеристика предметной области

Курьерские службы играют ключевую роль в современной экономике, обеспечивая быструю и надежную доставку товаров и документов. В условиях растущей конкуренции и увеличивающегося объема заказов важно оптимизировать процессы, чтобы повысить эффективность и снизить затраты.

В рамках выпускной квалификационной работы было придумано предприятие, выполняющее функции доставки грузов от отправителя получателю. Для описания технико-экономических характеристик нужно учитывать несколько аспектов предприятия:

- формат оказываемых предприятием услуг. Предприятие выполняет доставку некрупногабаритных грузов от отправителя получателю такие как: документы, бандероли, письма, некрупные посылки;
- условия предлагаемых услуг. Курьерская служба работает в режимах моментальных заказов. То есть можно либо заказать доставку и ближайший свободный курьер в порядке очереди выполняет услуги по доставке посылок;
- процедура оказания услуг. Менеджер принимает заказ по телефону, проводит расчет стоимости доставки, оговаривает сумму с клиентом, фиксирует необходимые данные, потом передает их курьеру;
- организация курьерской службы. Для выполнения курьерских заказов используется свой штат курьеров.

Так как данной курьерской службы не существует, то необходимо с нуля охарактеризовать данное предприятие. Для описания предприятия необходимо составить организационную структуру. На рисунке 1 изображена организационная структура предприятия.

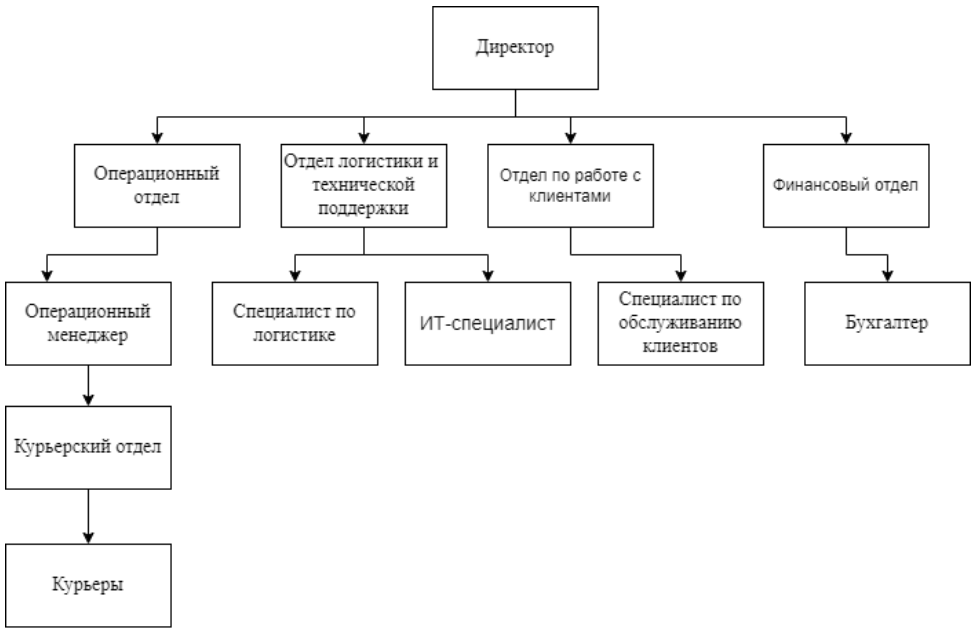

Рисунок 1 – Организационная структура предприятия

Директор предприятия выполняет функции по руководству компании. Принимает ключевые решения в вопросах развития компании.

Операционный менеджер выполняет следующие функции: организация и контроль ежедневных доставок, оптимизация процессов, руководство командой курьеров и оптимизация из работы, обучение и развитие сотрудников. Также операционный менеджер выполняет функции по контролю качества обслуживания, такие как работа с жалобами.

Курьеры выполняют функции по принятию и доставке грузов, отчетность о выполнении заказа.

Специалист по логистике выполняет следующие функции: оптимизация маршрутов, контроль процесса отгрузки и доставки, планирование и распределение курьеров, сбор и анализ данных для улучшения качества доставки, внедрение технологий для отслеживания доставки.

ИТ-специалист занимается задачами: поддержка и развитие ИТ-инфраструктуры, разработка и поддержка приложений/веб-сайтов компании.

Специалист по обслуживанию клиентов выполняет следующие функции: отвечает на звонки и почту от клиентов, консультирует по вопросам

услуг и статуса заказа, решает вопросы с доставкой и управляет жалобами клиентов, передает важные данные о клиентах в другие отделы, собирает отзывы клиентов для улучшения качества обслуживания, подтверждает заказы на доставку.

Бухгалтер выполняет функции: ведение бухучета, обработка счетов, квитанций и других финансовых документов, готовит и контролирует бюджет компании, анализ финансовых показателей, подготовка финансовых отчетов для руководства и составление налоговых деклараций, контроль затрат, расчеты с контрагентами.

Работа курьерской службы выполняется следующим образом:

– прием заказов по мобильному телефон;

– специалист по логистике оптимизирует маршрут;

– назначается курьер и направляется, чтобы забрать посылку из пункта отправления;

– курьер доставляет посылку в пункт назначения

– клиент подтверждает получение посылки.

После того как технико-экономическая характеристика предприятия была описана, можно приступать к концептуальному моделированию предметной области.

1.2 Концептуальное моделирование предметной области

В этой главе рассматривается концептуальное моделирование предметной области курьерской службы. Модель будет служить основой для разработки программного обеспечения для оптимизации работы курьерской службы.

Начнем с моделирования процессов организации «как есть». Строить диаграммы в рамках выпускной работы я буду в нотации IDEF0 с помощью программного обеспечения Ramus.

Модель выполнения заказа «как есть» изображена на рисунке 2 в виде контекстной диаграммы концептуальной модели выполнения заказа курьерской службой в нотации IDEF0.

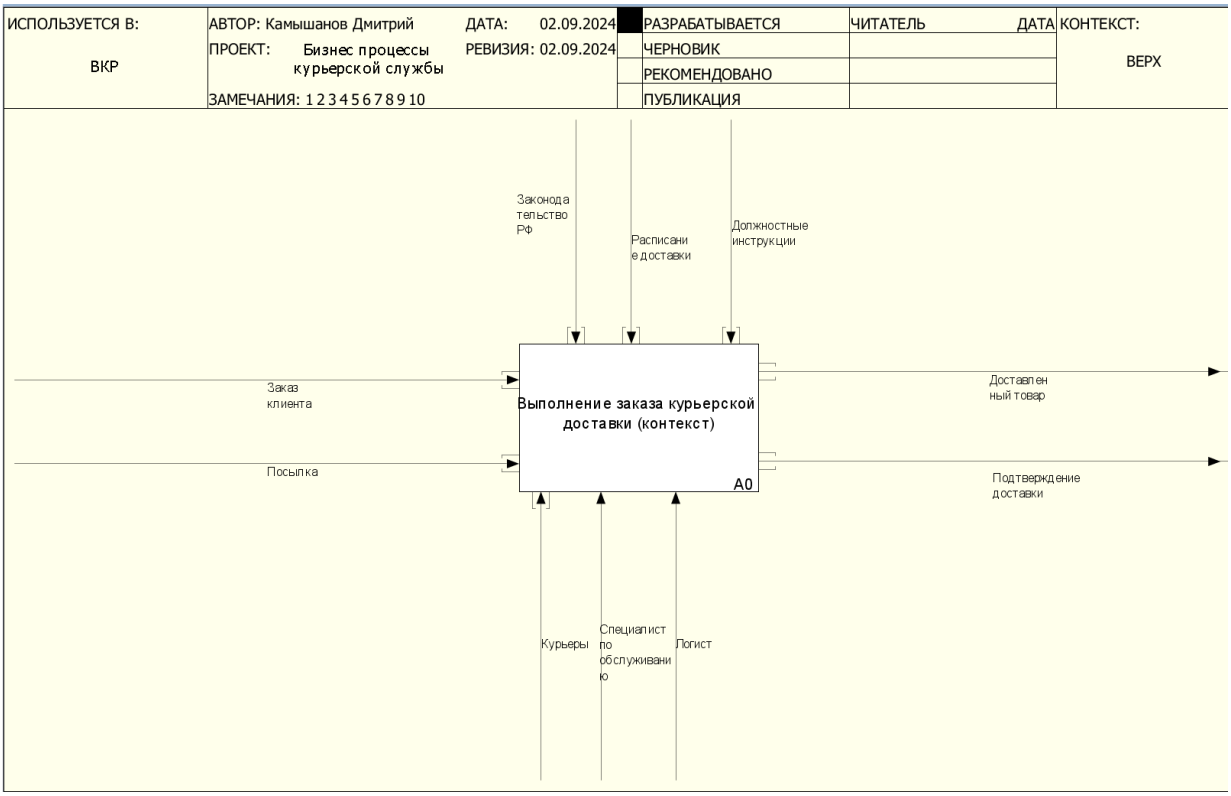

Рисунок 2 – Контекстная диаграмма процесса выполнения заказа «как есть»

На рисунке 3 выполнена декомпозиция первого уровня процесса выполнения курьерской доставки.

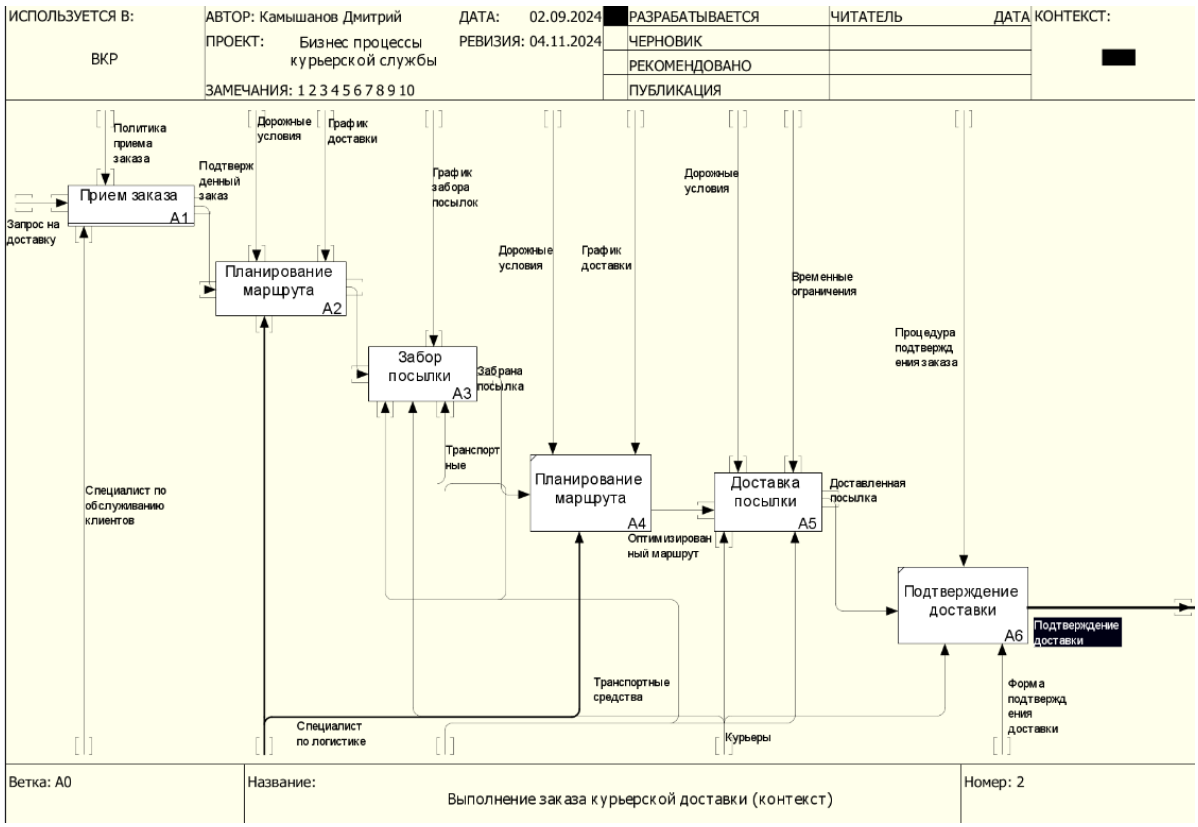

Рисунок 3 – Декомпозиция первого уровня процесса выполнения заказа

Декомпозиция второго уровня будет приложена в приложении А. Рисунки А1 - А6.

По телефону клиент связывается с курьерской службой для создания заказа, специалист по обслуживанию клиентов проверяет, подтверждает заказ, вручную производит расчет стоимости заказа, согласовывает данные и стоимость с клиентом и подтверждает заказ.

Логист собирает информацию о заказах, производит анализ географического положения заказов, определяет оптимальный маршрут, утверждает его и уведомляет курьеров.

Курьеры предварительно звонят отправителю, согласовывают время направляются к отправителю, чтобы забрать посылку, проверяют ее на месте, забирают ее, получают подпись о заборе посылки с отправителя. Далее согласовывают время с получателем по телефону, доставляют посылку,

подтверждают ее передачу с помощью подписи получателя, получают оплату, возвращаются обратно.

Проанализировав данный процесс можно заметить, что есть огромный объем ручной работы, которую следует автоматизировать, чтобы сократить количество ошибок. Также автоматизировав процессы можно сократить время обработки заказов и снизить нагрузку на отделы логистики и обслуживания клиентов. Дополнительно можно улучшить взаимодействие заказчиков с сервисов путем уменьшения времени коммуникации, создав приложение для клиентов, в котором заказчик может сам указать адрес отправки и получения, а также отслеживать расположение курьера.

Так как в описанной компании не внедрены АИС, то стоит задуматься об их внедрении.

Исходя из описания бизнеса, а также оценки слабых мест [20], можно выделить следующие требования к проектируемому АИС:

- клиент-серверное приложение с ролевой системой для сотрудника бизнеса, клиент;
- клиентское приложение, в котором можно создать заказ;
- должен отображаться статус заказа;
- должно отображаться местоположение курьера у сотрудника бизнеса и клиента;
- в клиенте для создания заказа требуется указать точку отправления и точку назначения для создания заказа;
- в приложении должна быть система аккаунтов, где есть контактные данные клиентов;
- у сотрудника бизнеса должна быть возможность подтвердить заказ без контакта с клиентом;
- сотруднику бизнеса должно быть доступны контактные данные клиента для уточнения данных, в случае ошибок;
- АИС должно автоматически строить маршрут для курьеров;

– должна быть возможность отменить заказ;
– автоматический расчет стоимости маршрута при создании заказа.

1.3 Анализ существующих аналогов на предмет соответствия сформулированным требованиям

В интернете есть много готовых решений для автоматизации требуемых задач. Эти комплексы решают как глобальные так и точечные задачи бизнеса.

Решение mobidel [10] - это программный комплекс для автоматизации службы доставки. Решение предлагает несколько решений: клиентское приложение, операторское приложение, приложение для курьеров и приложение для руководителей.

Самая дешевая версия предлагает CRM систему за 490 р в месяц за 450 заказов и по 14 р за заказ для клиентского приложения. По описанию продукта данное АИС подходит для курьерских служб разного размера.

Из недостатков, данное АИС рассчитано на очень широкий круг доставок и интернет магазинов, поэтому будет иметь много функций, которые будут не нужны бизнесу. Например, интеграции с магазинами.

Measoft [9] предлагает большой пакет функций и автоматизаций для курьерской службы. Как дополнительный плюс – имеет гибкую настройку предоставляемого функционала, например, можно не покупать модуль складского учета, если он не нужен. Систему позволяется купить или взять в аренду.

Ниже приведена сводная таблица аналогов, найденных в сети интернет.

Таблица 1 – Сводная таблица аналогов АИС, подходящих для проекта

Требование/Аналог	Mobidel	MeaSoft
Система ролей	+	+
Создание заказа	+	+
Местоположение курьера	+	+ (только для сотрудников)
Создание заказа с помощью добавления точек отправления и назначения	-	-
Аккаунты с контактными данными	+	+
Бесконтактное ведение заказа	-	-
Автопостроение маршрута	-	-
Статус заказа	+	-
Отмена заказа	+	-
Автоматический расчет стоимости	-	+
Доступ к контактным данным у сотрудников	+	+

Оценив готовые решения из сети интернет можно сделать вывод, что это дорогостоящие решения, которые не соответствуют описанным выше требованиям, а также имеют ряд функций, которые на данном этапе будут лишними. В среднем данные решения имеют подписочный сервис и стоят около 200 тысяч рублей за годовую поддержку. Чтобы точечно решить проблемы данного бизнеса нужна собственная разработка, которая будет соответствовать требованиям, которые были описаны выше.

1.4 Постановка задачи на разработку проекта создания АИС

1.4.1 Цель и назначение автоматизированного варианта решения задачи

Проанализировав основные недостатки в бизнес-процессах, а также требования к АИС, можно выделить назначение автоматизированного варианта решения задачи:

- увеличить скорость создания заказа;
- увеличить скорость принятия заказа;
- сократить количество ручной работы при принятии заказа, планировании маршрута и информировании о статусе заказа;
- сократить количество телефонной связи с клиентом;
- упростить получение контактных данных по заказу;
- сократить количество ошибок при расчете стоимости доставки;
- ведение базы заказов в электронном виде;
- улучшение опыта взаимодействия заказчика с компанией.

1.4.2 Требования к функциональности АИС

Чтобы выполнить постановку задачи, нужно ответить на следующие вопросы:

Изменения в функциях подразделения, связанных с обработкой и выдачей информации. После внедрения АИС изменится процесс создания и принятия заказа, что исключит ручную работу в большинстве случаев во время создания заказа, его принятия, а также планирования маршрута.

Источники поступления оперативно и условно-постоянной информации и периодичность ее поступления. Для начала нужно разобраться в терминах «оперативная» и «условно-постоянная информация. Оперативной принято считать информацию, которая часто меняется и требует постоянного обновления. Обычно используется для решений в реальном времени, таких как: местоположение курьеров, статус заказов, а также новые заказы. Условно-постоянная информация обновляется значительно реже и эти данные остаются относительно стабильными в течение какого-то времени. Обычно используются для стратегического планирования и анализа. Примеры данных: данные о клиентах, расписания курьеров, исторические данные доставок. Отвечая на этот вопрос, выделяем следующие источники поступления оперативной информации: клиентское приложение для курьеров будет предоставлять данные о местоположении курьеров в реальном времени. Также

через приложение курьеров обновляется информация о статусе заказов. Эти данные обновляются при изменении статуса вручную. Источник заказов – клиентское приложение. Периодичность поступления – в реальном времени. Источники поступления условно-постоянной информации: Информация о клиентах: будет поступать из клиентского приложения при создании учетной записи и создании заказов. Обновляется силами клиентов. Исторические данные о заказах обновляются с целью анализа и отчетности с разной периодичностью.

Краткая характеристика системы ведения файлов в базе данных (перечень файлов с условно-постоянной и оперативной информацией, периодичность обновления, требования защиты целостности и секретности); Отвечая на данный вопрос можно выделить перечни файлов:

Условно-постоянная информация:

- данные о клиентах (имена, адреса, контактная информация);
- данные о курьерах (имена, контактные данные, график);
- сведения о тарифах;
- история заказов;

Оперативная информация:

- текущие заказы (данные о заказах, которые на данный момент находятся в процессе выполнения);
- местоположение курьеров;
- статусы доставки (принят, в пути, доставлен).

Периодичность обновления для условно-постоянных данных – примерно раз в несколько месяцев – полгода. Обычно связано с изменениями данных. Для оперативной информации – обновление в реальном времени, т.к. актуальность данной информации напрямую влияет на качество предлагаемых услуг.

Можно выделить основные требования защиты целостности и секретности данных:

– контроль доступа;
– резервное копирование;
– постоянный мониторинг доступа к данным;
– режим решения задачи (пакетный, диалоговый, с использованием методов телеобработки или смешанный).

Для решения задачи стоит использовать смешанный режим.

Этот режим объединяет преимущества пакетного, диалогового режимов и методов телеобработки, обеспечивая гибкость и эффективность в решении различных аспектов задачи.

Автоматизация рутинных задач: Пакетный режим используется для обработки данных и автоматического распределения заказов.

Интерактивное взаимодействие: Диалоговый режим обеспечивает удобный интерфейс для операторов и клиентов для ввода и отслеживания информации.

Удалённый доступ: Методы телеобработки позволяют курьерам и менеджерам работать эффективно, находясь в любом месте.

1.4.3 Требование к архитектуре реализации АИС

Для обеспечения эффективного функционирования и адаптации к особенностям ведения производственной деятельности, а также интеграции в ИТ-инфраструктуру предприятия, предлагаемая АИС должна удовлетворять следующим требованиям к архитектуре:

– простота адаптации: Архитектура АИС должна обеспечивать простоту адаптации к особенностям ведения производственной деятельности, включая возможность настройки под различные модификации предметной технологии;
– интеграция в ИТ-инфраструктуру: АИС должна быть способна интегрироваться в существующую ИТ-инфраструктуру предприятия, включая взаимодействие с другими информационными системами и базами данных;

- использование веб-технологий: Архитектура АИС должна предусматривать использование веб-технологий для обеспечения доступа к системе из любой точки, где есть подключение к интернету;
- использование бесплатно распространяемых средств разработки: Для снижения затрат на разработку и поддержку АИС рекомендуется использовать бесплатно распространяемые средства разработки программного обеспечения и СУБД;
- гибкость и настраиваемость: Проектируемое программное средство должно быть гибким и настраиваемым на различные модификации предметной технологии. Механизм настройки должен быть интуитивно понятным и не требовать глубоких знаний в области программирования;
- масштабируемость: Архитектура АИС должна обеспечивать возможность масштабирования системы в соответствии с ростом потребностей предприятия.
- безопасность: АИС должна обеспечивать высокий уровень безопасности данных, включая защиту от несанкционированного доступа и шифрование передаваемой информации;
- возможность внедрения на аналогичных объектах управления. Проектируемая АИС должна быть универсальной и адаптируемой для внедрения на аналогичных объектах управления.

1.5 Разработка модели бизнес-процесса «как должно быть»

Анализ бизнес-процессов «как есть», а также а также представленная в графическом виде нотация в IDEF0 послужит основой для построения модели бизнес-процесса «как должно быть». Контекстная диаграмма бизнес–процесса «Как должно быть» в нотации IDEF0 представлена на рисунке 4, ее декомпозиция – на рисунке 5.

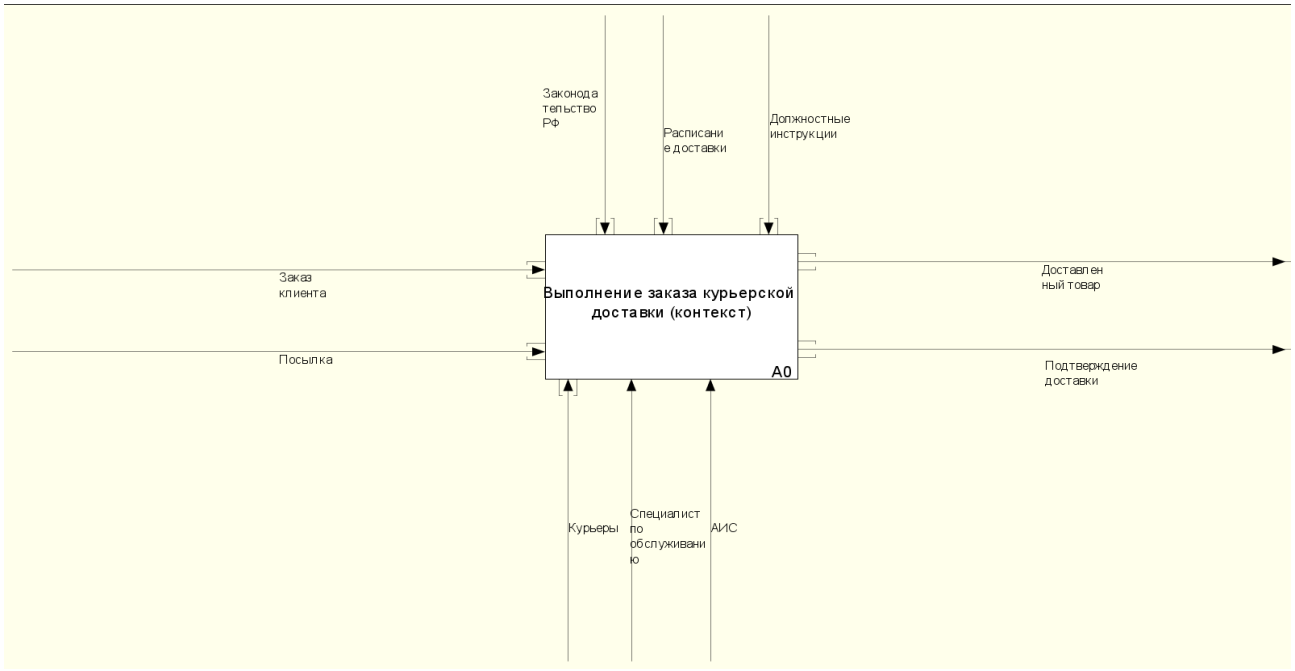

Рисунок 4 – Контекстная диаграмма «Как должно быть»

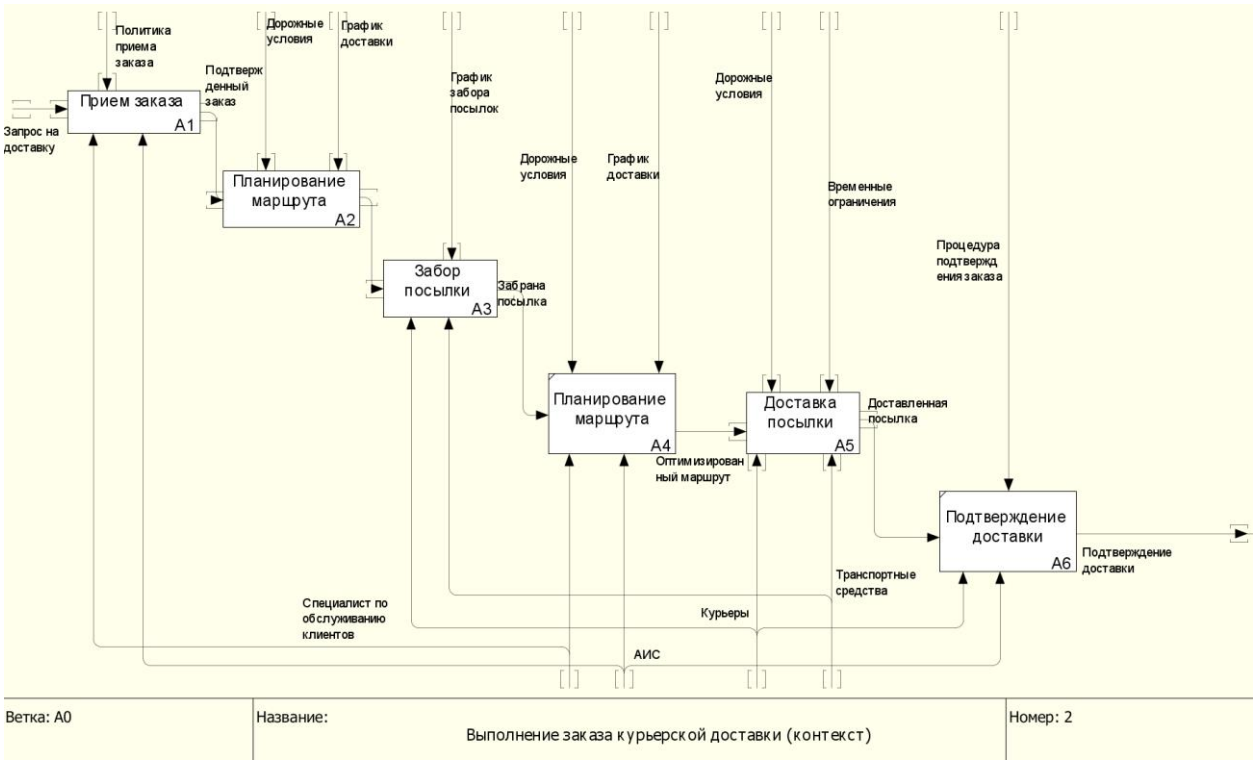

Рисунок 5 – Декомпозиция процесса «Выполнения заказа курьерской доставки»

Выводы по разделу 1

В разделе 1 была исследована предметная область курьерской службы доставки.

С учетом созданных концептуальных моделей, требованиям к разрабатываемому ПО, а также анализа готовых решений в сети интернет были сформулированы основные требования к разрабатываемому АИС.

Также анализ процесса «выполнения заказа курьерской доставки» показал неэффективность в связи с большим количеством ручной работы и большого количества взаимодействия с клиентами.

Анализ готовых решений из сети Интернет показал, что покупка готового решения для малого предприятия нецелесообразна в связи с дороговизной решения, и сложной адаптируемостью под точечные цели предприятия. Было принято решение выполнить собственную разработку АИС.

2 Логическое проектирование АИС

2.1 Выбор технологии логического моделирования

Логическое моделирование информационной системы представляет собой ключевой этап в проектировании автоматизированной системы. На данном этапе формируется логическая структура базы данных системы. Проектируемая система автоматизации приема заказов для курьерской службы будет использовать реляционную базу данных. Для реляционной модели данных существуют формальные правила, которые позволяют преобразовать концептуальную модель предметной области, представленную в виде ER-диаграммы, в логическую схему базы данных. В процессе логического моделирования помимо разработки схемы данных создается схема отношений и проводится их нормализация.

Сегодня существует множество инструментальных средств, которые можно использовать для реализации проекта автоматизированной информационной системы, начиная с этапа анализа и заканчивая созданием программного кода. В данной выпускной квалификационной работе для логического моделирования информационной системы выбрана графическая нотация UML.

UML (Unified Modeling Language) – унифицированный язык объектно-ориентированного моделирования – включает в себя разнообразные инструменты, поддерживающие весь жизненный цикл информационной системы и позволяющие настроить и отразить особенности деятельности разработчиков различных элементов проекта [3]. Проектирование системы с использованием UML предполагает создание различных диаграмм.

Для разработки диаграмм UML в выпускной квалификационной работе использовалось программное обеспечение Star UML, которое предоставляется бесплатный доступ на 30 дней.

2.2 Логическая модель АИС и ее описание

Логическая модель автоматизированной информационной системы (АИС) описывает внутреннюю организацию и структуру системы, а также взаимодействие между ее компонентами. В данном разделе представлены логические компоненты системы, их функции и связи, а также их роль в обеспечении эффективной работы курьерской службы.

Диаграмма вариантов использования помогает визуализировать функциональные требования системы, показывая, как пользователи (актеры) взаимодействуют с системой через различные функции (варианты использования). Она упрощает понимание требований, помогает в проектировании системы, позволяет идентифицировать ключевые сценарии использования и определяет, как система будет удовлетворять потребности пользователей. На рисунке 6 изображена диаграмма вариантов использования для курьерской службы.

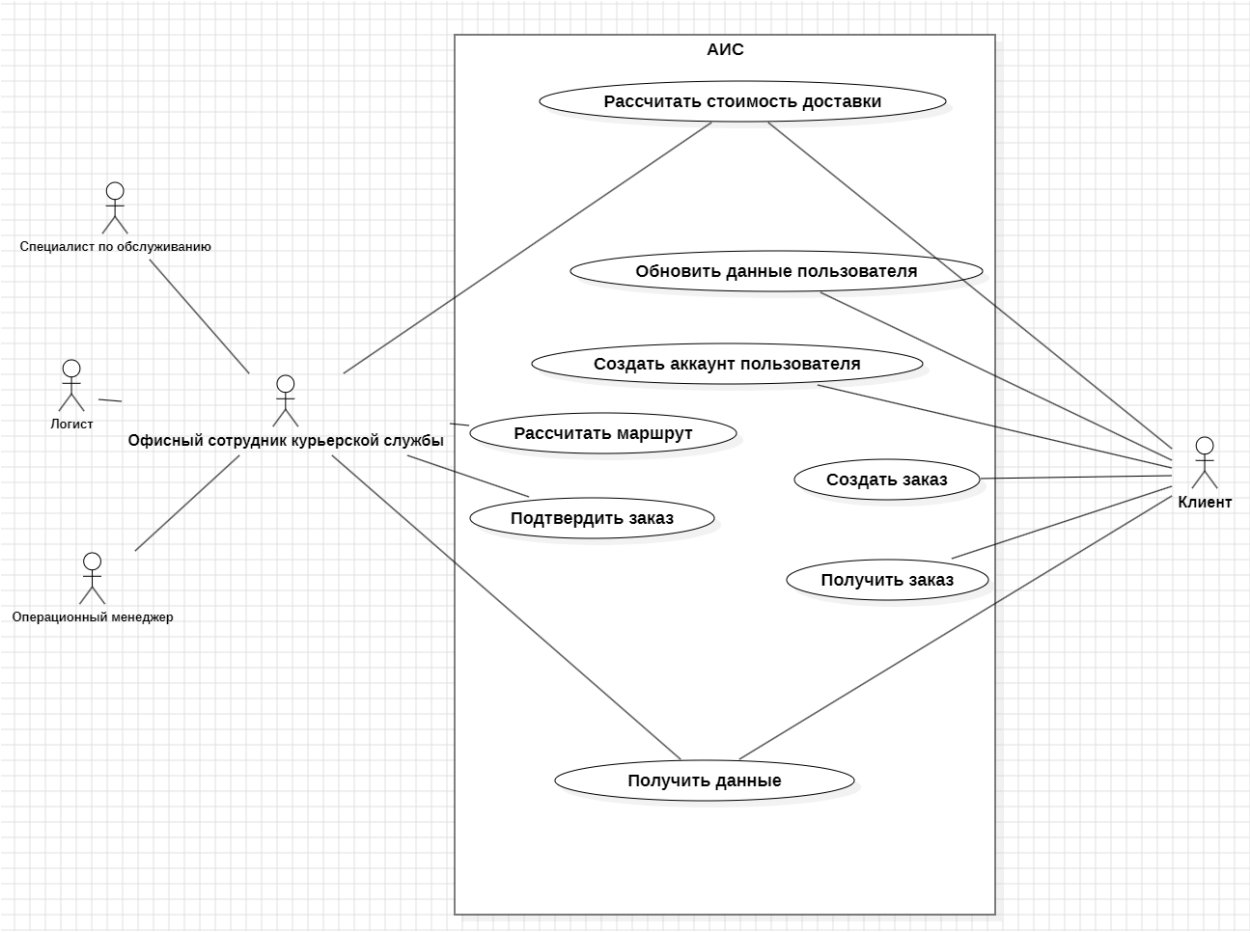

Рисунок 6 – Диаграмма вариантов использования

В предметной области выделены 3 пользователя:

- офисный сотрудник курьерской службы – сотрудники, работающие в офисе и выполняющие действия по контакту с заказчиком, оптимизации маршрутов, проверке данных;
- курьеры – сотрудники занимающиеся непосредственно доставкой посылок. Он обновляет статус заказа на актуальный;
- клиент – заказчик, согласно требованиям которого выполняется доставка. Клиент создает учетную запись, создает заказ и отправляет/получает его.

На следующем рисунке рассмотрим диаграмму классов. Диаграмма классов — это тип диаграммы в объектно-ориентированном моделировании, который отображает классы системы, их атрибуты, методы и взаимосвязи

между классами. Она помогает визуализировать структуру системы и понять, как различные компоненты взаимодействуют друг с другом.

На рисунке 7 показана диаграмм классов для АИС курьерской службы.

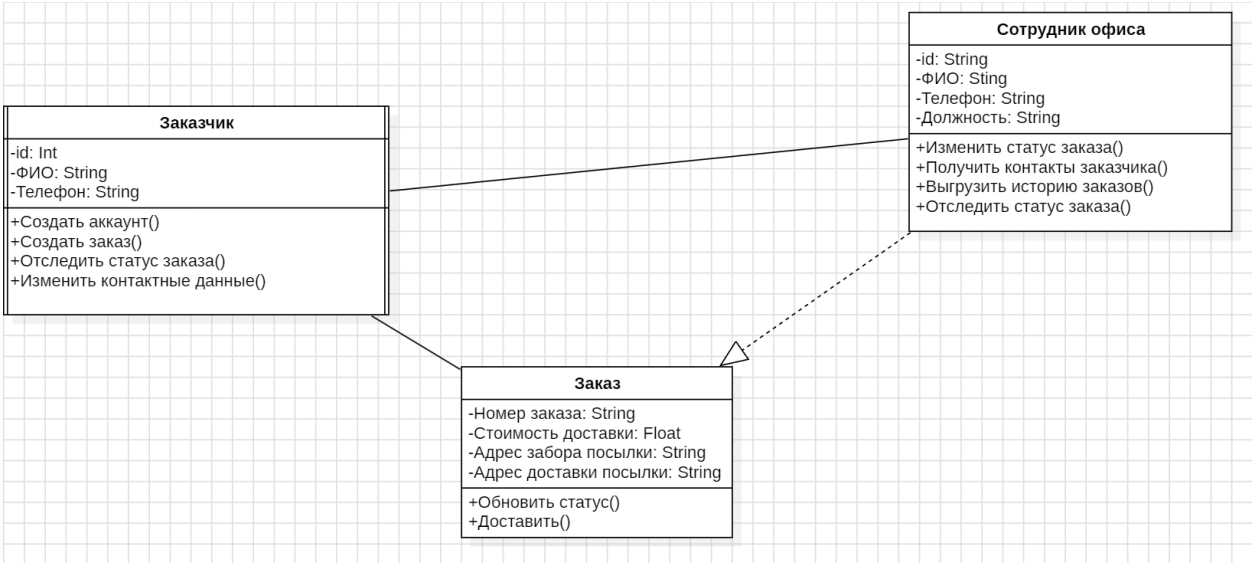

Рисунок 7 – Диаграмма классов

На представленной диаграмме показано 3 класса, каждый из которых представляет сущность предметной области.

Классы заказчик и заказ связаны отношением ассоциации. Классы сотрудник офиса и заказчик связаны отношением ассоциации. Классы сотрудник офиса и заказ связаны отношением реализация.

2.3 Информационное обеспечение АИС

Используемые классификаторы системы кодирования.

Для корректной работы автоматизированной информационной системы (АИС) доставки посылок используются классификаторы и системы кодирования, которые обеспечивают однозначную идентификацию объектов: клиентов, посылок, сотрудников, транспортных средств и маршрутов. Структура классификаторов и систем кодирования отображена в таблице 2.

Таблица 2 Структура классификаторов и систем кодирования

Наименование	Значность кода	Система кодирования	Система классификации	Вид классификатора
клиенты	12 символов	Порядковая	Иерархическая	Общесистемный
Сотрудники	8 символов	Порядковая	Иерархическая	Отраслевой

Описание классификаторов:

- клиенты: каждому клиенту присваивается уникальный идентификатор, который хранится в системе для удобного отслеживания истории заказов;
- сотрудники курьерской службы: каждому сотруднику присваивается идентификатор.

Ведение классификаторов осуществляется автоматически системой, с регулярным обновлением и проверкой данных. В приложении будут приведены примеры заполненных классификаторов [7].

Нормативно-справочная информация:

Справочники клиентов, посылок, сотрудников, транспортных средств, а также маршрутов доставки. Эти справочники обеспечивают корректную работу системы и позволяют автоматически формировать заявки на основе введенных данных.

Пример структуры:

Клиенты: ID клиента, ФИО, контактные данные.

Посылки: ID посылки, вес, габариты, категория груза, особые условия доставки.

Маршруты доставки: ID маршрута, адрес получения, адрес доставки, промежуточные точки.

Оперативная информация:

Основная оперативная информация поступает от клиента, который через интерфейс приложения заполняет заявку на доставку. Клиент указывает адрес получения и адрес доставки посылки, а также информацию о посылке (вес, размеры, особые условия).

Пример входного документа — заявка на доставку посылки:

Источник получения: интерфейс клиента в приложении или на сайте.

Файл использования: файл заказов с данными клиента и характеристиками посылки.

Структура документа: ID заказа, ФИО клиента, адрес получения, адрес доставки, параметры посылки (вес, габариты, категория).

Структура файлов оперативной информации указана в таблице 3.

Таблица 3 – Структура файлов оперативной информации

Поле	Id	Шаблон
ID клиента	user_id	Int
Координата получения посылки X	AdressAt_X	Float
Координата получения посылки Y	AdressAt_Y	Float
Координата доставки посылки X	AdressTo_X	Float
ID заказа	id	Int

Описание файлов с условно-постоянной информацией (например, справочников посылок и клиентов) включает те же поля, с добавлением данных о частоте актуализации информации.

Характеристика выходной информации:

Выходная информация включает отчеты и ведомости, которые формируются для оценки и управления процессом доставки посылок. Примеры выходных данных:

Отчет по доставленным посылкам: информация о доставленных посылках за определенный период, включая адреса получения и доставки, время доставки, данные о курьере и транспортном средстве.

Маршрутные листы: генерируются для курьеров с указанием адресов получения и доставки посылок, особенностей доставки (например, указание времени доставки или специальных требований к посылке).

Файлы с результатной информацией описываются аналогично, с указанием ключевых полей и структуры данных.

2.4 Проектирование БД АИС

Для проектирования базы данных автоматизированной информационной системы (АИС) курьерской службы доставки посылок используется методология IDEF1X — стандартная методология, применяемая для создания реляционных баз данных. Она позволяет структурировать данные таким образом, чтобы отражать реальные процессы предметной области и обеспечить эффективную работу системы. На завершающем этапе проектирования выполняется трансформация диаграммы классов в логическую модель данных, используя ER-диаграмму (Entity-Relationship), построенную по методологии IDEF1X. Это позволяет четко определить сущности, их атрибуты и взаимосвязи, а также спроектировать структуру базы данных, которая будет эффективно хранить и обрабатывать информацию.

ER-диаграмма (диаграмма "сущность-связь") представляет собой графическое отображение базы данных, где:

Сущности (entities) — это объекты реальной предметной области, которые требуют хранения данных.

Связи (relationships) — это взаимоотношения между сущностями, которые показывают, как эти объекты взаимодействуют друг с другом в реальной системе.

На рисунке 8 изображена ER-диаграмма, построенная по методологии IDEF1X на основе диаграммы классов из рисунка 9 [19]. Данная диаграмма построена в приложении StarUML.

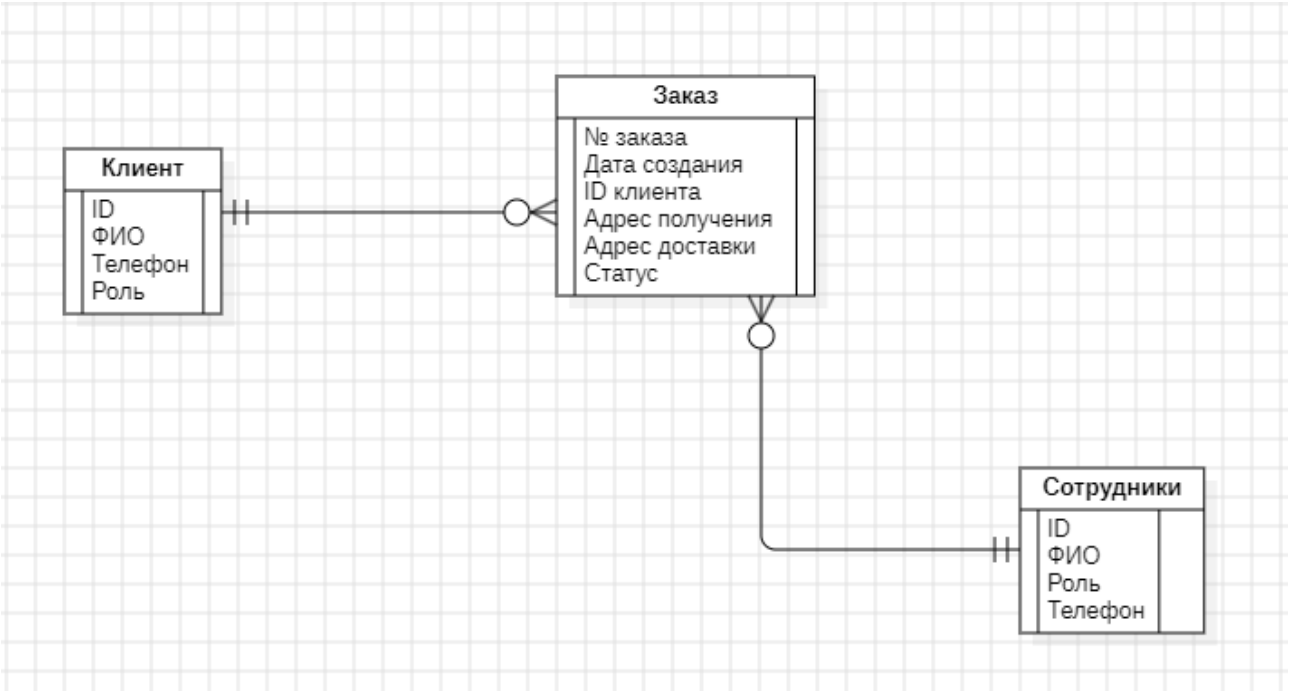

Рисунок 8 – Логическая модель данных, построенная по методологии IDEF1X

В рамках курьерской службы можно выделить следующие сущности:

– клиенты — физические или юридические лица, которые создают заявки на доставку посылок. Атрибуты: ID клиента, ФИО, Контактные данные (телефон), роль;

– заявки — данные о заказах, которые клиенты создают, указывая информацию о посылках и адресах доставки.

Атрибуты: номер заказа, дата создания, ID клиента, адрес получения, адрес доставки, статус.

Связь: Клиенты → Заявки (связь один, к многим, т.к. один клиент может создавать несколько заявок).

Атрибуты: ID маршрута, начальная точка, конечная точка, Время в пути.

Связь: Маршруты → Заявки (один к многим, т.к один маршрут может быть связан с несколькими заявками).

Клиенты создают заявки через систему, где указывают параметры доставки, такие как адрес получения и адрес доставки. Каждая заявка может содержать несколько посылок.

Сотрудники (курьеры) назначаются на выполнение заявок.

Каждая заявка обрабатывается в рамках определенного маршрута, который формируется с учетом адресов доставки нескольких посылок за одну поездку.

Эти взаимосвязи на диаграмме отражают реальные бизнес-процессы курьерской службы, где данные о клиентах, заявках, посылках, сотрудниках, транспортных средствах и маршрутах используются для управления операциями по доставке.

2.5 Требования к аппаратно-программному обеспечению АИС

Проектное решение должно быть поддержано как с технической, так и с программной точки зрения. Под техническим обеспечением подразумевается совокупность компьютеров, периферийных устройств, оргтехники и каналов связи. Учитывая, что у предприятия ранее не было развернуто решений по автоматизации то в требованиях нужно учесть несколько аспектов.

- планируется разработка десктопного приложения для сотрудников. Для приема заказа;
- планируется разработка клиентского приложения на операционную систему Android для возможности создания клиентам заказы лично.

Исходя из этих аспектов выявлены следующие требования:

Серверная часть:

Поскольку предприятие не имеет собственной серверной инфраструктуры, требуется рассмотреть внедрение сервера. Это решение

обеспечит вычислительные ресурсы для обработки запросов клиентов, хранения данных, а также автоматическое масштабирование по мере роста нагрузки.

Так как предприятие небольшое, для развертывания базы данных также предпочтительно использовать локальные решения, которые предоставляют гибкую инфраструктуру для работы с базами данных.

Мобильное приложение:

Для обеспечения работы мобильного приложения клиенты должны иметь доступ к серверной части через интернет. При отсутствии внутренней инфраструктуры особенно важно развертывание API-интерфейсов на облачном хостинге, обеспечивающем высокую доступность и низкую задержку.

Требования к серверным ресурсам можно начинать с минимальных мощностей и постепенно увеличивать их в зависимости от числа пользователей и объема запросов.

Приложение на ПК (для сотрудников):

ПК-приложение для сотрудников, обрабатывающих заявки, также должно быть интегрировано с серверной частью. Это позволит им получать доступ к данным заявок и клиентской информации через интернет-соединение.

Рабочие станции сотрудников должны быть оснащены современными компьютерами с операционными системами (Windows 10 и выше) и стабильным интернет-соединением для взаимодействия с облачной системой.

База данных:

Для хранения данных о клиентах, заявках и посылках необходимо развернуть базу данных в облаке. Облачное решение, позволит гибко управлять данными и обеспечит высокую доступность, резервное копирование и защиту данных.

Это решение избавляет от необходимости устанавливать физическое серверное оборудование и выполнять его обслуживание, что существенно упрощает процесс развертывания.

Выводы по разделу 2

Во втором разделе представлены результаты логического проектирования предметной области, основанные на предварительном анализе. В ходе логического проектирования были определены ключевые сущности предметной области, построена диаграмма вариантов использования и разработана диаграмма классов.

На этапе логического моделирования базы данных выполнены следующие шаги:

- определение информационных объектов предметной области;
- указание атрибутов для каждого объекта;
- установление взаимосвязей между объектами.

Работа, проведенная во втором разделе, создаёт основу для перехода к физическому проектированию АИС.

3 Физическое проектирование АИС

3.1 Выбор архитектуры АИС

На этапе реализации автоматизированной информационной системы (АИС) была выбрана трехзвенная архитектура «клиент-сервер». Эта архитектура предполагает наличие трех ключевых компонентов: клиентская часть, сервер приложений и сервер базы данных, что соответствует требованиям для разработки распределенной системы с поддержкой мобильных и настольных пользователей. Ниже прикладываю сравнение возможных архитектур.

Файл-серверная архитектура.

Преимущества: простота реализации, низкие требования к ресурсам.

Недостатки: отсутствие централизованного контроля над данными, низкая безопасность, слабая поддержка распределенных пользователей.

Данный подход не подходит для проекта, так как требуется централизованное управление данными и поддержка как мобильных, так и настольных приложений.

Двухзвенная архитектура «клиент-сервер».

Преимущества: улучшенная производительность за счет разделения клиентской и серверной частей, централизованное управление данными.

Недостатки: клиентская часть сильно нагружена логикой, низкая масштабируемость.

Двухзвенная архитектура могла бы быть применима для упрощенной АИС, однако она не поддерживает гибкое масштабирование и интеграцию нескольких типов клиентов (мобильные устройства, десктопные приложения).

Трехзвенная архитектура «клиент-сервер».

Преимущества: распределение логики между клиентом, сервером приложений и сервером базы данных позволяет гибко масштабировать систему, улучшает безопасность и поддерживает различные типы клиентов.

Недостатки: более сложная реализация, необходимость разработки серверной части.

Трехзвенная архитектура была выбрана как наиболее подходящая для системы с мобильными и десктопными клиентами, а также требующей централизованное хранение данных.

Обоснование выбора:

Трехзвенная архитектура позволяет разделить логику приложения и базы данных, что упрощает разработку, поддержку и масштабирование. В нашем случае она включает:

Клиентская часть:

Мобильное приложение на Kotlin для пользователей, где они могут создавать и отслеживать заказы.

Десктопное приложение на C# для сотрудников компании, которое позволяет обрабатывать заказы.

Серверная часть:

Сервер на Node.js, который отвечает за обработку запросов от клиентов и взаимодействие с базой данных.

База данных:

MongoDB для хранения данных о пользователях, заказах и других сущностях. Она выбрана благодаря своей гибкости и поддержке работы с неструктурированными данными.

Выбор компонентов:

Сервер базы данных: MongoDB была выбрана как база данных благодаря своей гибкости и возможности работать с динамически изменяемыми данными, что идеально подходит для хранения информации о заказах, пользователях и изменяющихся статусах доставки.

Сервер приложений: Node.js был выбран как сервер приложений из-за своей высокой производительности при обработке большого числа асинхронных запросов, что критически важно для системы, работающей в режиме реального времени с клиентами и сотрудниками.

Клиентские приложения.

Мобильное приложение на Kotlin использует возможности взаимодействия с сервером через API для создания и обработки заказов.

Десктопное приложение на C# предоставляет интерфейс для сотрудников, упрощая взаимодействие с системой через удобный графический интерфейс.

3.2 Выбор технологий разработки программного обеспечения АИС

При выборе технологий разработки АИС были рассмотрены следующие варианты:

Технология разработки клиентских приложений:

Мобильное приложение было решено разрабатывать с использованием Kotlin и Android SDK, что обеспечивает быструю разработку и гибкую интеграцию с различными API и библиотеками.

Десктопное приложение разрабатывалось на C# с использованием WPF, что обеспечивает удобство создания сложных графических интерфейсов и возможность интеграции с Windows.

Технология серверной части: Node.js был выбран для разработки серверной части благодаря его асинхронной архитектуре и широкому набору библиотек, что позволяет эффективно обрабатывать множество запросов и работать с базами данных.

База данных.

MongoDB была выбрана как база данных за счет ее гибкости и возможностей горизонтального масштабирования.

Таблица сравнительного анализа средств разработки.

Таблица 4 – Таблица сравнительного анализа средств разработки

Компонент	Технология	Преимущества	Недостатки
Клиентское приложение (мобильное)	Kotlin	Быстрая разработка, поддержка Android	Требуются знания специфики Android SDK
Клиентское приложение (десктоп)	C# (WPF)	Удобство создания интерфейсов, поддержка Windows	Меньшая кроссплатформенность
Серверная часть	Node.js	Высокая производительность, асинхронная обработка	Сложность работы с потоками данных
База данных	MongoDB	Масштабируемость, гибкость	Необходимость настройки для больших объемов данных

Таким образом, выбор технологий и архитектуры обоснован требованием создания распределенной системы, поддерживающей как мобильных, так и настольных пользователей, а также обеспечивающей централизованное управление данными.

Выбор системы управления базами данных (СУБД) является важным этапом разработки автоматизированной информационной системы (АИС), так как от него зависит эффективность хранения и обработки данных. В процессе выбора СУБД были учтены требования системы, такие как поддержка распределенной архитектуры, гибкость, возможность масштабирования, а также опыт использования СУБД совместно с серверной частью на Node.js.

Для реализации системы был проведен сравнительный анализ следующих СУБД: MongoDB, MySQL, PostgreSQL, Microsoft SQL Server. Основные критерии для выбора СУБД включали:

- гибкость хранения данных;
- поддержка масштабирования;

- производительность при высоких нагрузках;
- совместимость с Node.js;
- лицензирование и стоимость;
- анализ СУБД.

Таблица 5 – Сравнительный анализ СУБД

СУБД	Тип	Лицензия	Поддержка масштабирования	Производительность	Гибкость работы с данными	Совместимость с Node.js
MongoDB	NoSQL	Бесплатная (Community), платная (Enterprise)	Высокая (шардинг и репликация)	Высокая для больших объемов данных и асинхронных запросов	Очень гибкая, поддержка JSON-документов	Отличная (официальные драйверы и библиотеки)
MySQL	SQL	Бесплатная (GPL), платная (Enterprise)	Ограниченная (репликация)	Хорошая, но падает с увеличением объема данных	Менее гибкая, строгая схема данных	Хорошая (драйверы Node.js)
PostgreSQL	SQL	Бесплатная (PostgreSQL License)	Ограниченная (репликация)	Высокая для сложных запросов	Гибкая, поддержка JSON	Хорошая (драйверы Node.js)
Microsoft SQL Server	SQL	Платная, бесплатная (Express)	Ограниченная в бесплатных версиях	Хорошая, но требует много ресурсов	Менее гибкая, строгая схема данных	Хорошая, но сложнее интегрировать с Node.js

На основании проведенного анализа была выбрана MongoDB как основная СУБД для разработки АИС по следующим причинам:

Гибкость данных: MongoDB позволяет хранить данные в формате JSON, что делает её особенно удобной для хранения динамических данных, таких как заказы и информация о пользователях, где поля могут меняться или обновляться по мере необходимости.

Масштабируемость: Благодаря поддержке горизонтального масштабирования через шардинг, MongoDB может эффективно справляться с большими объемами данных, что важно для роста системы.

Производительность: MongoDB обеспечивает высокую производительность при обработке больших объемов данных и асинхронных запросов, что критически важно для серверной части на Node.js, которая работает в режиме реального времени.

Совместимость с Node.js: MongoDB имеет официальные драйверы и библиотеки для Node.js, что упрощает интеграцию и разработку, а также обеспечивает высокую скорость обработки запросов.

Лицензия: Бесплатная версия MongoDB (Community Edition) обладает всеми необходимыми возможностями для разработки и тестирования системы, а при необходимости можно легко перейти на платную версию для поддержки на уровне предприятия.

Таким образом, MongoDB выбрана как оптимальная СУБД для реализации АИС благодаря своей гибкости, высокой производительности и возможности масштабирования, а также простоте интеграции с серверной частью, реализованной на Node.js.

3.3 Разработка физической модели данных АИС

Физическая модель данных разрабатывается на основе концептуальной модели и отражает структуру базы данных, используемой для хранения и обработки данных в автоматизированной информационной системе (АИС).

Для реализации АИС в данном проекте была выбрана нереляционная СУБД MongoDB, которая использует документо-ориентированный подход к хранению данных. Физическая модель в контексте MongoDB представляет собой коллекции, в которых хранятся документы (аналог записей в таблицах реляционных баз данных) [2].

MongoDB [15] использует собственный диалект запросов, отличающийся от классического SQL. Вместо таблиц и записей MongoDB оперирует коллекциями и документами, которые хранятся в формате BSON (Binary JSON). Для взаимодействия с данными используется библиотека MongoDB Query Language (MQL), которая предоставляет гибкие возможности для фильтрации, сортировки, агрегации и модификации данных.

Примеры основных операций.

Создание документа (аналог вставки записи в SQL):

– db.orders.insertOne({ "userId": 1, "price_RUR": 15000, "adressAt_X": 55.751244, "adressAt_Y": 37.618423, "adressTo_X": 55.764856, "adressTo_Y": 37.578841, "Status": "OPEN", "created": new Date() });
– Чтение документа (аналог запроса SELECT в SQL): db.orders.find({ "userId": 1 });

Структура коллекций и документов

Для обеспечения работы системы были разработаны коллекции для хранения информации о пользователях, заказах и других сущностях системы. Ниже приведены основные коллекции и их структура. Примеры документа коллекции можно увидеть в приложении Б

Коллекция пользователей (Users):

Типы данных:

userId — числовой тип (integer), уникальный идентификатор пользователя.

phone — строка (string), номер телефона пользователя.

password — строка (string), пароль пользователя.

FIO — строка (string), полное имя пользователя.

createdAt — дата и время создания (date).

role — строка (string), роль пользователя ("Client" или "Employee").

Коллекция заказов (orders):

Коллекция заказов (Orders)

Типы данных:

orderId — числовой тип (integer), уникальный идентификатор заказа.

userId — числовой тип (integer), идентификатор пользователя, сделавшего заказ.

price_RUR — числовой тип (integer), стоимость доставки в рублях.

adressAt_X, adressAt_Y — координаты отправления (double).

adressTo_X, adressTo_Y — координаты доставки (double).

Status — строка (string), статус заказа ("OPEN", "CLOSED").

created — дата и время создания (date).

Оптимизация запросов

В MongoDB важную роль в оптимизации запросов играет использование индексов. Индексы позволяют ускорить операции поиска по ключевым полям документов. В проекте были добавлены индексы на следующие поля:

Индекс по userId в коллекции orders для быстрого поиска заказов пользователя:

```
db.orders.createIndex({ "userId": 1 });
```

Индекс по orderId для быстрого доступа к заказам по их идентификатору:

```
db.orders.createIndex({ "orderId": 1 });
```

Также MongoDB поддерживает агрегатные запросы с использованием aggregation pipeline, что позволяет выполнять сложные операции над данными, такие как группировка, сортировка и вычисление статистик.

Пример агрегатного запроса для подсчета количества заказов по статусу:

```
db.orders.aggregate([
  { $group: { _id: "$Status", count: { $sum: 1 } } }
]);
```

3.4 Разработка программного обеспечения АИС

3.4.1 Иерархия функций управления и обработки данных

Разрабатываемая автоматизированная информационная система (АИС) для управления курьерской службой предназначена для автоматизации следующих функций:

- управление пользователями и авторизация;
- обработка заказов (создание, просмотр истории, изменение статуса);
- управление информацией о пользователях (профиль клиента);
- отображение данных на карте (выбор координат точек доставки и отправки);
- генерация данных для мобильного приложения клиентов и десктопного приложения сотрудников.

Функции можно разделить на две основные категории:

Основные функции:

- регистрация пользователей: ввод персональных данных (ФИО, телефон, пароль) для регистрации в системе;
- создание заказов: выбор начальной и конечной точек на карте, расчёт стоимости доставки, подтверждение заказа;
- просмотр и редактирование профиля: отображение и изменение данных о пользователе (ФИО, телефон, пароль);
- просмотр истории заказов: клиент может просматривать свои заказы, а сотрудник — заказы всех клиентов, с возможностью фильтрации по статусам;
- обновление статуса заказа: сотрудники могут обновлять статус заказа (например, «OPEN», «CLOSED»).

Служебные функции:

- авторизация пользователей: вход в систему на основе введённых данных (телефон и пароль) с проверкой роли пользователя (клиент или сотрудник);
- проверка аутентификации: валидация логина и пароля пользователя при каждом запросе к серверу;
- обработка географических данных: передача координат точек (адрес отправления и получения) между клиентом и сервером с использованием API Yandex Maps;
- обмен данными с сервером: передача данных о заказах, пользователях и профилях через REST API на сервере Node.js.

3.4.2 Описание и проектирование программных модулей

Разработка программного обеспечения АИС разбита на несколько модулей, каждый из которых отвечает за определённый набор функций [6].

Модули для мобильного приложения клиента (Kotlin):

AuthModule: отвечает за авторизацию и регистрацию пользователей.

ProfileModule: обеспечивает работу с профилем пользователя (просмотр и редактирование).

OrderModule: модуль для работы с заказами (создание, просмотр).

MapModule: интеграция с Yandex Maps для выбора начальной и конечной точек доставки.

NetworkModule: взаимодействие с сервером (отправка и получение данных через API).

Модули для серверной части (Node.js):

UserController: обработка запросов, связанных с пользователями (регистрация, авторизация, получение профиля).

OrderController: управление заказами (создание, обновление статусов, получение истории заказов).

Модули для десктопного приложения (C#):

AuthModule: отвечает за авторизацию и регистрацию пользователей

OrderManagementModule: просмотр, обновление статусов и управление заказами.

ReportModule: генерация отчётов для сотрудников о статусах и выполнении заказов.

ClientsManagementModule: просмотр, данных пользователей, а также списка их заказов

Диалог в мобильном приложении для клиентов и десктопном приложении для сотрудников организованы через интерфейсы ввода и выбора, которые отображают текущую информацию и предоставляют пользователю возможности для ввода данных.

Ввод первичной информации: пользователи вводят данные для регистрации.

Обработка данных: система обрабатывает введённые данные и отправляет их на сервер.

Просмотр информации: пользователи могут просматривать свои заказы, а сотрудники — заказы всех пользователей.

Ввод данных: пользователи мобильного приложения могут создавать свои заказы, вводя ключевую информацию.

Пример диалога для создания заказа:

Выбор начальной точки: пользователь перемещает карту и выбирает место отправления.

Выбор конечной точки: пользователь перемещает карту и выбирает место доставки.

Расчёт стоимости доставки: система автоматически рассчитывает стоимость на основе расстояния между точками.

Подтверждение заказа: после подтверждения заказа данные отправляются на сервер.

Ниже, на рисунке 9, представлена структура программных модулей.

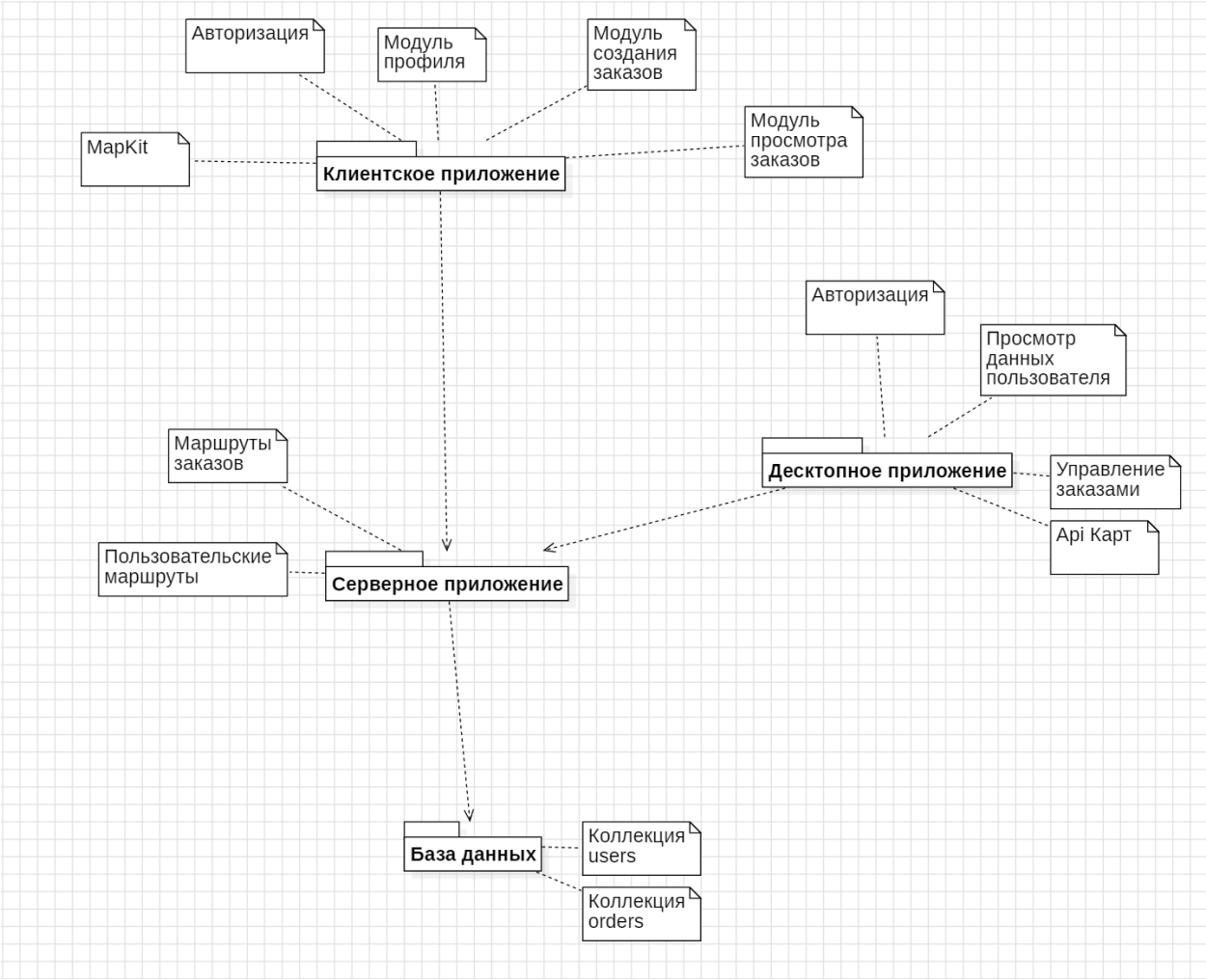

Рисунок 9 – Структура программных модулей

Представленная декомпозиция разработки программного обеспечения АИС на модули позволит структурировать процесс создания системы и упростить ее дальнейшее сопровождение.

3.4.3 Блок-схемы алгоритмов работы

В данном разделе будут показаны две блок схемы: авторизация и создание заказа [1], [4]. Остальные блок-схемы можно увидеть в приложении Ж.

Блок-схема алгоритмов авторизации представлена на рисунке 10.

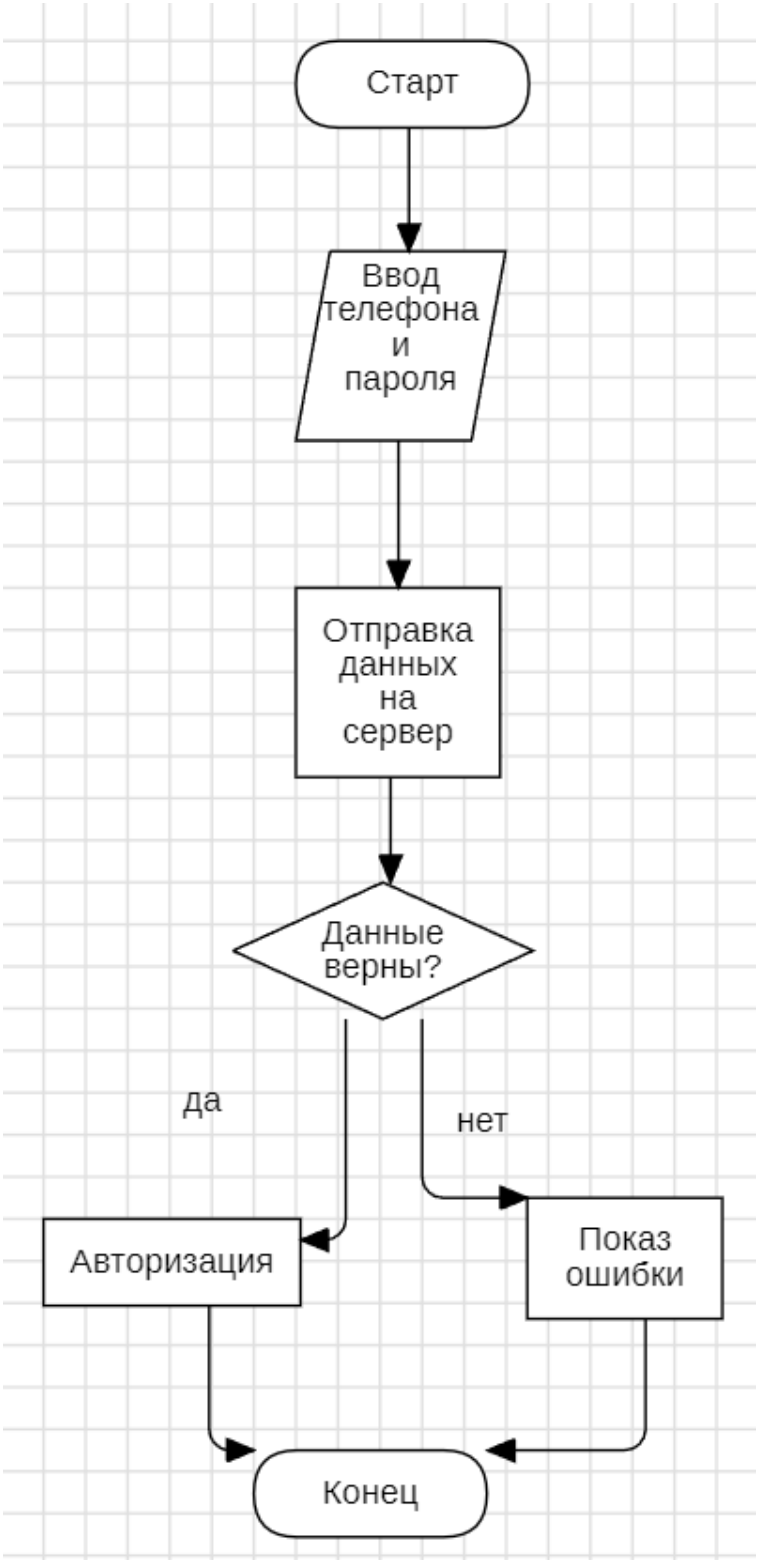

Рисунок 10 – Блок схема алгоритма авторизации

– пользователь вводит телефон и пароль;
– отправка данных на сервер;

– если данные верны – создание сессии и вход;

– если неверны – вывод ошибки;

Рисунок 11 – Блок схема создания заказа

– пользователь выбирает начальную и конечную точку на карте;

– расчёт стоимости доставки;
– подтверждение заказа;
– отправка данных на сервер;
– сохранение заказа в базе данных.

3.5 Реализация АИС

Разработка информационной системы (АИС) для автоматизации работы курьерской службы выполнена с нуля. Основные задачи заключались в создании и разработке всех компонентов системы: базы данных, серверной и клиентской частей, а также десктопного приложения для сотрудников. Проект был выполнен в выбранной среде программирования, а ключевые модули были написаны с использованием современных технологий, таких как Node.js для сервера, MongoDB для хранения данных, а также Kotlin для мобильного приложения и C# для десктопного приложения.

Разработка интерфейса клиентской и десктопной частей была выполнена с использованием фреймворков для работы с графическими компонентами (Android SDK для мобильного приложения и WPF для десктопного). В интерфейсе учитываются удобство использования и адаптивность.

Разработанная система использует трехуровневую архитектуру "клиент-сервер", которая включает:

Клиентское мобильное приложение — пользователи (клиенты) могут создавать заказы и отслеживать их статус.

Десктопное приложение для сотрудников — сотрудники курьерской службы обрабатывают поступающие заказы.

Серверная часть — сервер взаимодействует с клиентами и сотрудниками через REST API, обрабатывает запросы, управляет сессиями и ведет взаимодействие с базой данных MongoDB.

Мобильное приложение на Kotlin [13].

Мобильное приложение предоставляет пользователю интерфейс для создания доставки, отслеживания заказов и управления профилем. Основные функции включают:

Авторизация и регистрация. Пользователь вводит номер телефона и пароль, которые проверяются на сервере.

Создание заказов. Пользователь выбирает точку отправки и доставки на карте и подтверждает создание заказа [18].

Отслеживание истории заказов. Пользователь видит все свои заказы и их статус.

Изменения данных профиля. Пользователь видит ключевую информацию о себе и может ее изменить.

Основным функционалом клиентского приложения – создание заказа. Он реализован в виде так называемого «визарда»: несколько фрагментов с порционно выдаваемой информацией. В моем случае: фрагмент с кнопкой создания заказа, фрагмент с выбором точки забора посылки, фрагмент с выбором точки доставки, расчет стоимости и кнопка создания заказа. В приложении В будет предоставлен листинг кода, с помощью которого отрисовываются страницы создания заказа.

Также важным элементом приложения является авторизация. Благодаря ней можно идентифицировать пользователя и отследить его заказы.

Листинг 1 – Метод авторизации пользователя в мобильном приложении

```
private fun loginUser(phone: String, password: String) {
    val loginRequest = LoginRequest(phone, password)
    val call = apiService.loginUser(loginRequest)

    call.enqueue(object : Callback<LoginResponse> {
        override fun onResponse(call: Call<LoginResponse>, response:
Response<LoginResponse>) {
```

```
            if (response.isSuccessful) {
                val loginResponse = response.body()
                if (loginResponse?.message == "Login successful") {
                    val sharedPref = getSharedPreferences("UserPrefs",
MODE_PRIVATE)
                    val editor = sharedPref.edit()
                    editor.putInt("userId", loginResponse.userId) // Сохраняем userId
                    editor.apply()

                    // Переход на главную страницу
                    Toast.makeText(this@LoginActivity, "Успешная авторизация",
Toast.LENGTH_SHORT).show()
                    val intent = Intent(this@LoginActivity, MainActivity::class.java)
                    startActivity(intent)
                    finish()
                } else {
                    Toast.makeText(this@LoginActivity, "Неверный телефон или
пароль", Toast.LENGTH_SHORT).show()
                }
            } else {
                Toast.makeText(this@LoginActivity, "Ошибка сервера",
Toast.LENGTH_SHORT).show()
            }
        }

        override fun onFailure(call: Call<LoginResponse>, t: Throwable) {
            Toast.makeText(this@LoginActivity, "Ошибка подключения:
${t.message}", Toast.LENGTH_SHORT).show()
        }
    })
```

Важнейшим элементом работы приложения является работа с сервером, который обрабатывает запросы, обращается к базе данных и возвращает ответы приложению. Для реализации данного решения использовалась библиотека Retrofit, который сильно упрощает работу с сетевыми запросами. В листинге 2 продемонстрирована инициализация Апи клиента.

Листинг 2 – Инициализация Retrofit

```
object ApiClient {
   private const val BASE_URL = "http://10.0.2.2:3000/"

   private val retrofit by lazy {
      Retrofit.Builder()
         .baseUrl(BASE_URL)
         .addConverterFactory(GsonConverterFactory.create())
         .build()
   }

   fun <Api> createApi(api: Class<Api>): Api {
      return retrofit.create(api)
   }
}
```

Полный программный код АПИ сервиса, который обращается к сервису можно посмотреть в приложении Г.

Серверная часть на Node.js

Серверная часть разработана на платформе Node.js с использованием Express.js. Она предоставляет REST API для взаимодействия с клиентскими

приложениями, обрабатывает заказы и управляет профилями пользователей. Основные функции:

Управление пользователями: регистрация, авторизация и управление профилями.

Создание и управление заказами: при получении нового заказа сервер сохраняет его в базе данных и назначает уникальный ID.

Пример кода создания заказа предоставлен в листинге 3 [17].

Листинг 3 – Маршрут создания заказа

```
router.post('/create', async (req, res) => {
   const { userId, price_RUR, adressAt_X, adressAt_Y, adressTo_X, adressTo_Y
} = req.body;
   console.log('Received create order request:', req.body);
   const lastOrder = await Order.findOne().sort({ id: -1 });
   const nextOrderId = lastOrder ? lastOrder.id + 1 : 1
   try {
      const newOrder = new Order({
         id: nextOrderId,
         userId,
         price_RUR,
         adressAt_X,
         adressAt_Y,
         adressTo_X,
         adressTo_Y,
         Status: 'OPEN',
         created: new Date()
      });
      await newOrder.save();
      res.status(201).json({
```

```
            message: 'Заказ успешно создан',
            orderId: newOrder.id
        });
    } catch (err) {
        console.error('Error creating order:', err);
        res.status(500).json({ message: 'Ошибка при создании заказа', error:
err.message });
    }
});
```

Десктопное приложение предоставляет интерфейс для сотрудников курьерской службы, где они могут:

Просматривать новые заказы, изменять их статусы и управлять выполнением.

Получать подробную информацию о пользователях и заказах.

В приложении Д предоставлен код информации о заказе [14].

Также как и в мобильном приложении, важной частью взаимодействия с сервером считается апи клиент [16]. Пример запроса заказов пользователя представлен в листинге 4. С полным кодом API клиента можно ознакомиться в приложении Г.

Листинг 4 – Запрос заказов пользователя

```
public async Task<string> GetUserOrdersAsync(string userId)
{
    var response = await _httpClient.GetAsync($"users/{userId}/orders");
    response.EnsureSuccessStatusCode();
    return await response.Content.ReadAsStringAsync();
}
```

Аналогично мобильному приложению, для визуализации информации используется библиотека Яндекс Карт [11].

3.6 Описание функциональности АИС

Разработанная автоматизированная информационная система (АИС) для курьерской службы предназначена для автоматизации процесса обработки заказов, взаимодействия с клиентами, а также управления и отслеживания информации. Ниже приводится описание организации и технологии сбора, передачи, обработки и выдачи информации.

Сбор первичной информации

Сбор данных начинается с клиента, который через мобильное приложение вводит информацию о своем заказе. Это включает:

Выбор адреса отправления и доставки на карте.

Указание личной информации при регистрации (ФИО, номер телефона).

Создание и подтверждение заказа.

Передача информации

После того как клиент создает заказ, данные передаются на сервер через REST API. Запрос содержит следующую информацию:

Координаты точки отправления и доставки.

Стоимость доставки, рассчитанную на основании расстояния между точками.

Идентификатор пользователя, создавшего заказ.

Обработка информации на сервере

Серверная часть на Node.js принимает запросы от мобильного приложения. Основные операции обработки данных включают:

Авторизация пользователя на основании введенных данных (номер телефона и пароль).

Сохранение заказа в базе данных (MongoDB), присвоение уникального идентификатора и установка начального статуса заказа («OPEN»).

Обработка запросов от сотрудников через десктопное приложение для изменения статуса заказа (например, «IN_PROGRESS», «CLOSED»).

Передача результатной информации

После обработки сервер отправляет результатные данные:

Клиенту передается сообщение о создании заказа с номером заказа и его статусом.

Сотрудники курьерской службы через десктопное приложение видят список новых заказов и могут изменять их статус.

Вся информация обновляется в режиме реального времени, обеспечивая оперативную связь между клиентом и сотрудниками.

Выдача информации

Клиент может в любой момент проверить статус своего заказа через мобильное приложение. Он видит:

Текущий статус (например, «OPEN», «IN_PROGRESS», «CLOSED »).

Стоимость доставки и расчетное время доставки.

Сотрудники курьерской службы, используя десктопное приложение, могут:

- просматривать поступившие заказы;
- изменять статус заказов;
- отслеживать выполнение доставки.

Примеры скриншотов приложения

Экран авторизации пользователя представлен на рисунках 12, 13.

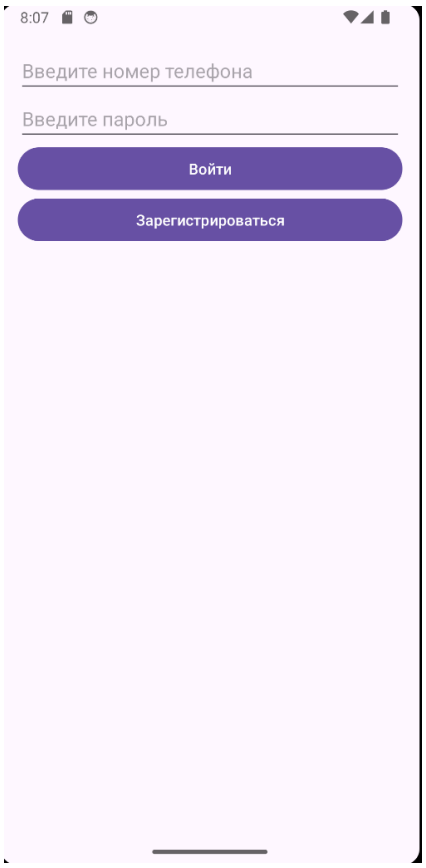

Рисунок 12 – Экран авторизации пользователя в мобильном приложении

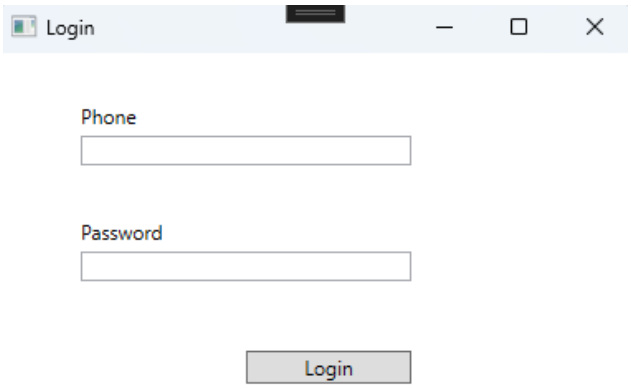

Рисунок 13 – Экран авторизации десктопного приложения

Экран создания заказа представлен на рисунке 14. Экран выбора точки доставки представлен на рисунке 15.

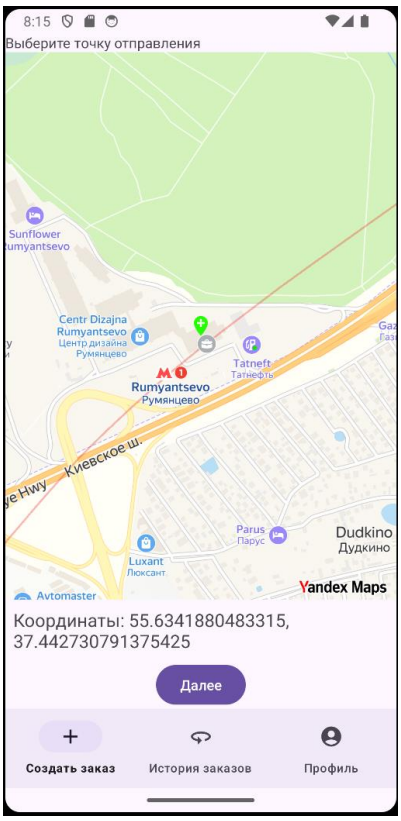

Рисунок 14 – Экран создания заказа. Выбор точки отправления

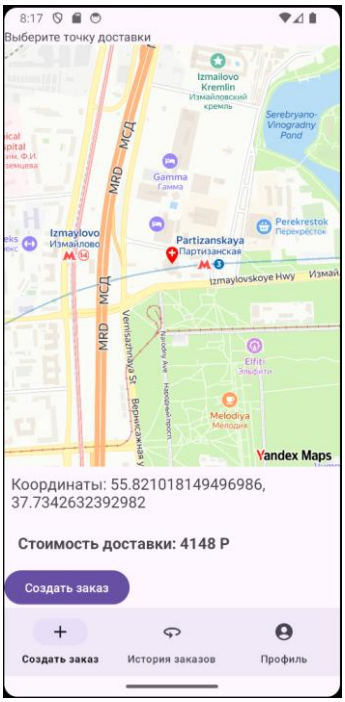

Рисунок 15 – Экран выбора точки доставки

Экран истории заказов представлен на рисунке 16.

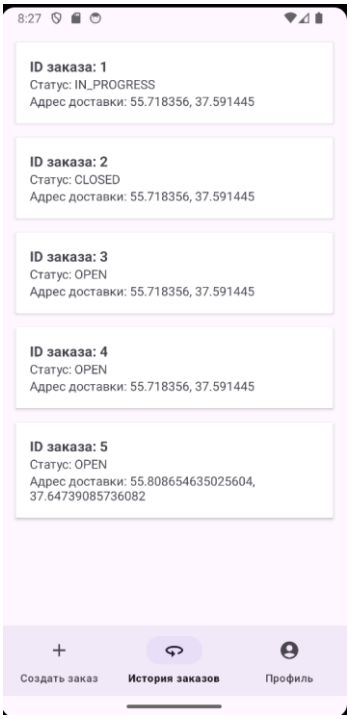

Рисунок 16 – Экран истории заказов в мобильном приложении

Возможность просмотреть детали каждого заказа показана на рисунке 17.

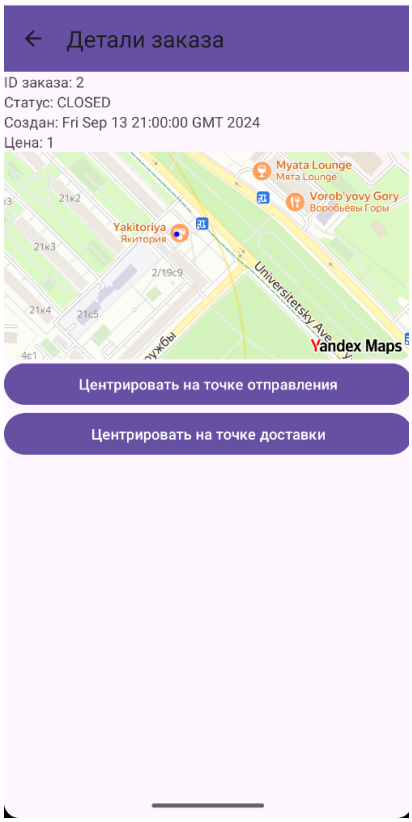

Рисунок 17 – Экран деталей заказа в мобильном приложении

Интерфейс для сотрудников представлен на рисунке 18.

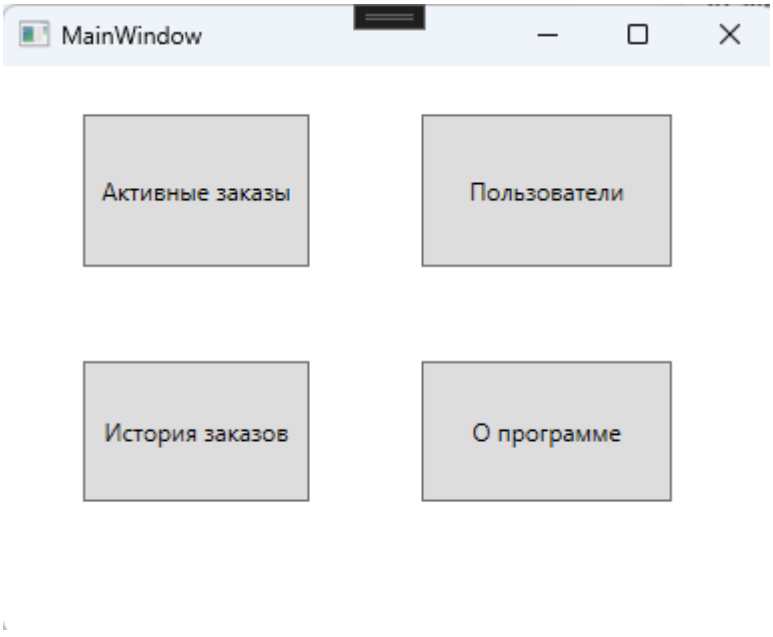

Рисунок 18 – Главное меню приложения сотрудников

Список открытых заказов, которые нужно подтвердить представлен на рисунке 19.

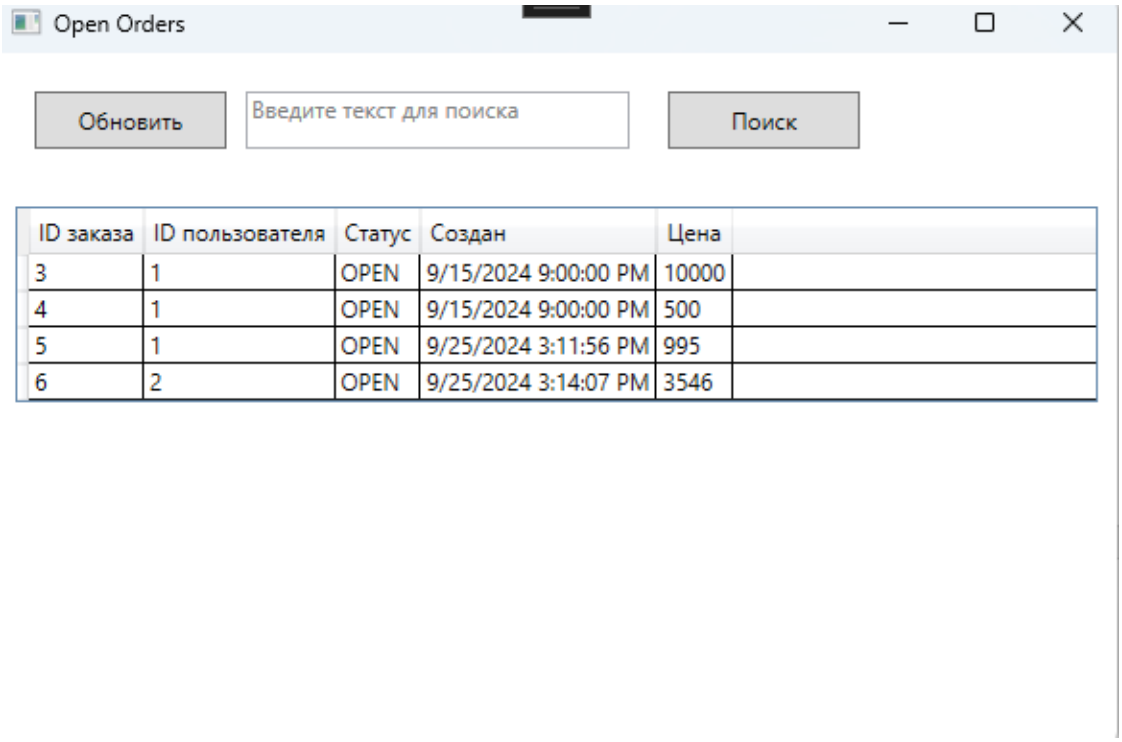

Рисунок 19 – Таблица открытых заказов

Детальная информация о заказе представлен на рисунке 20.

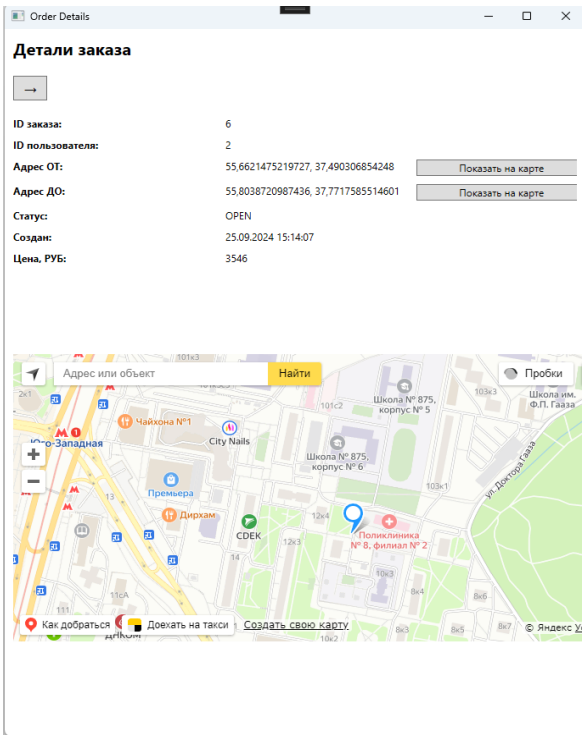

Рисунок 20 – Экран деталей заказа

Список всех заказов с возможностью экспортировать их в формате csv представлен на рисунке 21.

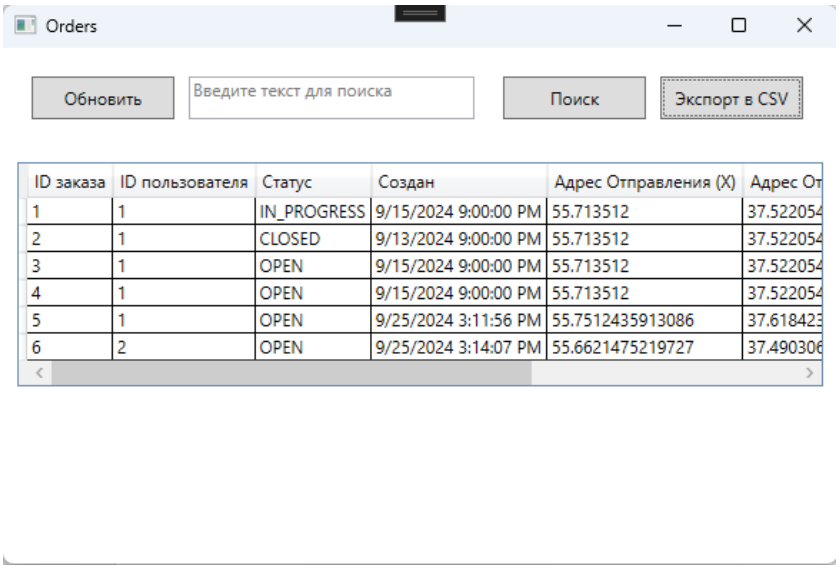

Рисунок 21 – Экран всех заказов

Экран информации о пользователе представлен на рисунке 22.

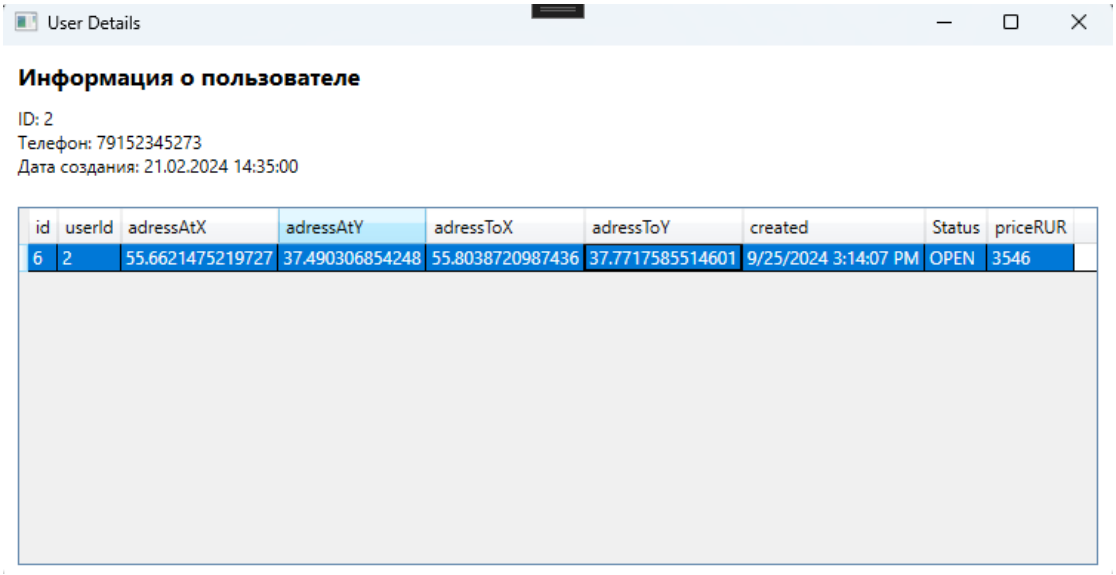

Рисунок 22 – Экран информации о пользователе с основной информацией и списком заказов

Представленные экранные формы демонстрируют функциональные возможности АИС.

3.7 Тестирование программного проекта

Тестирование является важной стадией разработки автоматизированной информационной системы (АИС), обеспечивая проверку корректности функционирования всех её компонентов, выявление возможных ошибок и оценку качества программного обеспечения. В рамках разработки АИС для курьерской службы использовались функциональные и интеграционные методы тестирования.

Для тестирования АИС был выбран метод функционального тестирования, который проверяет, корректно ли программное обеспечение выполняет свои функции в соответствии с требованиями. Также применялось интеграционное тестирование для проверки взаимодействия между

различными модулями системы, такими как клиентское приложение, серверная часть и база данных. Часть программного кода, которая не использует интеграцию с внешними элементами (API Яндекс Карт) было покрыто юнит тестами (C#) и автоматизированными UI тестами на базе Espresso (Kotlin) [12].

Критерии, которые использовались для оценки качества программного обеспечения, включают:

Функциональная корректность — проверка, корректно ли работают основные функции АИС.

Производительность — оценка времени отклика на пользовательские запросы и обработки данных.

Интеграция компонентов — проверка корректной передачи данных между клиентом, сервером и базой данных.

Тестирование было разделено на несколько этапов, охватывающих ключевые модули системы:

Мобильное приложение — тестировались экраны авторизации, создания заказа, просмотра истории заказов и деталей заказа.

Десктопное приложение — тестировались экраны авторизации, загрузки списка заказов, поиска, открытия детализации заказов.

Серверная часть — проверялась обработка запросов на создание заказа, изменение статуса заказа, авторизация и регистрация пользователей.

База данных (MongoDB) — тестировались операции записи и чтения данных (создание заказов, обновление статусов и т.д.).

Для тестирования мобильного и десктопных приложений были написаны автоматические тесты на проверку экранов, в которых нет сторонних библиотек (не были покрыты экраны с картами)

Помимо ручного тестирования также были написаны автоматические тесты. Для десктопного приложения использовался Appium. Автотесты для мобильного приложения написаны с помощью фреймфорка Espresso. Так как выполнение автотестов – трдозатратная операция для систем, было принято

решение покрыть автотестами только навигацию в приложении, а остальные сценарии проверить вручную. С полным кодом автотестов можно ознакомиться в приложении Е.

Результаты тестирования программного проекта представлены в виде таблицы, отражающей основные проверенные компоненты, тестовые сценарии, ожидаемые и фактические результаты, а также статус теста.

Таблица 6 – Результаты тестирования АИС

Компонент	Тестовый сценарий	Ожидаемый результат	Фактический результат	Статус
Мобильное приложение	Авторизация пользователя	Пользователь успешно авторизуется	Авторизация прошла успешно	Успешно
Мобильное приложение	Попытка входа с неверным паролем	Ошибка авторизации	Ошибка авторизации выведена	Успешно
Мобильное приложение	Создание заказа	Заказ создается с верными данными	Заказ создан, ID заказа передан	Успешно
Мобильное приложение	Неполные данные при создании заказа	Ошибка валидации данных	Сообщение о неверных данных	Успешно
Серверная часть	Создание заказа через API	Заказ создается и сохраняется в базе данных	Заказ создан и сохранен в базе данных	Успешно
Серверная часть	Проверка обработки неверного запроса	Сообщение об ошибке	Сообщение об ошибке отправлено	Успешно
База данных	Чтение и запись данных о пользователе	Данные сохраняются и извлекаются корректно	Данные сохранены и извлечены	Успешно

Продолжение таблицы 6

Компонент	Тестовый сценарий	Ожидаемый результат	Фактический результат	Статус
База данных	Обновление статуса заказа	Статус обновлен в базе данных	Статус обновлен корректно	Успешно
Мобильное приложение	Просмотр истории заказов	Список заказов отображается корректно	История заказов отображается	Успешно
Десктопное приложение	Авторизация пользователя	Пользователь успешно авторизуется	Пользователь успешно авторизуется	Успешно
Десктопное приложение	Авторизация пользователя не имеющего роль сотрудника	Ошибка авторизации	Ошибка авторизации	Успешно
Десктопное приложение	Просмотр истории заказов	Загружается список заказов	Загружается список заказов	Успешно
Десктопное приложение	Поиск заказа	Список заказов соответствует поисковому запросу	Список заказов соответствует поисковому запросу	Успешно
Десктопное приложение	Загрузка всех активных заказов	Список заказов сформировался	Список заказов сформировался	Успешно
Десктопное приложение	Загрузка информации о пользователе	Информация о пользователе загружена	Информация о пользователе загружена	Успешно
Десктопное приложение	Экспорт данных о заказах	Данные выгружаются	Данные выгружаются	Успешно

Отчет по выполненным автотестам показан на рисунке 23.

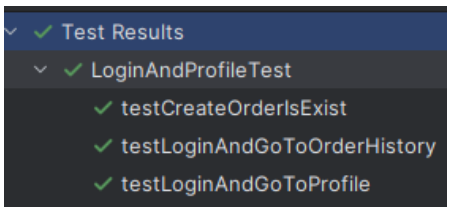

Рисунок 23 – Отчет по пройденным тестам в мобильном приложении

На рисунке 24 представлен отчет по пройденным тестам для десктопного приложения.

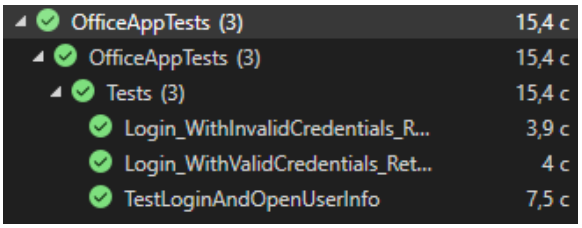

Рисунок 24 – Отчет по пройденным тестам в десктопном приложении

Все ключевые компоненты системы успешно прошли тестирование, показав корректную работу при выполнении основных операций: авторизация, создание и изменение заказов, взаимодействие с базой данных. Были протестированы и исправлены проблемы с валидацией данных при создании заказов, а также выявлены и устранены незначительные ошибки интерфейса.

Вывод по третьему разделу

В третьем разделе был выполнен выбор архитектуры АИС, выбор технологии разработки программного обеспечения, выбор СУБД, разработка физической модели данных, разработка ПО. Также было выполнено описание работы ПО и тестирование.

Заключение

В рамках выполненной работы был разработан программный комплекс для автоматизации работы курьерской службы, включающий серверную часть, клиентское приложение для пользователей и офисное приложение для сотрудников.

Актуальность данной работы обусловлена необходимостью повышения эффективности процесса обработки заказов в курьерской службе, улучшения точности и скорости доставки, а также минимизации человеческого фактора при учёте и обработке данных.

Целью работы являлась разработка автоматизированной информационной системы (АИС), которая позволила бы оптимизировать деятельность курьерской службы.

Для достижения поставленной цели были сформулированы и решены следующие задачи:

- выполнено функциональное моделирование процессов предприятия, которому необходима автоматизация бизнес-процессов, чтобы понять специфику области, для которой разрабатывается АИС;
- выполнено логическое проектирование АИС для создания четкой и формализованной модели информационной системы, которая определяет, как будут обрабатываться данные, взаимодействовать элементы системы и функционировать бизнес-процессы на уровне логики работы системы;
- выбор архитектуры АИС — выбрана трёхзвенная архитектура клиент-сервер, что позволило эффективно организовать взаимодействие между клиентским приложением, сервером и базой данных;
- выбор СУБД и проектирование базы данных — проанализированы и обоснованы различные СУБД, выбрана MongoDB, которая

обеспечила гибкость работы с данными и высокую производительность. Разработана физическая модель данных, включающая ключевые таблицы и связи для управления заказами, пользователями и маршрутами;

- разработка серверной части на Node.js — реализован API для обработки запросов от клиентских и офисных приложений, обеспечивающий создание заказов, аутентификацию пользователей и управление данными. Для хранения данных была интегрирована MongoDB;
- разработка клиентского и офисного приложений — созданы клиентское приложение на Kotlin для Android и офисное приложение на C#. Клиентское приложение предоставляет пользователям возможность создавать заказы и отслеживать их статус, офисное — обеспечивает работу сотрудников с заказами;
- тестирование — проведено тестирование всех компонентов системы, в результате которого подтверждена корректность работы основных функций, таких как аутентификация, создание и обновление заказов, взаимодействие с базой данных и отображение информации;

Научная значимость работы заключается в исследовании современных технологий разработки клиент-серверных систем и их адаптации для автоматизации бизнес-процессов курьерских служб.

Практическая ценность системы заключается в значительном упрощении управления заказами, повышении эффективности работы сотрудников и улучшении обслуживания клиентов.

В процессе разработки использовались проверенные подходы и методы проектирования АИС.

Основным источником данных для работы послужили реальные сценарии работы курьерских служб, а для проверки функционала была

проведена имитация реальных заказов. Результаты тестирования подтвердили надёжность и производительность системы.

В сравнении с аналогичными решениями, разработанная АИС демонстрирует высокую производительность и удобство использования, что обусловлено применением современных технологий, таких как MongoDB, Node.js и Kotlin.

Дальнейшее развитие системы возможно в направлении добавления новых функциональных возможностей, таких как интеграция с платёжными системами, улучшение аналитики заказов и расширение географической зоны обслуживания за счёт оптимизации маршрутов [5].

Полученные результаты открывают возможности для широкого внедрения системы в реальную эксплуатацию, что приведёт к увеличению эффективности работы курьерских служб.

Список используемой литературы

1. Гвозденко Н. П., Суслова С. А. Разработка блок-схем и алгоритмов: Липецкий государственный технический университет, 2021. – 59 с.
2. Дейт Дж. Введение в системы баз данных 8-е издание: Диалектика, 2019. –1328 с.
3. Как использовать хранилища данных в бизнес-аналитике / Atlassian/ Atlassian.com [Электронный ресурс]: сайт: https://www.atlassian.com/data/business-intelligence/data-warehouses-guide (дата обращения: 20.09.2024).
4. Клеппман М. UML. Основы, 3-е издание: Издательство Символ Плюс, 2014. –192 с.
5. Кормен Т. Алгоритмы. Вводный курс: Изд-во Вильямс, 2016. – 208 с.
6. Мкртычев, С. В. Прикладная информатика. Бакалаврская работа : электрон. учеб.-метод. пособие : Тольяттинский государственный университет. – Тольятти : Изд-во ТГУ, 2019. –74 с [Электронный ресурс]: сайт: https://dspace.tltsu.ru/handle/123456789/8868 (дата обращения: 01.09.2024).
7. Система автоматизации курьерской службы доставки MEASOFT. [Электронный ресурс]: сайт: https://courierexe.ru/calc.htm (дата обращения: 10.09.2024).
8. Система автоматизации курьерской службы доставки Mobidel. [Электронный ресурс]: сайт: https://mobidel.ru/about (дата обращения: 10.09.2024).
9. Фаулер М. Высоконагруженные приложения. Программирование, масштабирование, поддержка: Sprint Book, 2024. –640 с.

10. Форсгрен Н., Хамбл Дж., Ким Дж. Ускоряйся! Наука DevOps : Как создавать и масштабировать высокопроизводительные цифровые организации: Альпина PRO, 2022. –220 с.

11. Api Геокодера [Электронный ресурс] сайт: https://yandex.ru/dev/geocode/doc/ru/ (дата обращения: 15.09.2024).

12. Espresso [Электронный ресурс] сайт: https://developer.android.com/training/testing/espresso?hl=ru (дата обращения: 01.10.2024).

13. Hoberman S. - " Data Modeling Made Simple with ER/Studio Data Architect." Technocs Publications, 2015.

14. JetBrains, Kotlin Documentation [Электронный ресурс] сайт: https://kotlinlang.org/docs/home.html (дата обращения: 25.09.2024).

15. Microsoft, C# language documentation [Электронный ресурс], сайт: https://docs.microsoft.com/en-us/dotnet/csharp/ (дата обращения: 14.09.2024).

16. MongoDB, MongoDB documentation [Электронный ресурс], сайт: https://www.mongodb.com/docs/ (дата обращения: 05.09.2024).

17. NewtonSoft JSON Framework [Электронный ресурс] сайт: https://www.newtonsoft.com/json (дата обращения: 15.09.2024).

18. Node.JS, Node.js documentation [Электронный ресурс], сайт: https://nodejs.org/api/all.html (дата обращения: 11.09.2024).

19. Subharum Pal, Optimizing the Last Mile: Advanced Predictive Analytics for Delivery Time Estimation in Supply Chain Logistics - International Journal for Innovative research in multidisciplinary field, 2023.

20. Yandex MapKit SDK [Электронный ресурс] сайт: https://yandex.ru/dev/mapkit/doc/ru/ (дата обращения: 25.09.2024).

Приложение А

Декомпозиция второго уровня выполнения заказа курьерской службы «КАК ЕСТЬ»

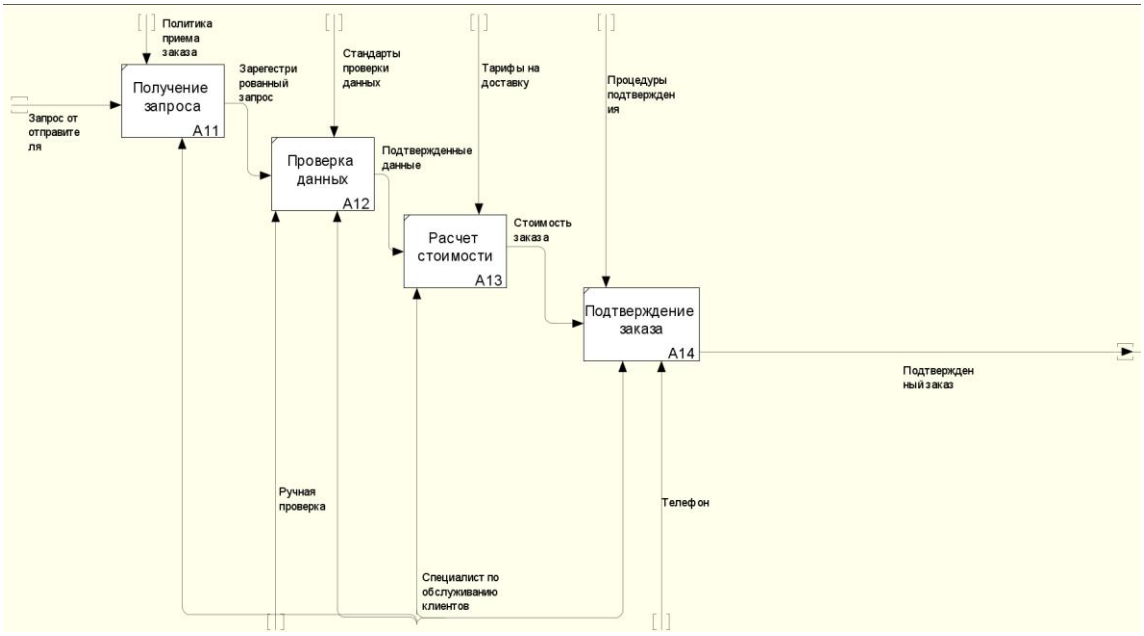

Рисунок А1 – Декомпозиция второго уровня для приема заказа

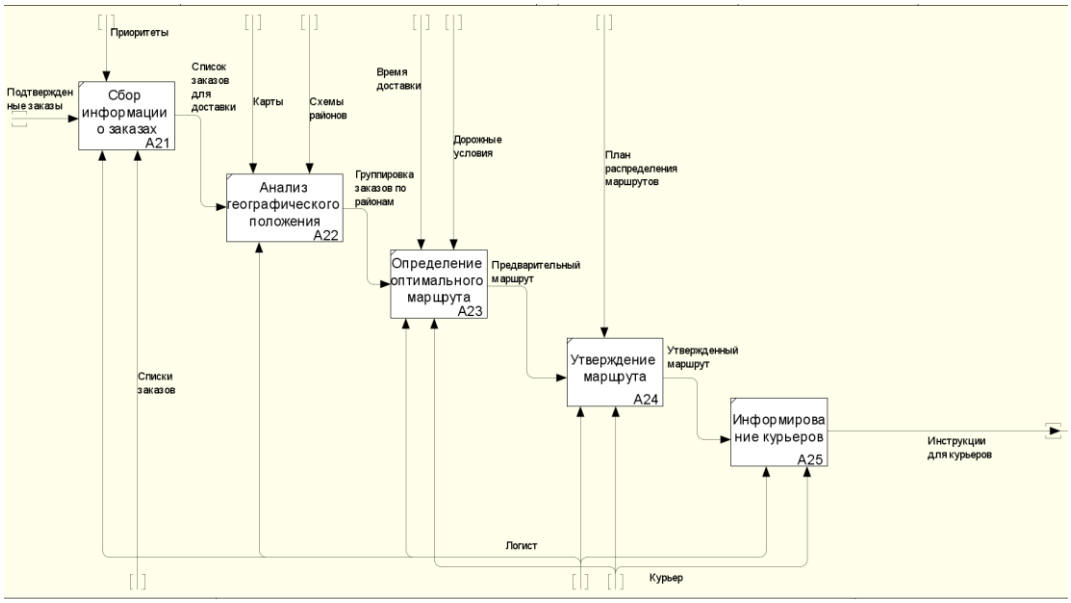

Рисунок А2 – Декомпозиция второго уровня для планирования маршрута

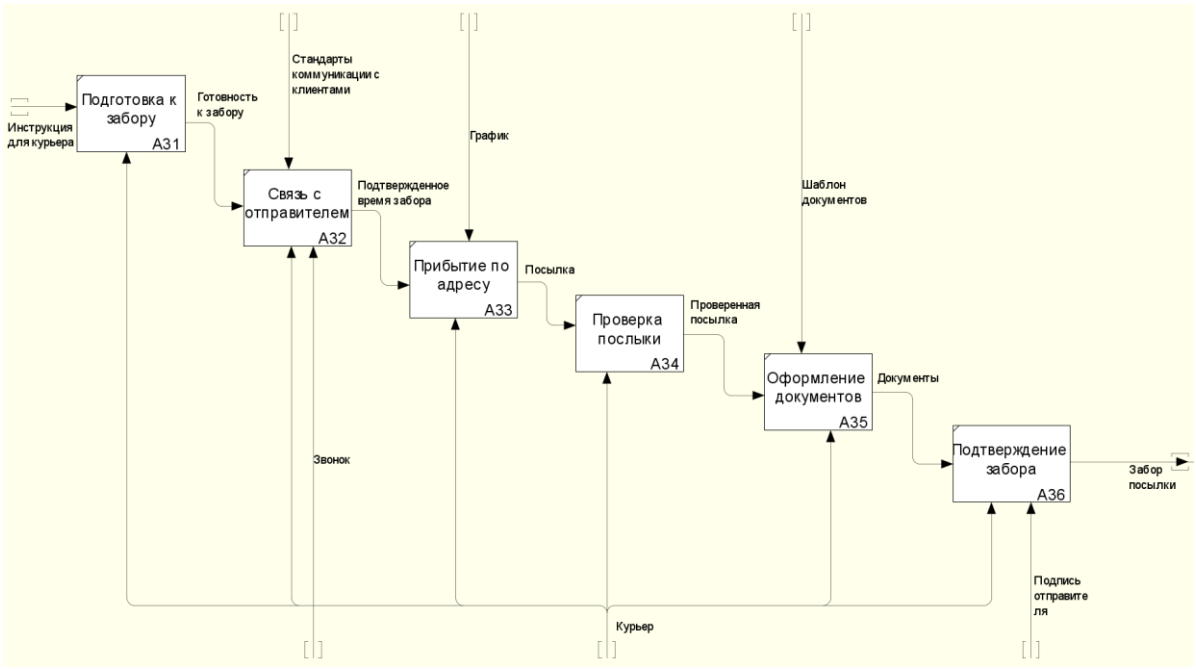

Рисунок А3 – Декомпозиция второго уровня для забора посылки

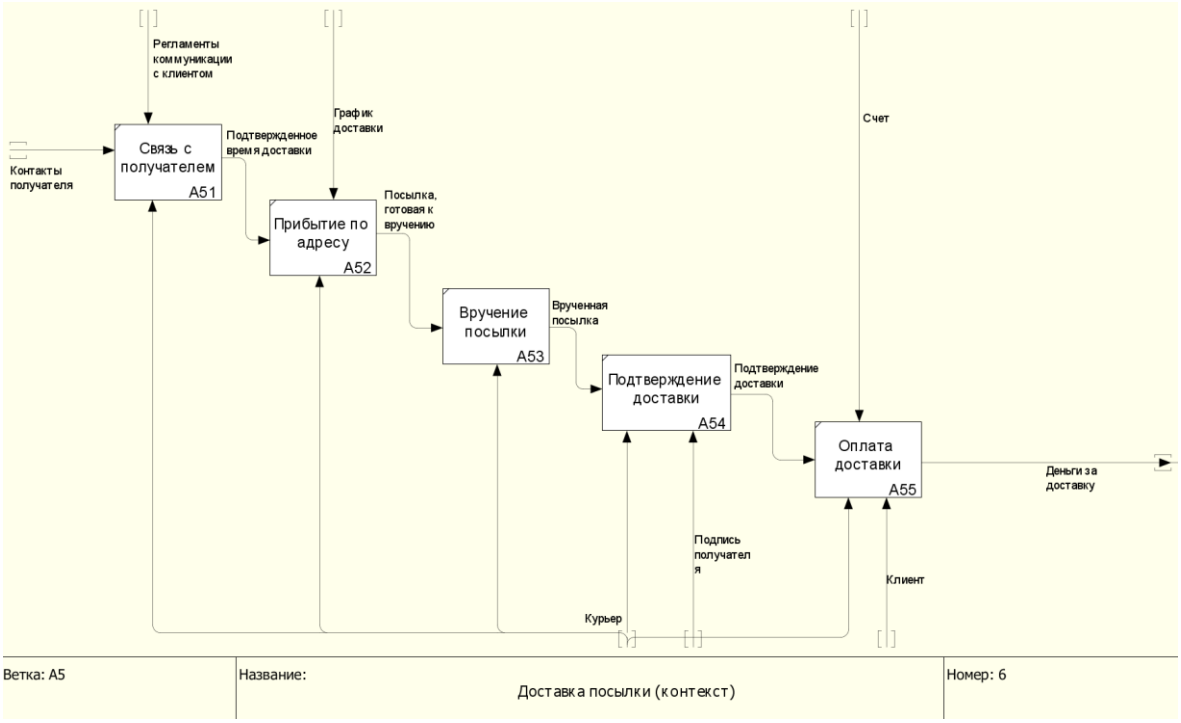

Рисунок А4 – Декомпозиция второго уровня для доставки посылки

Приложение Б

Пример данных коллекций проектируемой БД

Листинг Б1 – Пример данных Users (Пользователи)


```
{
"userId": 1,
"phone": "79152345278",
"password": "1234",
"FIO": "Иванов Иван Иванович",
"createdAt": "2024-02-20T14:30:00.000Z",
"role": "Client"
}
```


Листинг Б2 – Пример данных из коллекции Orders (заказы)


```
{
  "orderId": 1,
  "userId": 1,
  "price_RUR": 15000,
  "adressAt_X": 55.751244,
  "adressAt_Y": 37.618423,
  "adressTo_X": 55.764856,
  "adressTo_Y": 37.578841,
  "Status": "OPEN",
  "created": "2024-09-15T21:00:00.000+00:00"
}
```

Приложение В

Программный код экранов создания заказа

Листинг В1 – фрагмент CreateOrderFragment (создание заказа)

```
class CreateOrderFragment : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_create_order, container, false)

        val createOrderButton: Button = view.findViewById(R.id.createOrderButton)

        createOrderButton.setOnClickListener {
            // Переход к фрагменту выбора точки отправления

findNavController().navigate(R.id.action_createOrderFragment_to_pickStartLocationFragment)
        }

        return view
    }
}
```

Листинг В2 – Фрагмент PickStartLocationFragment (выбор точки забора)

```
class PickStartLocationFragment : Fragment() {
    private lateinit var mapView: MapView
    private lateinit var nextButton: Button
    private lateinit var mapCenterMarker: ImageView
    private lateinit var coordinatesTextView: TextView
    private var startPoint: Point? = null
```

```
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_pick_start_location, container, false)
        mapView = view.findViewById(R.id.mapView)
        nextButton = view.findViewById(R.id.nextButton)
        mapCenterMarker = view.findViewById(R.id.mapCenterMarker)
        coordinatesTextView = view.findViewById(R.id.coordinatesTextView)
        nextButton.visibility = View.GONE
        MapKitFactory.initialize(requireContext())
        val yandexMap = mapView.map
        yandexMap.move(CameraPosition(Point(55.751244, 37.618423), 10.0f, 0.0f, 0.0f))
        yandexMap.addCameraListener(object : CameraListener {
            override fun onCameraPositionChanged(
                map: com.yandex.mapkit.map.Map,
                cameraPosition: CameraPosition,
                cameraUpdateReason: CameraUpdateReason,
                finished: Boolean
            ) {
                Log.d("PickStartLocation", "Camera moved: $cameraPosition, Finished: $finished,
Reason: $cameraUpdateReason")
                if (finished) {
                    val latitude = cameraPosition.target.latitude
                    val longitude = cameraPosition.target.longitude
                    if (startPoint == null || latitude != startPoint!!.latitude || longitude !=
startPoint!!.longitude) {
                        startPoint = cameraPosition.target
                        Log.d("PickStartLocation", "Updated coordinates: $latitude, $longitude")
                        coordinatesTextView.text = "Координаты: $latitude, $longitude"
                        nextButton.visibility = View.VISIBLE
                    } else {
                        Log.d("PickStartLocation", "Coordinates have not changed.")
                    }
```

```
            }
        }
    })
    nextButton.setOnClickListener {
        if (startPoint != null) {
            val bundle = Bundle().apply {
                putFloat("startLat", startPoint!!.latitude.toFloat())
                putFloat("startLng", startPoint!!.longitude.toFloat())
            }
            findNavController().navigate(R.id.pickEndLocationFragment, bundle)
        } else {
            Toast.makeText(requireContext(), "Пожалуйста, выберите точку отправления",
Toast.LENGTH_SHORT).show()
        }
    }
    return view
  }
  override fun onStart() {
    super.onStart()
    mapView.onStart()
    MapKitFactory.getInstance().onStart()
  }
  override fun onStop() {
    mapView.onStop()
    MapKitFactory.getInstance().onStop()
    super.onStop()
  }
  override fun onDestroyView() {
    super.onDestroyView()
  }
}
```

Листинг В3 – Фрагмент PickEndLocationFragment (выбор точки доставки)

```
class PickEndLocationFragment : Fragment() {
    private lateinit var mapView: MapView
    private lateinit var priceTextView: TextView
    private lateinit var finishButton: Button
    private lateinit var coordinatesTextView: TextView
    private lateinit var mapCenterMarker: ImageView
    private var endPoint: Point? = null
    private val args: PickEndLocationFragmentArgs by navArgs()
    private var startLat: Float = 0f
    private var startLng: Float = 0f
    private lateinit var apiService: ApiService
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_pick_end_location, container, false)
        mapView = view.findViewById(R.id.mapView)
        priceTextView = view.findViewById(R.id.priceTextView)
        finishButton = view.findViewById(R.id.finishButton)
        coordinatesTextView = view.findViewById(R.id.coordinatesTextView)
        mapCenterMarker = view.findViewById(R.id.mapCenterMarker)
        startLat = arguments?.getFloat("startLat") ?: 0f
        startLng = arguments?.getFloat("startLng") ?: 0f
        apiService = ApiClient.createApi(ApiService::class.java)
        MapKitFactory.initialize(requireContext())
        val yandexMap = mapView.map
        yandexMap.move(CameraPosition(Point(55.751244, 37.618423), 10.0f, 0.0f, 0.0f))
        finishButton.setOnClickListener {
            if (endPoint != null) {
```

```
            val price = calculatePrice(startLat, startLng, endPoint!!.latitude, endPoint!!.longitude)
            createOrder(startLat.toDouble(), startLng.toDouble(), endPoint!!.latitude,
endPoint!!.longitude, price)
        } else {
            Toast.makeText(requireContext(), "Выберите точку доставки",
Toast.LENGTH_SHORT).show()
        }
    }
    yandexMap.addCameraListener(object : CameraListener {
        override fun onCameraPositionChanged(
            map: com.yandex.mapkit.map.Map,
            cameraPosition: CameraPosition,
            reason: CameraUpdateReason,
            finished: Boolean
        ) {
            Log.d("PickEndLocation", "Camera moved: $cameraPosition, Finished: $finished,
Reason: $reason")
            if (finished) {
                endPoint = cameraPosition.target
                val latitude = endPoint!!.latitude
                val longitude = endPoint!!.longitude
                Log.d("PickEndLocation", "Updated coordinates: $latitude, $longitude")
                coordinatesTextView.text = "Координаты: $latitude, $longitude"
                val distance = calculateDistance(
                    startLat,
                    startLng,
                    latitude.toFloat(),
                    longitude.toFloat()
                )
                val price = (distance * 150).toInt()
                priceTextView.text = "Стоимость доставки: %d Р".format(price)
            }
```

```
            }
        })
        return view
    }
    private fun calculateDistance(lat1: Float, lng1: Float, lat2: Float, lng2: Float): Double {
        val earthRadius = 6371.0
        val dLat = Math.toRadians((lat2 - lat1).toDouble())
        val dLng = Math.toRadians((lng2 - lng1).toDouble())
        val a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
                Math.cos(Math.toRadians(lat1.toDouble())) *
Math.cos(Math.toRadians(lat2.toDouble())) *
                Math.sin(dLng / 2) * Math.sin(dLng / 2)
        val c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
        return earthRadius * c
    }
    private fun calculatePrice(startLat: Float, startLng: Float, endLat: Double, endLng: Double):
Int {
        val distance = calculateDistance(startLat, startLng, endLat.toFloat(), endLng.toFloat())
        return (distance * 150).toInt()
    }
    private fun createOrder(startLat: Double, startLng: Double, endLat: Double, endLng: Double,
price: Int) {
        val sharedPref = requireContext().getSharedPreferences("UserPrefs",
android.content.Context.MODE_PRIVATE)
        val userId = sharedPref.getInt("userId", 0)
        if (userId == 0) {
            Toast.makeText(requireContext(), "Ошибка: пользователь не авторизован",
Toast.LENGTH_SHORT).show()
            return
        }
        val request = CreateOrderRequest(
            userId = userId,
```

```
            price_RUR = price,
            adressAt_X = startLat,
            adressAt_Y = startLng,
            adressTo_X = endLat,
            adressTo_Y = endLng
        )
        apiService.createOrder(request).enqueue(object : Callback<CreateOrderResponse> {
            override fun onResponse(call: Call<CreateOrderResponse>, response:
Response<CreateOrderResponse>) {
                if (response.isSuccessful) {
                    Toast.makeText(requireContext(), "Заказ создан! ID заказа:
${response.body()?.orderId}", Toast.LENGTH_SHORT).show()
                } else {
                    Toast.makeText(requireContext(), "Ошибка создания заказа",
Toast.LENGTH_SHORT).show()
                }
            }
            override fun onFailure(call: Call<CreateOrderResponse>, t: Throwable) {
                Toast.makeText(requireContext(), "Ошибка сети: ${t.message}",
Toast.LENGTH_SHORT).show()
            }
        })
    }
    override fun onStart() {
        super.onStart()
        mapView.onStart()
        MapKitFactory.getInstance().onStart()
    }
    override fun onStop() {
        mapView.onStop()
        MapKitFactory.getInstance().onStop()
        super.onStop()
    }
}
```

Приложение Г

Апи сервис мобильного и десктопного приложения для обращений к серверу

Листинг Г1 – API сервис мобильного приложения

```
data class LoginRequest(val phone: String, val password: String)
data class LoginResponse(val message: String, val userId: Int)
data class RegisterRequest(val phone: String, val password: String, val fio: String)
data class RegisterResponse(val message: String)
data class UserProfile(
    val userId: Int,
    val phone: String,
    val FIO: String,
    val password: String? = null
)
data class UserProfileResponse(
    val userId: Int,
    val phone: String,
    val FIO: String,
    val createdAt: String,
    val role: String
)
data class CreateOrderResponse(
    val message: String,
    val orderId: Int
)
data class CreateOrderRequest(
    val userId: Int,
    val price_RUR: Int,
    val adressAt_X: Double,
    val adressAt_Y: Double,
    val adressTo_X: Double,
    val adressTo_Y: Double
```

```
)
interface ApiService {
    @POST("/users/login")
    fun loginUser(@Body request: LoginRequest): Call<LoginResponse>

    @POST("/users/register")
    fun registerUser(@Body request: RegisterRequest): Call<RegisterResponse>

    @GET("/users/profile/{userId}")
    fun getProfile(@Query("userId") userId: Int): Call<UserProfileResponse>

    @PUT("/users/profile/{userId}")
    fun updateUserProfile(@Body profile: UserProfile): Call<Void>

    @GET("users/{userId}/orders")
    fun getUserOrders(@Path("userId") userId: Int): Call<List<Order>>

    @POST("/orders/create")
    fun createOrder(@Body request: CreateOrderRequest): Call<CreateOrderResponse>
}
```

Листинг Г2 – API сервис десктопного приложения

```
namespace OfficeApp.Services
{
    public class ApiClient
    {
        private readonly HttpClient _httpClient;
        private string _authToken;
        public ApiClient()
```

```
    {
        _httpClient = new HttpClient { BaseAddress = new System.Uri("http://localhost:3000/")
};
    }
    public async Task<bool> LoginUserAsync(string phone, string password)
    {
        var json = JsonConvert.SerializeObject(new { phone, password });
        var content = new StringContent(json, Encoding.UTF8, "application/json");
        try
        {
            var response = await _httpClient.PostAsync("users/login", content);
            response.EnsureSuccessStatusCode();
            var responseBody = await response.Content.ReadAsStringAsync();
            var tokenResponse = JsonConvert.DeserializeObject<dynamic>(responseBody);
            _authToken = tokenResponse.token;
            return true;
        }
        catch (HttpRequestException e)
        {
            MessageBox.Show("Login failed: " + e.Message);
            return false;
        }
    }
    private void AddAuthorizationHeader()
    {
        if (!string.IsNullOrEmpty(_authToken))
        {
            _httpClient.DefaultRequestHeaders.Authorization = new
System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", _authToken);
        }
    }
    public async Task<string> GetUsersAsync()
```

```
{
    AddAuthorizationHeader();
    var response = await _httpClient.GetAsync("users");
    response.EnsureSuccessStatusCode();
    return await response.Content.ReadAsStringAsync();
}
public async Task<string> GetOrdersAsync()
{
    AddAuthorizationHeader();
    try
    {
        var response = await _httpClient.GetAsync("orders");
        response.EnsureSuccessStatusCode();
        return await response.Content.ReadAsStringAsync();
    }
    catch (HttpRequestException e)
    {
        MessageBox.Show("Error fetching orders: " + e.Message);
        return null;
    }
}
public async Task<string> RegisterUserAsync(User user)
{
    var json = JsonConvert.SerializeObject(user);
    var content = new StringContent(json, Encoding.UTF8, "application/json");
    var response = await _httpClient.PostAsync("users/register", content);
    response.EnsureSuccessStatusCode();
    return await response.Content.ReadAsStringAsync();
}
public async Task<bool> UpdateOrderStatusAsync(int orderId, string newStatus)
{
```

```
            AddAuthorizationHeader();
            var content = new StringContent(JsonConvert.SerializeObject(new { Status = newStatus
}), Encoding.UTF8, "application/json");
            var response = await _httpClient.PutAsync($"orders/{orderId}/status", content);
            return response.IsSuccessStatusCode;
        }
        public async Task<string> GetUserOrdersAsync(string userId)
        {
            var response = await _httpClient.GetAsync($"users/{userId}/orders");
            response.EnsureSuccessStatusCode();
            return await response.Content.ReadAsStringAsync();
```

Приложение Д

Программный код десктопного приложения

Листинг Д1 – OrderDetailWindow (Информация о заказе)

```
namespace OfficeApp  {
   public partial class OrderDetailsWindow : Window
   {
      private readonly Orders _order;
      private readonly ApiClient _apiClient;
      private readonly double _adressAt_X;
      private readonly double _adressAt_Y;
      private readonly double _adressTo_X;
      private readonly double _adressTo_Y;
      bool mapInitialized = false;
      public OrderDetailsWindow(Orders order)
      {
         InitializeComponent();
         _order = order;
         _apiClient = new ApiClient();
         _adressAt_X = order.adressAt_X;
         _adressAt_Y = order.adressAt_Y;
         _adressTo_X = order.adressTo_X;
         _adressTo_Y = order.adressTo_Y;
         DisplayOrderDetails(order);
         LoadMap();    }
      private void DisplayOrderDetails(Orders order)
      {
         OrderIdTextBlock.Text = order.id.ToString();
         UserIdTextBlock.Text = order.userId.ToString();
         AddressAtTextBlock.Text = $"{order.adressAt_X}, {order.adressAt_Y}";
         AddressToTextBlock.Text = $"{order.adressTo_X}, {order.adressTo_Y}";
         StatusTextBlock.Text = order.Status;
         CreatedTextBlock.Text = order.created.ToString();
```

```
        price_RURTextBlock.Text = order.price_RUR.ToString();
        btn_NextStatus.IsEnabled = order.Status == "OPEN";
    }
    private async void MarkAsInProgress_Click(object sender, RoutedEventArgs e)
    {
        if (_order.Status == "OPEN")
        {
            try
            {
                bool success = await _apiClient.UpdateOrderStatusAsync(_order.id,
"IN_PROGRESS");
                if (success)
                {
                    _order.Status = "IN_PROGRESS";
                    StatusTextBlock.Text = _order.Status;
                    btn_NextStatus.IsEnabled = false;
                } else
                {
                    MessageBox.Show("Failed to update order status.");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error updating order status: " + ex.Message);
            }
        }
    }
    private async void LoadMap()
    {
        string mapFilePath =
@"C:\Users\MetMi\OneDrive\Документы\OfficeApp\OfficeApp\map.html";
        string formattedPath = $"file:///{mapFilePath.Replace("\\", "/")}";
```

```
            await MapWebView.EnsureCoreWebView2Async(null);
            MapWebView.CoreWebView2.Navigate(mapFilePath);
        mapInitialized = true;
    }
    private void ShowMapAt_Click(object sender, RoutedEventArgs e)
    {
        if (MapWebView.CoreWebView2 != null)
        {
            var coordinatesAt_X =
_adressAt_X.ToString(System.Globalization.CultureInfo.InvariantCulture);
            var coordinatesAt_Y =
_adressAt_Y.ToString(System.Globalization.CultureInfo.InvariantCulture);
            var script = $"updateMap({coordinatesAt_X}, {coordinatesAt_Y});";
            MapWebView.CoreWebView2.ExecuteScriptAsync(script).ContinueWith(task =>
            {
                if (task.Exception != null)
                {
                    MessageBox.Show($"Ошибка JavaScript: {task.Exception.Message}");
                }
            });
        }
        else
        {
            MessageBox.Show("WebView2 не инициализирован.");
        }
    }
    private void ShowMapTo_Click(object sender, RoutedEventArgs e)
    {
        if (MapWebView.CoreWebView2 != null)
        {
```

```
            var coordinatesTo_X =
_adressTo_X.ToString(System.Globalization.CultureInfo.InvariantCulture);
            var coordinatesTo_Y =
_adressTo_Y.ToString(System.Globalization.CultureInfo.InvariantCulture);
            var script = $"updateMap({coordinatesTo_X}, {coordinatesTo_Y});";
            MapWebView.CoreWebView2.ExecuteScriptAsync(script).ContinueWith(task =>
            {                   if (task.Exception != null)
               {                        MessageBox.Show($"Ошибка JavaScript:
{task.Exception.Message}");
               }
            });
         }
         else
         {
            MessageBox.Show("WebView2 не инициализирован.");
         }
      }
      private void MapWebView_NavigationCompleted(object sender,
CoreWebView2NavigationCompletedEventArgs e)
      {
         if (e.IsSuccess)
         {
            var coordinatesAt_X =
_adressAt_X.ToString(System.Globalization.CultureInfo.InvariantCulture);
            var coordinatesAt_Y =
_adressAt_Y.ToString(System.Globalization.CultureInfo.InvariantCulture);
            var script = $"updateMap({coordinatesAt_X}, {coordinatesAt_Y});";
            MapWebView.CoreWebView2.ExecuteScriptAsync(script).ContinueWith(task =>
            {
               if (task.Exception != null)
               {
                  MessageBox.Show($"Ошибка JavaScript: {task.Exception.Message}");
               }
            });
```

```
    }
    else
    {
        MessageBox.Show($"Ошибка загрузки карты: {e.WebErrorStatus}");
    }
}
```

Приложение Е

Программный код автотестов

Листинг Е1 – Автотесты для мобильного приложения.

```
@RunWith(AndroidJUnit4::class)
class LoginAndProfileTest {
    @get:Rule
    var activityScenarioRule = ActivityScenarioRule(LoginActivity::class.java)
    @Test
    fun testLoginAndGoToProfile() {
        onView(withId(R.id.phoneEditText))
            .perform(typeText("79152345278"), closeSoftKeyboard())
        onView(withId(R.id.passwordEditText))
            .perform(typeText("1234"), closeSoftKeyboard())
        onView(withId(R.id.loginButton))
            .perform(click())
        onView(withId(R.id.bottomNavigationView))
            .check(matches(isDisplayed()))
        onView(withId(R.id.navigation_profile))
            .perform(click())
        onView(withId(R.id.phoneEditText)).check(matches(withText("79152345278")))
        onView(withId(R.id.fioEditText)).check(matches(withText("Иванов Иван Иванович")))
    }
    @Test
    fun testLoginAndGoToOrderHistory() {
        onView(withId(R.id.phoneEditText))
            .perform(typeText("79152345278"), closeSoftKeyboard())
        onView(withId(R.id.passwordEditText))
            .perform(typeText("1234"), closeSoftKeyboard())
        onView(withId(R.id.loginButton))
            .perform(click())
        onView(withId(R.id.bottomNavigationView))
            .check(matches(isDisplayed()))
```

```
        onView(withId(R.id.navigation_order_history))
            .perform(click())
        onView(withId(R.id.ordersRecyclerView))
            .check(matches(isDisplayed()))
        onView(withId(R.id.ordersRecyclerView))
            .check(matches(hasMinimumChildCount(1)))
    }

    @Test
    fun testCreateOrderIsExist() {
        onView(withId(R.id.phoneEditText))
            .perform(typeText("79152345278"), closeSoftKeyboard())
        onView(withId(R.id.passwordEditText))
            .perform(typeText("1234"), closeSoftKeyboard())
        onView(withId(R.id.loginButton))
            .perform(click())
        onView(withId(R.id.bottomNavigationView))
            .check(matches(isDisplayed()))
        onView(withId(R.id.createOrderButton))
            .check(matches(isDisplayed()))
    }
}
```

Листинг Е2 – Автотесты для десктопного приложения

```
namespace OfficeAppTests
{
    [TestFixture]
    public class Tests
    {
        {
        private AuthService _authService;
        private Mock<IUserRepository> _userRepositoryMock;
```

```
[SetUp]
public void Setup()
{
    _userRepositoryMock = new Mock<IUserRepository>();
    _authService = new AuthService(_userRepositoryMock.Object);
}
[Test]
public void Login_WithValidCredentials_ReturnsTrue()
{
    string phone = "79152345278";
    string password = "1234";
    _userRepositoryMock.Setup(repo => repo.IsValidUser(phone, password)).Returns(true);
    var result = _authService.Login(phone, password);
    Assert.IsTrue(result);
}
[Test]
public void Login_WithInvalidCredentials_ReturnsFalse()
{
    string phone = "79152345278";
    string password = "wrongpassword";
    _userRepositoryMock.Setup(repo => repo.IsValidUser(phone, password)).Returns(false);
    var result = _authService.Login(phone, password);
    Assert.IsFalse(result);
}
[Test]
public void TestLoginAndOpenUserInfo()
{
    var phoneField = _session.FindElementByAccessibilityId("phoneInputField");
    var passwordField = _session.FindElementByAccessibilityId("passwordInputField");
    var loginButton = _session.FindElementByAccessibilityId("loginButton");
    phoneField.SendKeys("79152345278");
    passwordField.SendKeys("1234");
    loginButton.Click();
```

```
            var userInfoButton = _session.FindElementByAccessibilityId("userInfoButton");
            userInfoButton.Click();
            var userNameField = _session.FindElementByAccessibilityId("userNameField");
            var userPhoneField = _session.FindElementByAccessibilityId("userPhoneField");
            Assert.AreEqual("Иванов Иван Иванович", userNameField.Text);
            Assert.AreEqual("79152345278", userPhoneField.Text);
        }
        [TearDown]
        public void TearDown()
        {
            if (_session != null)
            {
                _session.Quit();
            }
        }
    }
}
```

Приложение Ж

Блок-схемы алгоритмов работы основных модулей приложения

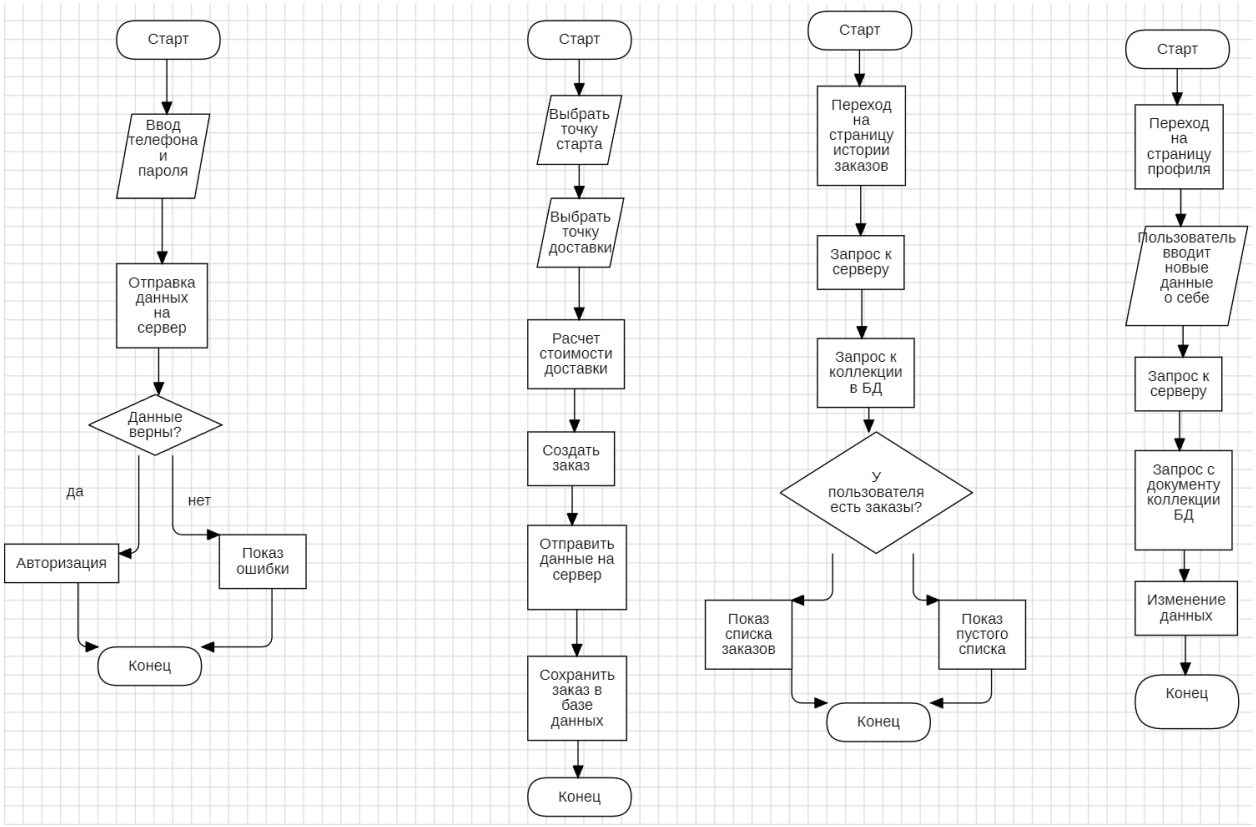

Рисунок Ж1 – Блок-схемы модулей авторизации, создания заказа, показа списка заказов, изменения данных профиля

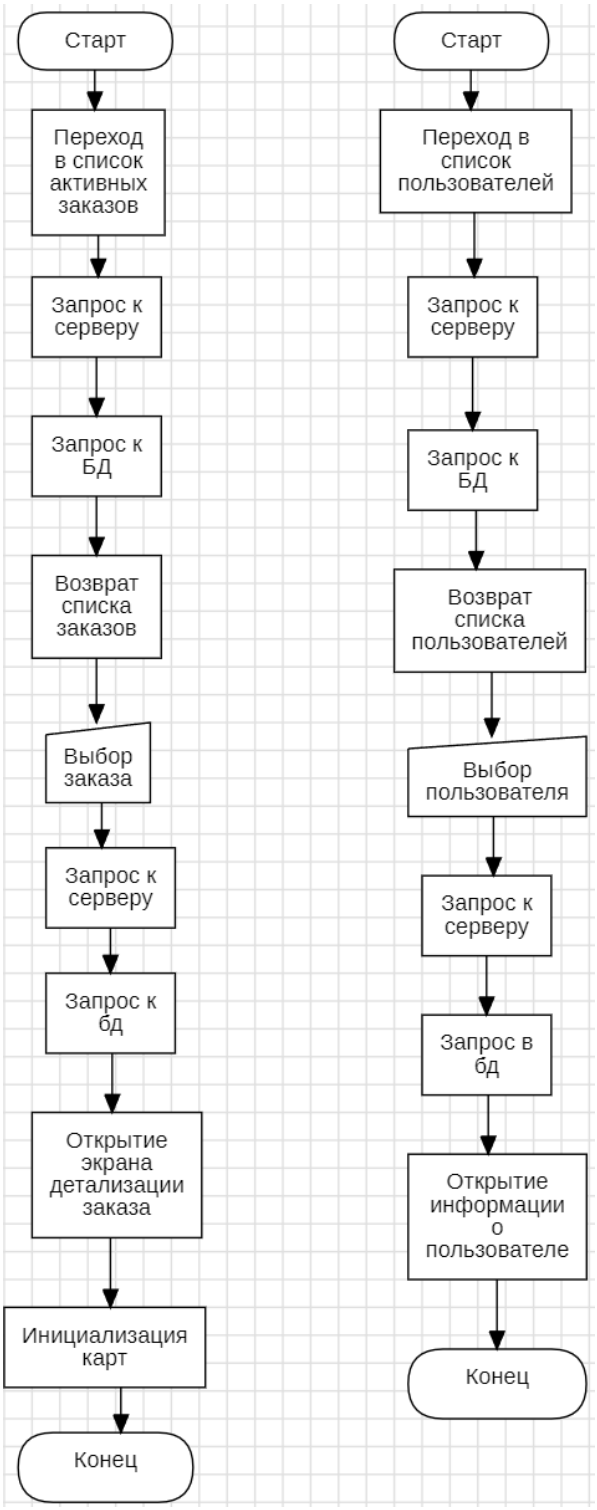

Рисунок Ж2 – Блок-схемы алгоритмов работы загрузки информации о заказе, загрузки информации о пользователе