

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.....	5
1.1 Техническое задание.....	5
1.2. Понятие UML.....	6
1.3 Виды диаграмм UML.....	8
1.3.1 Диаграмма вариантов использования.....	9
1.3.2 Диаграмма классов.....	11
1.3.3 Диаграмма деятельности.....	12
1.3.4 Диаграмма последовательности.....	15
1.4 Сравнительный анализ информационных систем транспортных компаний.....	16
1.5 Обзор методов проектирования и разработки информационных систем.....	20
1.5 Инструментальные средства проектирования ИС.....	22
2 ПРАКТИЧЕСКАЯ ЧАСТЬ.....	23
2.1 Разработка диаграммы вариантов использования.....	23
2.2 Разработка диаграммы классов.....	25
2.3 Разработка диаграммы деятельности.....	27
2.4 Разработка диаграммы последовательности.....	28
2.5 Реализация методов.....	30
2.5.1 Проектирование пользовательского интерфейса.....	30
2.5.2 Разработка структуры приложения.....	33
ЗАКЛЮЧЕНИЕ .....	36
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	37
Приложение А.....	39

## ВВЕДЕНИЕ

Жизнь современной организации, которая, особенно, занимается коммерческой деятельностью, совершенно невозможна без эффективного управления. Одним из самых важных аспектов эффективной работы организации являются системы обработки информации. Системы обработки информации должны соответствовать следующим требованиям:

- обеспечивать получение общих и детализированных отчётов в ходе и по итогам работы;
- обеспечивать получение информации, которая является критической по времени, без задержек;
- давать возможность быстро и эффективно вносить новые данные;
- иметь перспективы для расширения функционала.

Целью курсовой работы является проектирование информационной системы транспортной компании с частичной реализацией приложения для работы с этой системой.

Актуальность данной работы обусловлена тем, что почти любому бизнесу нужно средство эффективной обработки информации, поэтому актуальной становится задача проектирования и создание систем хранения и обработки информации с целью сокращения рутинного, малоэффективного человеческого труда.

Первый раздел содержит техническое задание и теоретические сведения, которые необходимы для успешной реализации проектирования информационной системы.

Второй раздел содержит разработанные диаграммы вариантов использования, классов, деятельности и последовательности. Помимо диаграмм продемонстрирована частично разработанное приложение на основе диаграммы последовательности.

В третьем разделе находится список использованных источников.

В приложении находится листинг программного кода с комментариями к нему.

Результатом выполнения курсовой работы является разработанные диаграммы вариантов использования, классов, деятельности последовательности и частично разработанное приложение на основе диаграммы последовательности.

## 1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

### 1.1 Техническое задание

Цель курсовой работы – рассмотреть теоретические и практические аспекты процесса экспедирования транспортной компании и частично разработать информационную систему.

Задачи курсовой работы:

- проанализировать теоретические основы процесса экспедирования транспортной компании;
- выполнить частичную разработку информационной системы транспортной компании;
- выполнить анализ процесса экспедирования транспортной компании.

Разрабатываемая система должна будет использоваться для хранения и обработки информации о перевозчиках, загрузках и заказах.

Информационная система создаётся для обслуживания логиста организации - он сможет следить за процессом выполнения заказа, вести учёт водителей и заказчиков.

Информационная система должна обеспечивать:

- ведение базы данных (запись, чтение, модификация, удаление);
- хранение информации о заказах;
- хранение информации о перевозчиках;
- хранение информации о заказчиках.

Целью разработки будет являться создание клиентского приложения и базы данных для хранения информации о перевозчиках, заказчиках и заказах. Отслеживать новые заказы и контролировать выполнение действующих. Внедрение данной информационной системы даст возможность сократить

время на обработку информации, что позволит увеличить продуктивность работы сотрудников и вероятно, сократить расходы на логистику.

## **1.2. Понятие UML**

UML – это аббревиатура, обозначающая язык унифицированного моделирования.

UML является графическим языком, который подходит для различных изобразительных средств, таких как презентации, печатная или электронная документация, инструменты для построения диаграмм или специально предназначенных для этого компьютерных инструментов UML. Из-за столь широко используемого стандарта UML преднамеренно не требует использования цвета или другого подобного оформления. Однако некоторые инструменты UML допускают цвет, затенение или даже псевдо-3D-эффект, но у них есть возможность отключить эти функции.

UML имеет синтаксис (грамматику), нотацию и семантику (значение).

UML - это язык, в частности, моделирования. UML позволяет создавать модели, отражение чего-либо в реальном мире или отражение чего-то в запланированном мире. Эти модели можно исследовать, чтобы определить, как может вести себя смоделированная система или как она структурирована, показывая особенности, проектные решения и архитектуру. [1. 5].

В качестве примера диаграммы моделирования будет рассмотрена диаграмма последовательности (рисунок 1).

На данной диаграмме последовательности показано типичное взаимодействие с электронной почтой. В этом взаимодействии участвуют пользователь, клиент и сервер электронной почты.

После того, как пользователь инициирует поведение с сообщением Check Mail, приложение электронной почты начинает взаимодействие с

сервером электронной почты; отправка любых неотправленных сообщений, а затем удаление любой старой почты.

Поведение, которое наглядно демонстрирует эта диаграмма является понятным. Мелкие детали, обозначения (например, стрелки и стиль линий) не являются необходимым для понимания основной информации.

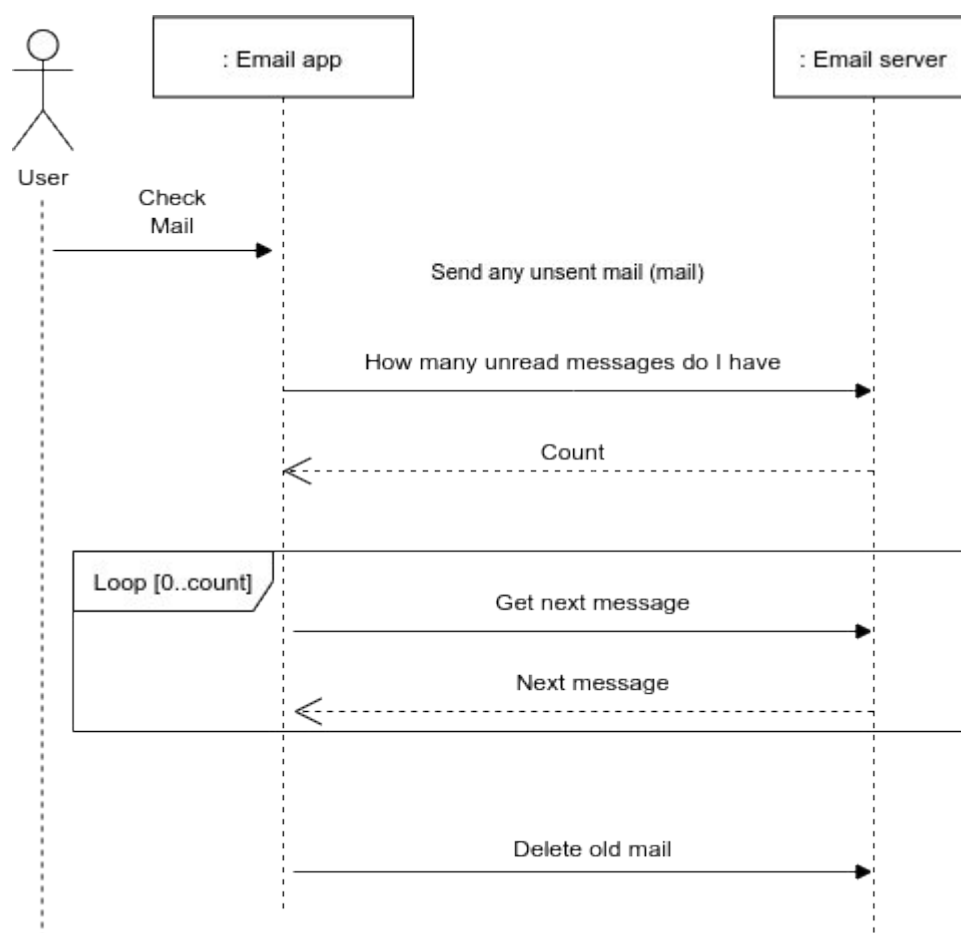


Рисунок 1 — Пример последовательности UML, показывающий обработку электронной почты

Модели UML показывают некоторые детали вещей в системе или области интересов, а также их свойства, отношения и поведение. Моделируемые вещи могут быть физическими или концептуальными. [1,6]

UML отличается от других языков моделирования, прежде всего по мощности и объёму. Например, блок-схемы обычно ограничены показом потока управления, этапов обработки и ввода-вывода. Их эквивалент в UML,

диаграммы деятельности, могут показать на одной диаграмме всё это, а также параллелизм, прерывания и поток данных. [1,7] На других диаграммах UML существует возможность отображать структурную информацию (диаграммы классов), ответы на события состояния (диаграммы состояния) и обмен сообщениями (диаграммы последовательности), их потоковые диаграммы отразить не смогут.

Язык моделирования, который наиболее схож с UML это семейство языков IDEF (Integration Definition). IDEF имеет в своём арсенале 16 типов диаграмм, которые предназначены для целей, являющимися аналогичными UML, но они ограничены в состояниях моделирования и обмена сообщениями. [2, 16]

### **1.3 Виды диаграмм UML**

Продуманные модели очень важны для взаимодействия внутри команды разработчиков, и для взаимопонимания с заказчиком. В конце концов, это позволяет убедиться в "архитектурной согласованности" проекта до того, как он будет реализован в коде. [3, 22]

Люди строят модели сложных систем, потому что не могут описать их полностью, поверхностно пройдясь по ним. Именно поэтому выделяются лишь существенные для конкретной задачи свойства системы и строится её модель, которая отражает её свойства. Модель строится с помощью диаграмм.

Диаграмма — это графическое представление множества элементов. Обычно изображается в виде графа с вершинами (сущностями) и ребрами (отношениями). [3, 25]

Если говорить о проектировании программного обеспечения, то в этой отрасли с помощью диаграмм можно визуализировать систему с различных точек зрения. Одна из диаграмм, например, может описывать взаимодействие пользователя с системой, другая - изменение состояния системы в процессе её

работы, третья же - взаимодействие между собой элементов системы и т. д. Важно понимать, что ни одна отдельная диаграмма не является моделью. Диаграмма - лишь средство визуализации модели и эти два понятия следует различать. Лишь набор диаграмм составляет модель системы и наиболее полно её описывает, но не одна диаграмма, которая вырвана из контекста.

UML 1.5 определял двенадцать типов диаграмм, разделенных на три группы:

- четыре типа диаграмм представляют статическую структуру приложения;
- пять представляют поведенческие аспекты системы;
- три представляют физические аспекты функционирования системы (диаграммы реализации). [3, 26]

Текущая версия UML 2.1 внесла не слишком много изменений. Диаграммы слегка изменились внешне (появились фреймы и другие визуальные улучшения), немного усовершенствовалась нотация, некоторые диаграммы получили новые наименования. [3, 26]

Далее будет обзор диаграмм построение которых будет в процессе выполнения курсовой работы.

### **1.3.1 Диаграмма вариантов использования**

Все системы, в том числе и программные проектируются с учётом того, что в процессе своей работы они будут использоваться людьми или взаимодействовать с другими системами. Сущности, с которыми система взаимодействует в процессе своей работы называются экторами. Каждый эктор ожидает, что система будет себя вести строго определённо предсказуемым образом.



Эктор (actor) — это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Эктором может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности.

Графически эктор изображается либо "человечком", либо символом класса, с соответствующим стереотипом. На рисунке 2 показано графическое изображение эктора.

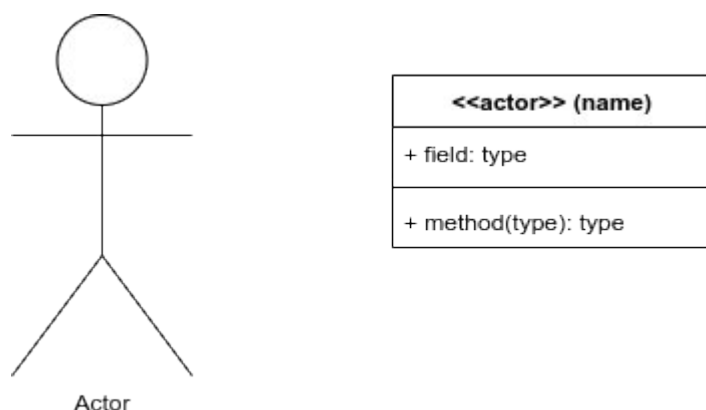


Рисунок 2 — графическое изображение эктора.

Прецедент (use-case) — описание множества последовательных событий (включая варианты), выполняемых системой, которые приводят к наблюдаемому эктором результату. Прецедент представляет поведение сущности, описывая взаимодействие между экторами и системой. Прецедент не показывает, как достигается некоторый результат, а только что именно выполняется [3, 28].

Прецеденты обозначаются в виде эллипса (рисунок 3).

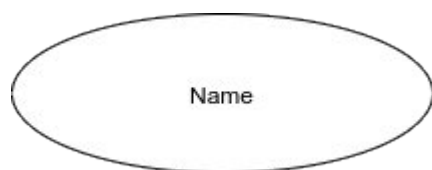


Рисунок 3 — графическое представление прецедента.

Следует отметить, что иногда на диаграммах вариантов использования границы системы обозначают прямоугольником, в верхней части которого может быть указано название системы. Таким образом, прецеденты — действия, выполняемые системой в ответ на действия актора.

Из всего сказанного выше можно понять, что диаграммы вариантов использования относятся к той группе диаграмм, которые представляют динамические или поведенческие аспекты системы. Это наиболее подходящее средство для достижения взаимопонимания между разработчиками, экспертами и конечными пользователями продукта.

Подводя итоги, можно выделить цели создания диаграмм вариантов использования:

- определение границ, а так же контекста моделируемой предметной области на ранних этапах проектирования;
- формирование общих требований к поведению проектируемой системы;
- разработка концептуальной модели системы для её последующей детализации.
- подготовка документации для взаимодействия с заказчиком и пользователями системы.

### **1.3.2 Диаграмма классов**

Класс (class) — категория вещей, которые имеют общие атрибуты и операции.

Класс представляет из себя описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. При проектировании объектно - ориентированных систем диаграммы классов обязательны.

Диаграмма классов — это набор статических, декларированных элементов модели. Такие диаграммы могут применяться при прямом проектировании (в процессе разработки новой системы) и при обратном - описании существующих и используемых систем.

На рисунке 4 изображён пример диаграммы классов.

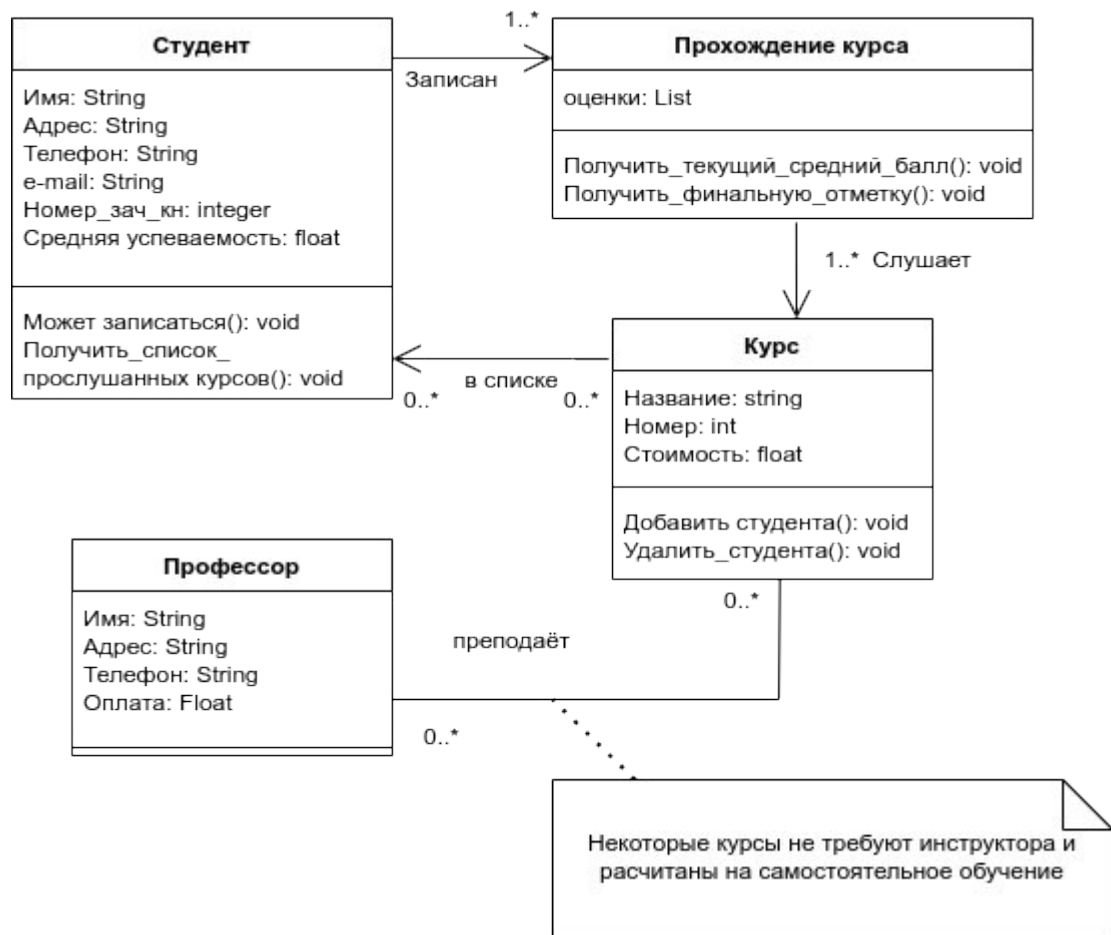


Рисунок 4 — Пример диаграммы классов

В данной диаграмме классов помимо кратности обозначены свойства (и их типы) и операции. Данная диаграмма производит впечатление набора классов для реализации, а не просто описания предметной области.

### 1.3.3 Диаграмма деятельности

Диаграмма деятельности в UML предназначена для описания логики процедур, бизнес-процессов и потоков работ. Во многих случаях они похожи

на блок-схемы. Разница между диаграммами деятельности и нотацией блок-схем заключается в том, что первые поддерживают параллельные процессы. [1, 61]

Самое большое достоинство диаграмм деятельности заключается в том, что они поддерживают и стимулируют применение параллельных процессов. Благодаря этому они представляют собой мощное средство моделирования потоков работ. В наибольшей степени их мощь может проявиться в случае применения UML как языка программирования.

Диаграмма деятельности – это граф, состоящий из узлов (Activity Nodes), соединенных направленными ребрами (ActivityEdges).

На диаграмме деятельности существует три типа узлов:

1.Исполняемые узлы (ExecutableNodes) показывают выполняемые действия в процессе осуществления деятельности.

2. Управляющие узлы (ControlNodes) позволяют описывать переходы от одного действия другим действиям в зависимости от определенных условий; осуществлять распараллеливание деятельности; проводить синхронизацию параллельных потоков деятельности.

3.Объектные узлы (ObjectNodes) описывают данные, используемые в процессе выполнения деятельности.

На диаграмме деятельности существует два типа ребер:

1.Ребра потоков управления (ControlFlow) показывают переходы от одного действия к другому действию.

2.Ребра потоков объектов(ObjectFlow) показывают передачу данных от одного объекта данных к другому. [4, 72]

На рисунке 5 показана диаграмма деятельности на примере работы веб-сервера.

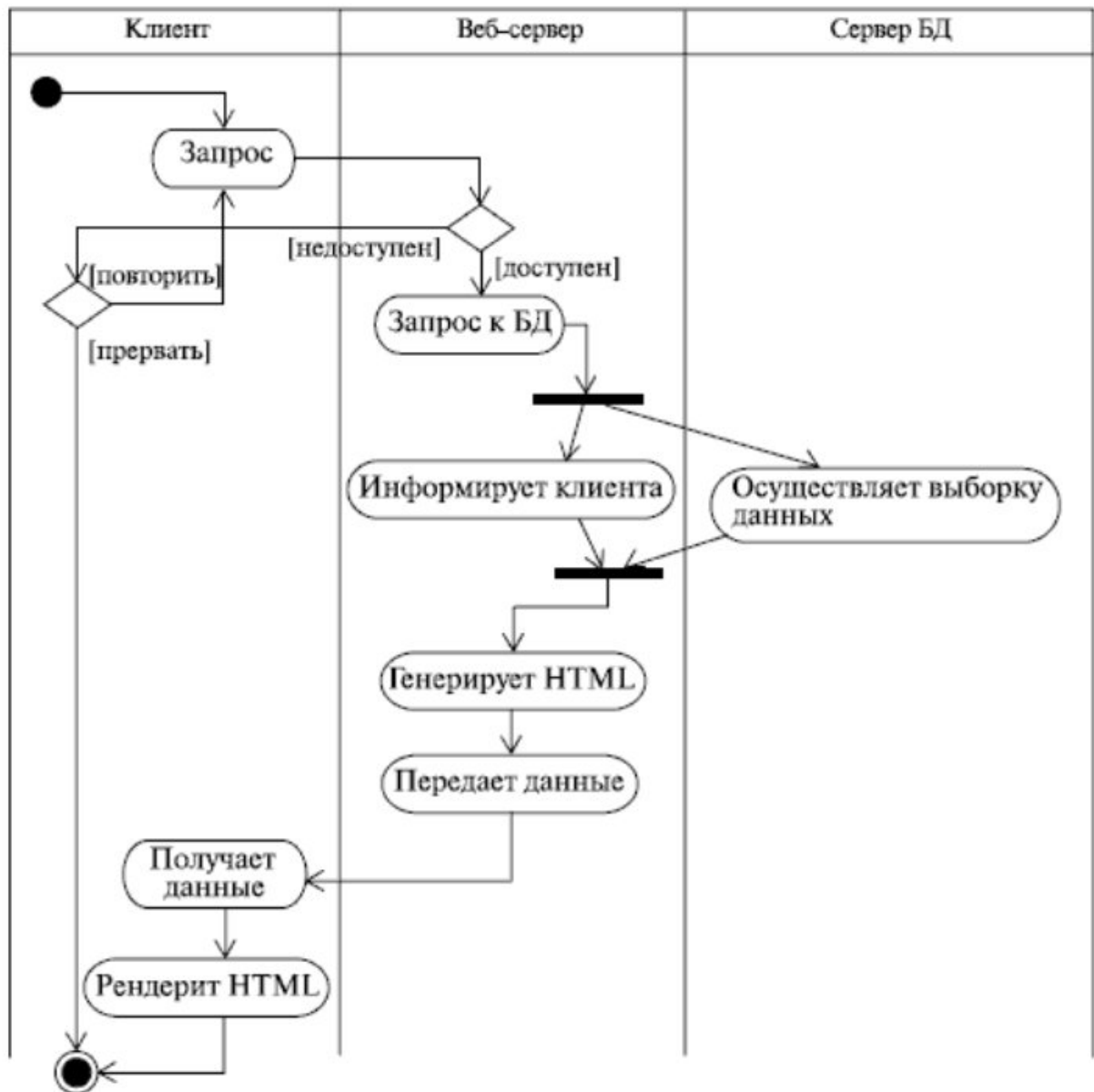


Рисунок 5 — Пример диаграммы деятельности работы веб-сервера

Исходя из вышеперечисленного можно сделать вывод, что диаграмма деятельности описывает некоторые поведение в моделируемой системе. Она применяется на разных этапах разработки программного обеспечения и может быть использована для таких целей как:

- описание вариантов использования;
- описание бизнес-процессов;
- описание алгоритмов программ.

### 1.3.4 Диаграмма последовательности

Диаграмма последовательностей относится к диаграммам взаимодействия UML, описывающим поведенческие аспекты системы, но рассматривает взаимодействие объектов во времени. Другими словами, диаграмма последовательностей отображает временные особенности передачи и приема сообщений объектами.

Это отличное средство документирования проекта с точки зрения сценариев использования. Диаграммы последовательностей обычно содержат объекты, которые будут взаимодействовать в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Часто возвращаемые результаты возвращают только в том случае, если это не очевидно из контекста.

Объекты обозначаются прямоугольниками с подчеркнутыми именами (чтобы отличить их от классов), сообщения (вызовы методов) - линиями со стрелками, возвращаемые результаты - пунктирными линиями со стрелками. Прямоугольники на вертикальных линиях под каждым из объектов показывают "время жизни" (фокус) объектов. Довольно часто их не изображают на диаграмме, все это зависит от индивидуального стиля проектирования

Пример диаграммы последовательности взаимодействия электронной почты можно увидеть на рисунке 6.

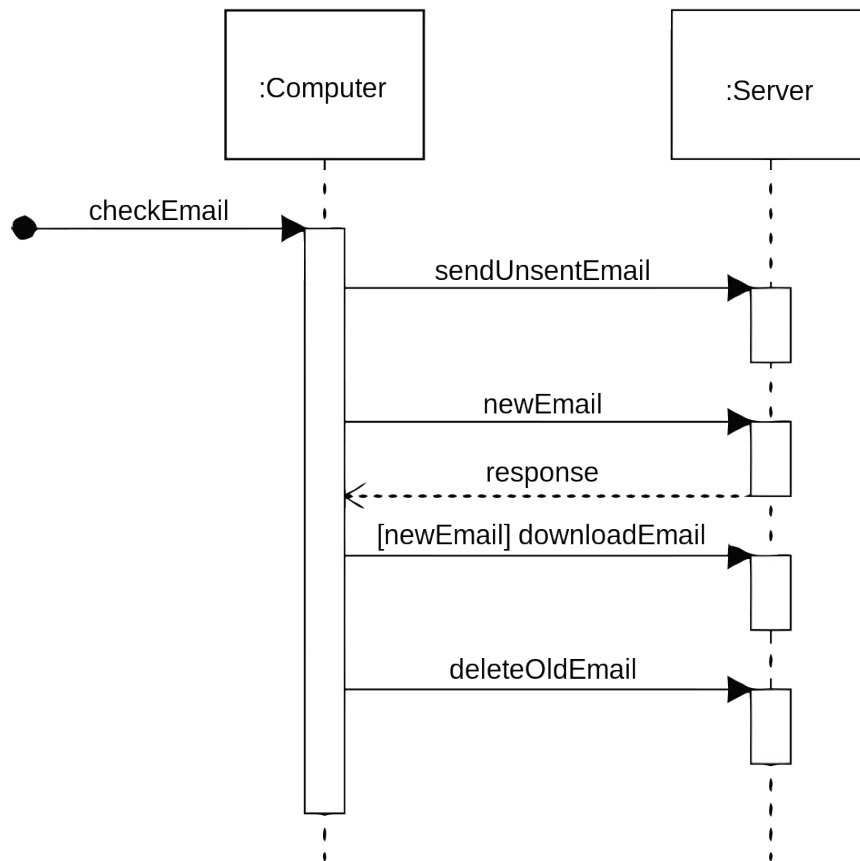


Рисунок 6 — Пример диаграммы последовательности

#### 1.4 Сравнительный анализ информационных систем транспортных компаний

До того как сформулировать подробные требования к разрабатываемому проекту, необходимо проанализировать уже существующие продукты. Подробный анализ нужен прежде всего для того, чтобы оценить в полной мере необходимый объём работ. Помимо этого анализ поможет учесть преимущества и недостатки существующих решений.

Система «АвтоПеревозки» — профессиональная компьютерная программа учета и управления работой парка собственной техники.

Основной функционал программы «АвтоПеревозки»:

- учет автотранспортной техники собственного парка, документов, связанных с учетом;
- печать и формирование путевых листов;
- потоковое формирование и печать сразу нескольких путевых листов на основе предыдущих данных;
- учет выполненных ремонтных работ;
- учет ремонтов и проведенных технических обслуживаний;
- планирование технического обслуживания;
- учет горюче-смазочных материалов и норм расходов топлива;
- учет работы водителей;
- учет работы шин и аккумуляторов;
- учет контрагентов и работы с ними;
- формирование и выдача различных сводок и отчетов;
- складской учет;

Стоимость данной программы, без внедрения 13000 российских рублей. Программа так же имеет расширенный функционал, направленный на ведение бухгалтерского и кадрового учёта.

#### 1С:Предприятие 8. Управление Автотранспортом. Стандарт

Программа "1С:Управление автотранспортом редакция Стандарт" предназначена для автоматизации управленческого и оперативного учета в автотранспортных предприятиях и организациях, а также в автотранспортных подразделениях торговых, производственных и прочих предприятиях, использующих автотранспорт для собственных нужд. Решение является



самостоятельным продуктом, разработанным на платформе 1С:Предприятие 8.3, не требующим приобретения дополнительных продуктов.

Программа состоит из следующих подсистем:

- Подсистема работа транспортных средств;
- Подсистема производственно-технический отдел. ;
- Подсистема учета горюче-смазочных материалов;
- Подсистема учета ремонтов;
- Подсистема складского учета;
- Подсистема взаиморасчетов;
- Подсистема учета работы водителей;
- Подсистема учета затрат.

Программа превосходит по функционалу АвтоПеревозки 4. Но ее стоимость намного выше, и равна 59.700 рублей без внедрения.

Система автоматизации транспортно-экспедиционной компании «1С Форес: Учет заказов» на платформе 1С 8.2 конфигурации «Управление торговлей» 10.3. Данная конфигурация сама по себе обладает богатыми функциональными возможностями. Кроме того, в нее были внесены изменения,отражающие специфику деятельности автотранспортных предприятий: реализован учет транспорта, транспортных расходов, выполненных заказов и многое другое.

Возможности программы:

- когда заказ принимается от перевозчика, то указывается информация о транспорте, водителе, маршруте, также из формы можно

подобрать соответствующую заявку заказчика и послать уведомление клиенту по электронной почте о выполнении заявки;

- менеджер имеет возможность контролировать выполнение заказа: прибытие машины, время разгрузки, загрузки, сведения о поломках, данные о товарно-транспортной накладной и ряд других показателей, оформляется документом «Выполнение».

- документ «Выполнение заказов заказчиков» и «Выполнение заказов перевозчиков» позволяет обобщать информацию по выполненным заказам и формировать финансовые документы (акт, счет, счет-фактуру), а также реестр выполненных заказов;

- отдельными документами оформляются заключение договоров с заказчиками, арендодателем транспортного средства и перевозчиками с указанием реквизитов всех нужных документов;

- печатается подробная карточка партнёра;

- перевозки могут осуществляться как собственным транспортом, так и арендованным с экипажем, и транспортом сторонних перевозчиков;

- отдельными документами формируются цены на городские перевозки, на междугородние перевозки;

- для каждого заказчика можно утверждать свои цены на каждый маршрут;

- заработная плата менеджеров считается отдельным документом «Расчёт заработной платы менеджера»;

- в системе реализован ряд и других решений.

Стоимость программного обеспечения 45000 российских рублей.

В ходе анализа систем аналогов можно сделать вывод, что системы обладают огромной мощностью обработки информации и широчайшими возможностями. В связи с этим стоимость и поддержка данных систем обходится очень дорого. Также люди, которые будут работать с данными системами должны проходить специальное обучение, а так как развитие систем не стоит на месте, то обучение будет многократным и периодическим, что несет за собой дополнительные финансовые траты. Сравнение программных продуктов представлено в таблице 1.

Таблица 1 — Сравнение программных продуктов

Название программного продукта	Функциональность	Сложность эксплуатации	Сложность сопровождения	Стоимость
Автоперевозки 4	Средняя	Средняя	Высокая	Средняя
1С Предприятие 8.3 Управление автотранспортом Стандарт	Высокая	Высокая	Средняя	Высокая
1С Форес: Учёт Заказов	Высокая	Высокая	Высокая	Высокая
Разрабатываемая система	Низкая	Низкая	Низкая	Низкая

Подводя итог, можно сказать, разрабатываемая информационная система гораздо дешевле, а узкая направленность системы облегчит её сопровождение и эксплуатацию. Помимо этого она увеличит скорость обучения сотрудников.

## 1.5 Обзор методов проектирования и разработки информационных систем

Методы проектирования информационных систем обычно относят к одному из трех видов – структурному, объектному и смешанному.

Суть структурного подхода заключается в том, что в основе структурного подхода положен принцип функциональной декомпозиции, при которой структура системы описывается в терминах иерархии ее функций и передачи информации между отдельными функциональными элементами [7].

Система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система охраняет целостное представление, в котором все составляющие компоненты взаимосвязаны. При разработке системы «снизу-вверх» от отдельных задач ко всей системе целостность теряется, возникают проблемы при информационной стыковке отдельных компонентов [8].

При объектно-ориентированном подходе программа представляет собой описание объектов, их свойств (или атрибутов), совокупностей (или классов), отношений между ними, способов их взаимодействия и операций над объектами (или методов) [6].

Преимуществом данного подхода будет являться концептуальная близость к предметной области произвольной структуры и назначения. Механизм наследования атрибутов и методов позволяет строить производные понятия на основе базовых и таким образом создавать модель сколь угодно сложной предметной области с заданными свойствами.

Еще одним теоретически интересным и практически важным свойством объектно-ориентированного подхода является поддержка механизма обработки событий, которые изменяют атрибуты объектов и моделируют их взаимодействие в предметной области.

Перемещаясь по иерархии классов от общих понятий предметной области к более конкретным и наоборот, программист получает возможность изменять степень абстрактности или конкретности взгляда на моделируемый реальный мир.

Смешанный подход, позволяет применять одновременно несколько подходов к проектированию информационной системы. Проведя анализ и выделив сильные и слабые стороны структурного и объектно-ориентированного подходов, можно использовать их в комбинации, используя сильные стороны каждого из них. Структурный подход применяется на этапе системного анализа, который является первым этапом жизненного цикла информационной системы, так как именно на этом этапе изучается деятельность и строится модель деятельности. В дальнейшем, на этапе проектирования информационной системы, второй этап жизненного цикла информационной системы, происходит применение объектно-ориентированного анализа для построения модели информационной системы[7].

### **1.5 Инструментальные средства проектирования ИС**

В качестве средств проектирования ИС транспортной компании:

- для построения организационной модели, моделей процессов/управления, диаграмм UML – Draw.io;
- для реализации базы данных — SQLite3.
- Для реализации клиентской части Python и библиотека Tkinter.

Выбранные средства позволяют с успехом выполнить необходимую работу, одновременно являясь при этом удобными и эффективными в использовании.

## 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

### 2.1 Разработка диаграммы вариантов использования

Для описания поведения и определения функциональных требований разрабатываемой информационной системы существует диаграмма вариантов использования, которая состоит из актёров (действующих лиц), которые взаимодействуют с системой по средствам вариантов использования.

На рисунке 7 представлена диаграмма вариантов использования для информационной системы транспортной компании.

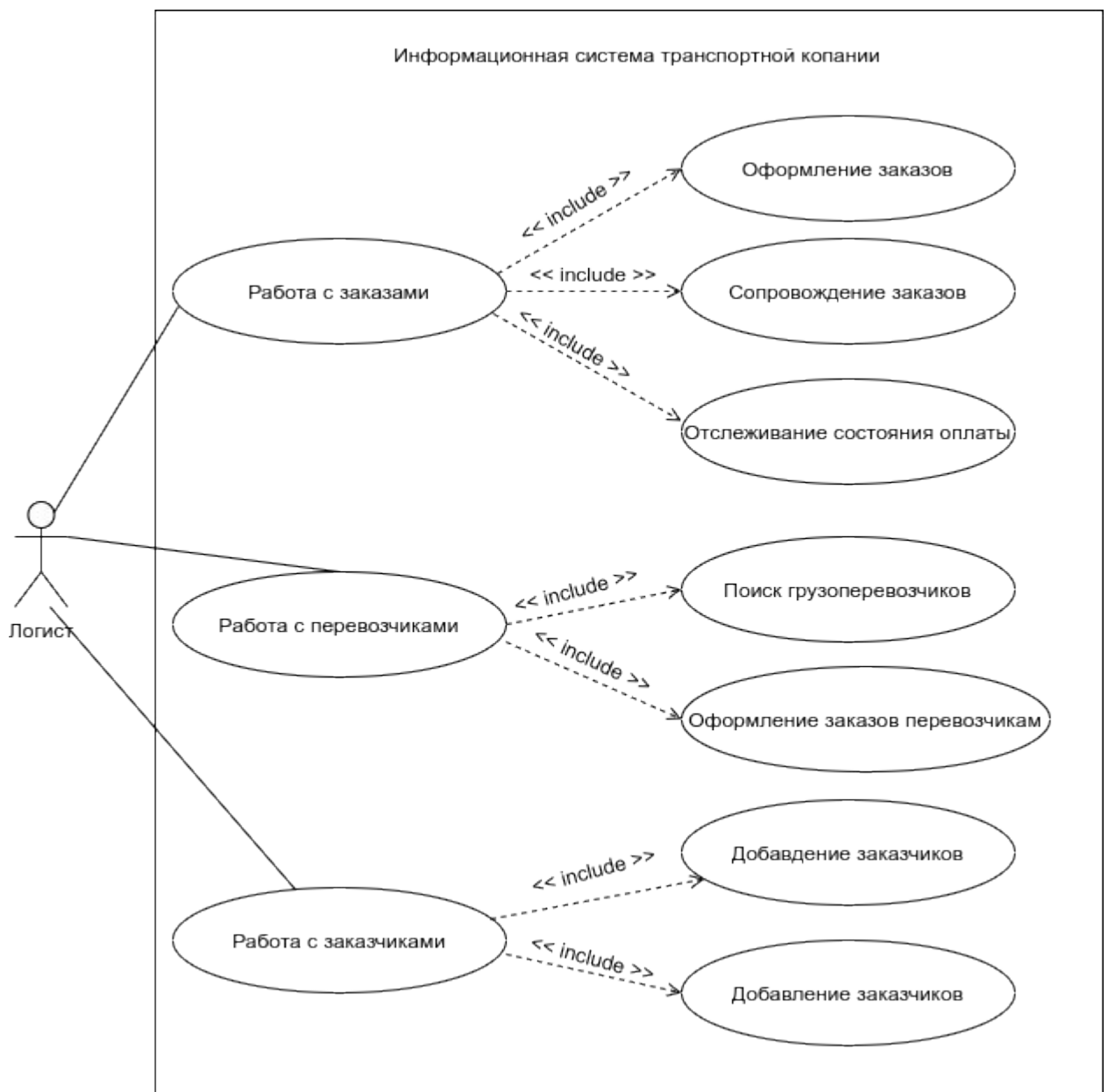


Рисунок 7 — Диаграмма вариантов использования

Описание действующих лиц представлено в таблице 2, а краткое описание вариантов использования представлено в таблице 3.

Таблица 2 — Описание действующих лиц

Название	Профиль, подготовка и навыки
Логист	Нерегулярный пользователь, вероятно, что ранее уже работал с графическим интерфейсом, может быть знаком с некоторыми функциями программного обеспечения.

Таблица 3 — Описание вариантов использования.

Действующее лицо	Цель	Краткое описание
Логист	Работа с заказами	Оформление заказов. Отслеживание состояния заказов. Отслеживание оплаты по заказам.
Логист	Работа с перевозками	Поиск перевозчиков, отвечающих требованиям заказов.
Логист	Работа с заказчиками	Построение списка заказчиков, работающих с компанией, и получение данных о них.

При выполнении служебных обязанностей логист работает в трех основных направлениях:

- работа с заказами;
- работа с перевозчиками;
- работа с заказчиками.

Работа с заказами включает в себя оформление заявок по полученным заказам, отслеживание состояния заказа и оплаты.

Работа с перевозчиками заключается в том, чтобы найти отвечающего заявленным требованиям в заявке перевозчика.

Работа с заказчиками состоит из добавления данных о новых заказчиках, ранее неизвестных компании, а также получение информации о уже добавленных клиентах.

Диаграмма позволяет конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

## 2.2 Разработка диаграммы классов

Диаграмма классов является ключевым элементом в объектно-ориентированном моделировании. Является одной из форм статического описания системы с точки зрения её проектирования, показывая её структуру. Диаграмма классов не отображает динамическое поведение объектов, изображённых на ней классов. На диаграммах классов показываются классы, интерфейсы и отношения между ними.

Для проектируемой системы выделены следующие классы:

- MainMenu (главное меню);
- Calendar (календарь);
- DriverList (список перевозчиков);
- AddDriver (добавление перевозчика);
- ApplicationList (список загрузок);
- AddApplication (добавление загрузки);
- ClientList (список заказчиков);
- AddClient (добавление заказчика);
- DriverUpdate (обновление перевозчика);
- ClientUpdate (обновление заказчика);
- ApplicationUpdate (обновление загрузки);
- Payment (оплаты).

Диаграмма классов представлена на рисунке 8.



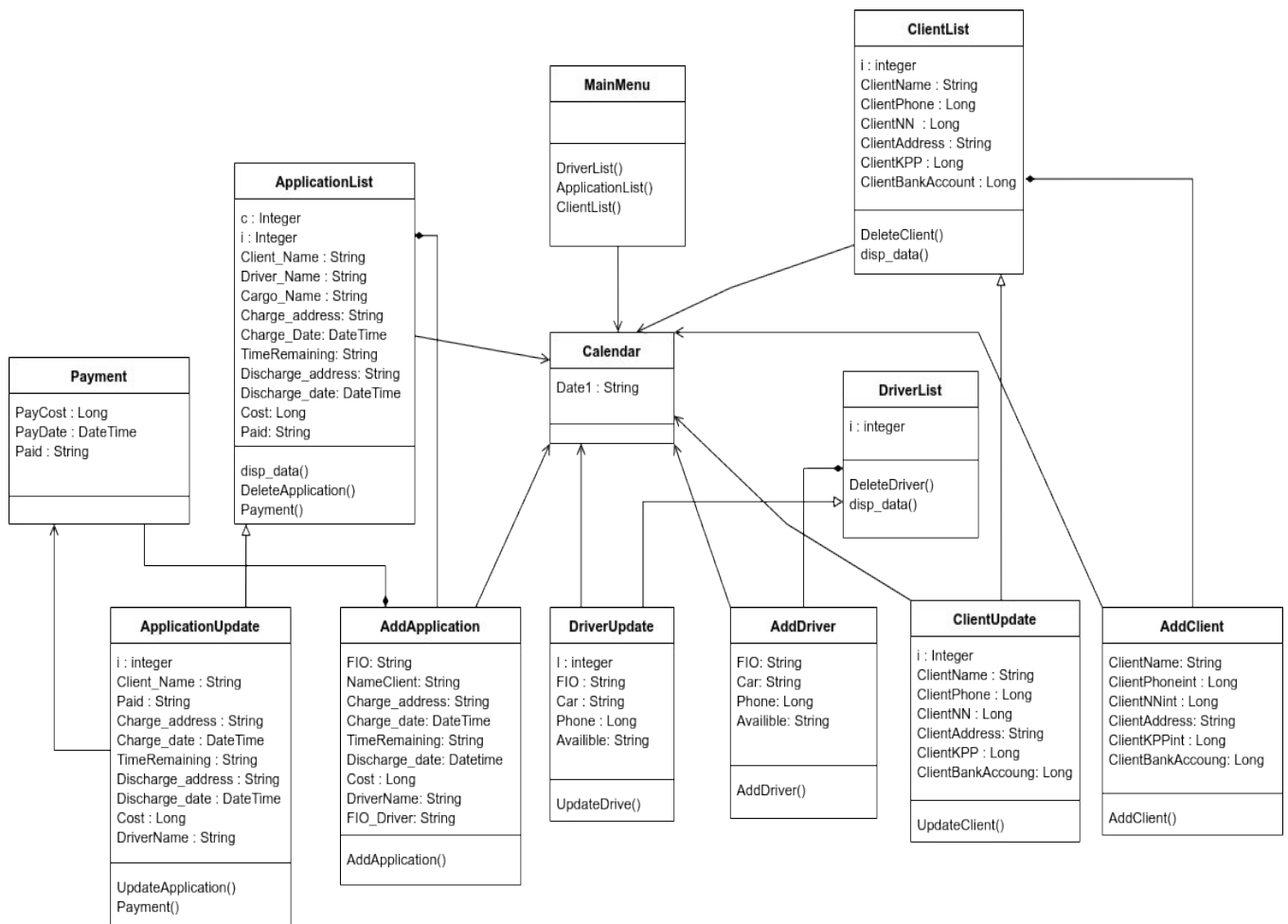


Рисунок 8 — Диаграмма классов

Между классами установлены следующие отношения (связи):

- Calendar связан со всеми классами отношением ассоциации, так как объекты всех классов осуществляют доступ к объектам класса Calendar;
- DriverList связан с DriverUpdate отношением обобщения (наследования), так как обновление перевозчика является потомком списка перевозчиков и включает в себя большую часть свойств перевозчика из списка
- 35
- ClientList связан с ClientUpdate отношением обобщения(наследования), так как обновление заказчика является потомком списка заказчиков и включает в себя большую часть свойств заказчика из списка;

- ApplicationList связан с ApplicationUpdate отношением обобщения (наследования), так как обновление загрузки является потомком списка загрузок и включает в себя большую часть свойств загрузки из списка;
- AddDriver связан с DriverList отношением композиции, так как список перевозчиков не может существовать без добавления перевозчиков;
- AddClient связан с ClientList отношением композиции, так как список заказчиков не может существовать без добавления заказчиков;
- AddApplication связан с ApplicationList отношением композиции, так как список загрузок не может существовать без добавления загрузок.

### 2.3 Разработка диаграммы деятельности

Диаграмма деятельности показывает разложение деятельности на её составные части. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий, которые соединены между собой потоками, которые идут от выходов одного узла к входам другого.

Диаграмма деятельности разработана для прецедента оформления заказов.

На диаграмме отражены следующие действия:

1. Ввод данных;
2. Происходит разветвление на получение данных о грузоперевозчике и получение данных о заказчике и грузе;
3. Происходит слияние в заполнение заявки.
4. Результатом деятельности будет являться отправка заявки заказчику.

Разработанная диаграмма деятельности представлена на рисунке 9.

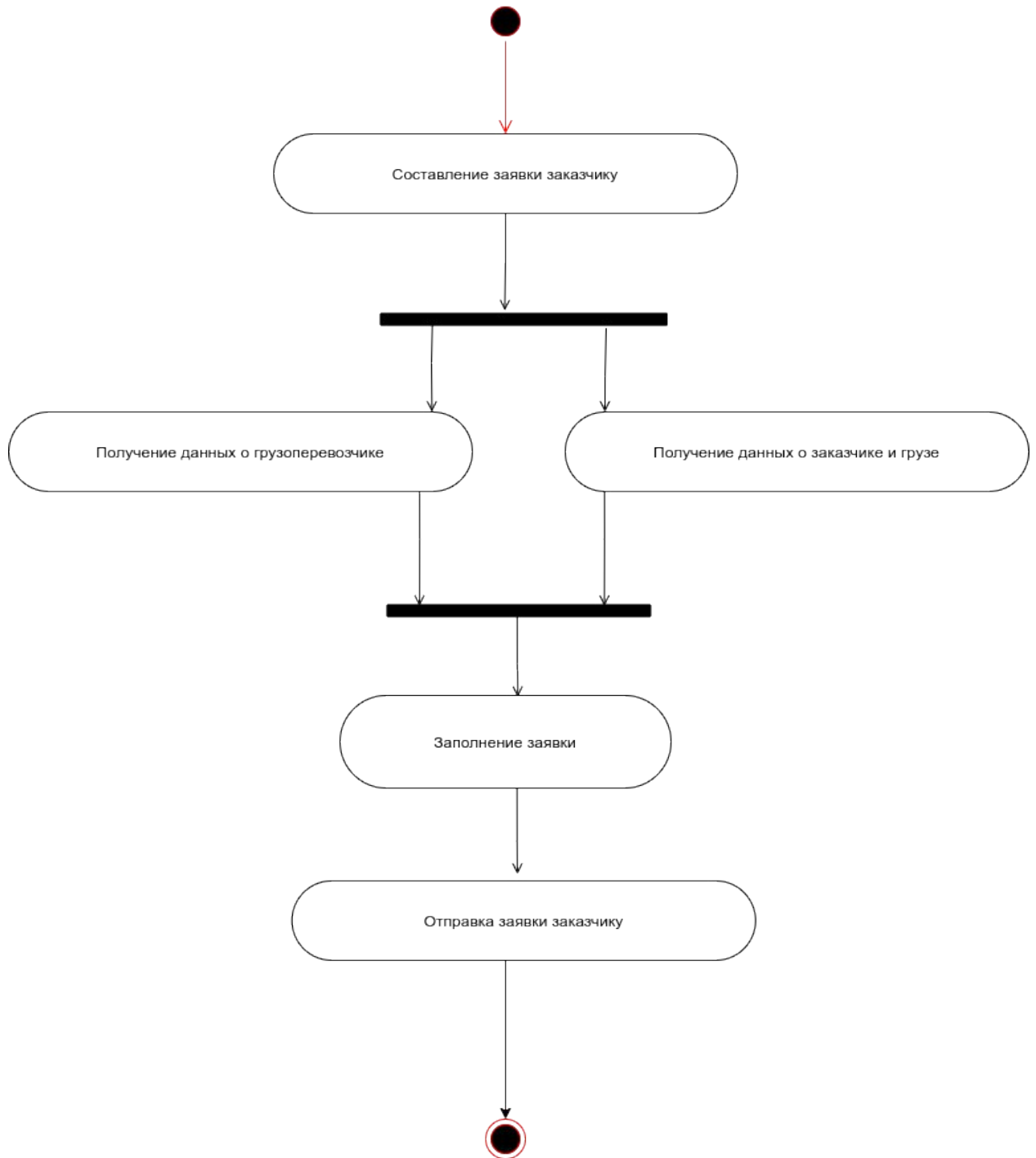


Рисунок 9 — Разработанная диаграмма деятельности

## 2.4 Разработка диаграммы последовательности

Диаграмма последовательности — диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл, какого-либо определённого объекта и взаимодействие актёров одной

информационной системы в рамках определённого прецедента (отправка запросов и получение ответов).

В данном случае диаграмма последовательности была разработана для прецедента «Добавление нового перевозчика». На данной диаграмме актёром является логист, который взаимодействует с объектами системы, а именно формой заполнения данных — опраывая и получая сообщения.

Объект, именуемый «Логист» отправляет синхронное сообщение «Форме заполнения данных перевозчика», заполняя форму данными, затем уж «Модуль работы с данными» совершает проверку корректности введённых им данных. Если данные являются корректными, то происходит их запись а базу данных, в ином случае, объект под названием «Логист» получает сообщение о некорректности введённым им данных. Объекты «Формы заполнения данных» и «Модуль работы с данными» существуют только в моменты заполнения формы данными и попытки записи данных в базу, в то время как логист и база данных существуют на всём протяжении.

Диаграмма последовательности представлена на рисунке 10.

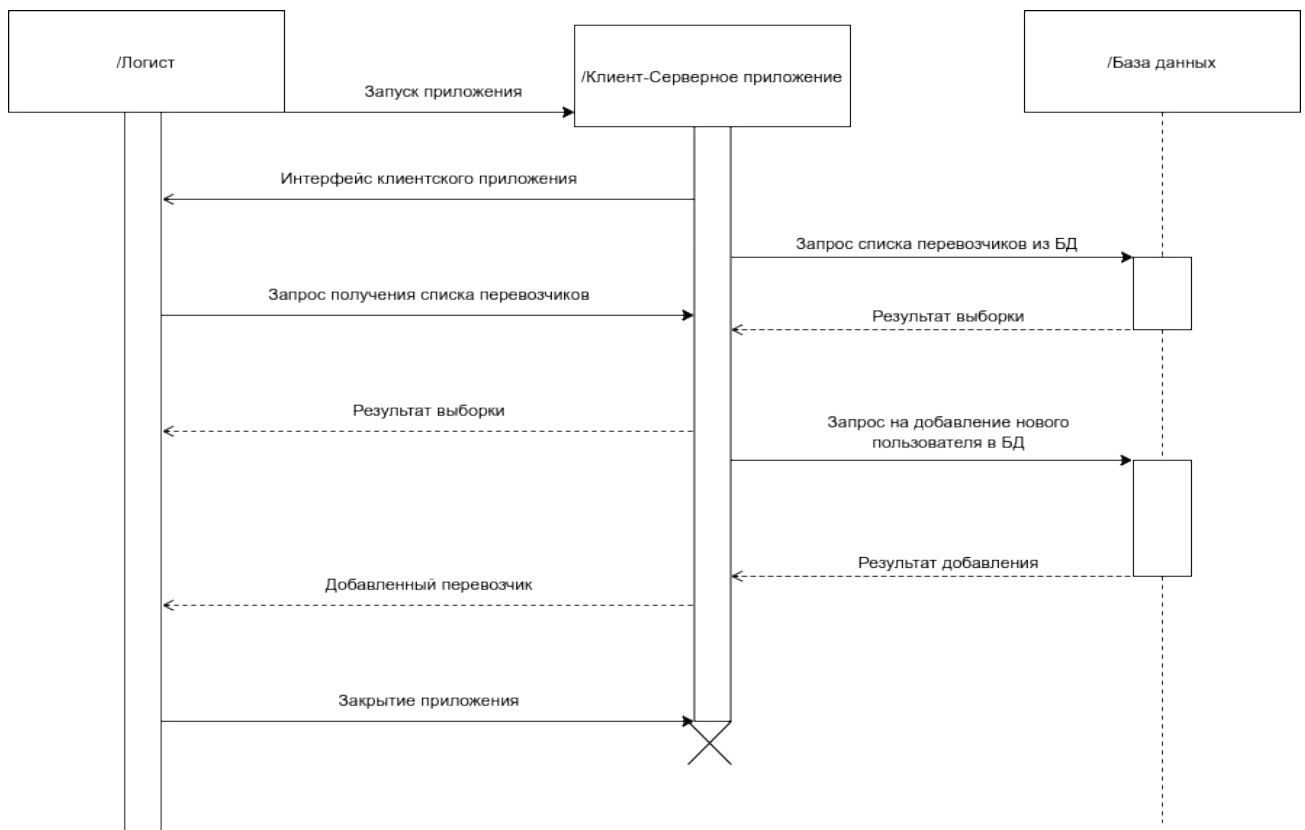


Рисунок 10 — Диаграмма последовательности

## 2.5 Реализация методов

### 2.5.1 Проектирование пользовательского интерфейса

Пользовательский интерфейс – это совокупность программных и аппаратных средств, которые обеспечивают взаимодействие пользователя с компьютером. Основу такого взаимодействия составляют диалоги. Под диалогом в данном случае понимают регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи. Каждый диалог состоит из отдельных процессов ввода/вывода, которые физически обеспечивают связь пользователя и компьютера. Обмен информацией осуществляется передачей сообщения.

В рамках курсовой работы предполагается частичная разработка программного обеспечения. Будет спроектировано окно «Перевозчики» и его дочерние окна для взаимодействия с программной.

Окно программы «Перевозчики» представлено на рисунке 11.

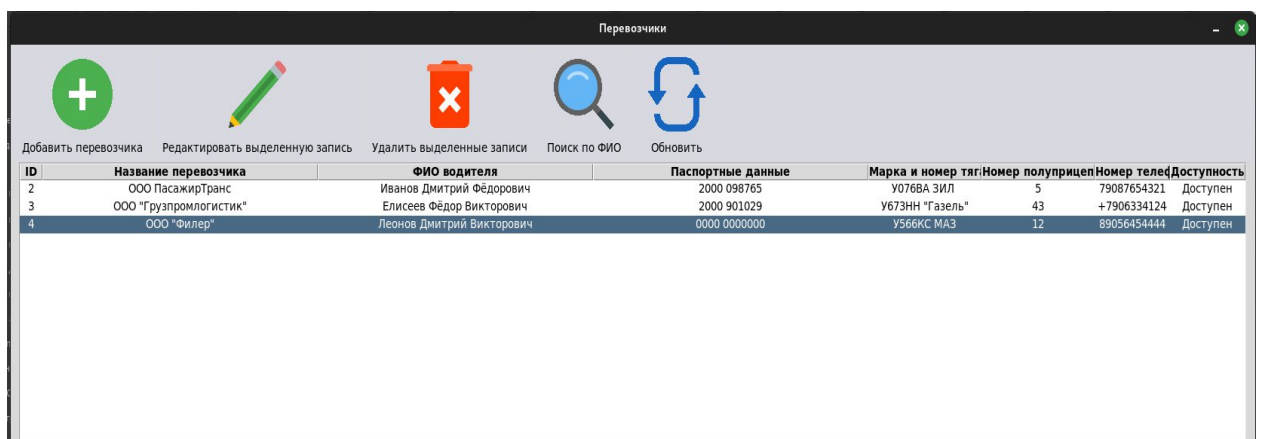


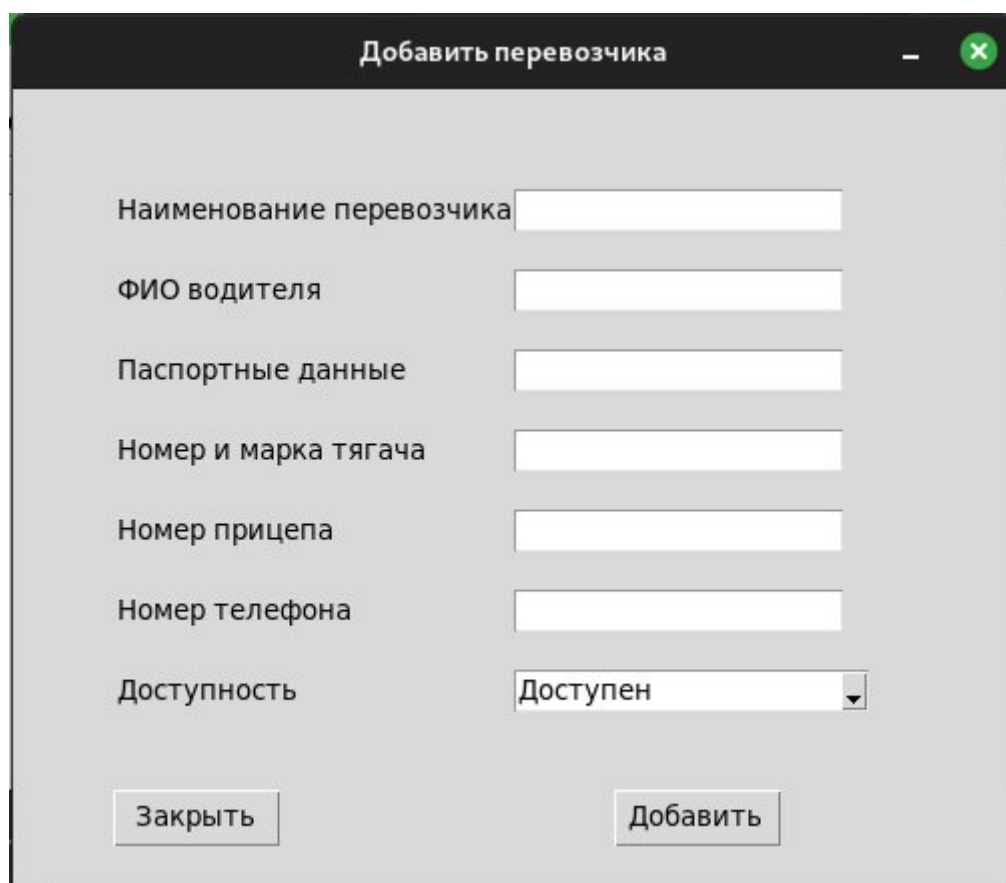
Рисунок 11 — окно программы «Перевозчики»

Форма перевозчиков состоит из трёх основных частей:

- Панель инструментов (Toolbar);
- форма отображения перевозчиков;

- пять кнопок, которые осуществляют следующий функционал: кнопка «Добавить перевозчика» открывает дочернее окно в котором реализована форма для добавления нового перевозчика, кнопка «Редактировать выделенную запись» при выделении нужной записи левой кнопкой мыши и последующим нажатием на эту кнопку открывает форму в которой можно редактировать запись об уже добавленном перевозчике, важно понимать, что нужно редактировать все данные формы, потому что очищается вся запись о перевозчике, кнопка «Удалить выделенную запись» удаляет выделенную запись в форме отображения перевозчиков, кнопка «Поиск по ФИО» позволяет найти нужного перевозчика, важно понимать, что поиск чувствителен к регистру, кнопка «Обновить» покажет все записи в таблице, её необходимо нажимать после выполненного поиска.

Форма добавления нового перевозчика представлена на рисунке 12.



Добавить перевозчика

Наименование перевозчика

ФИО водителя

Паспортные данные

Номер и марка тягача

Номер прицепа

Номер телефона

Доступность

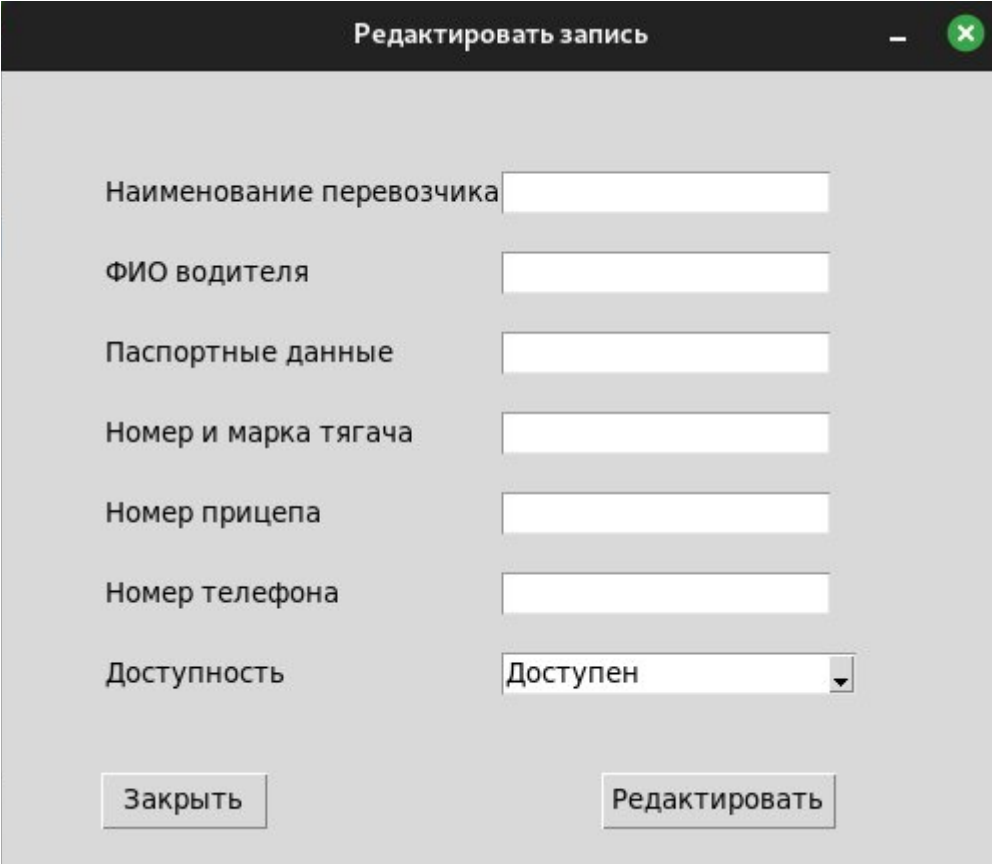
Заккрыть

Рисунок 12 — Форма добавления нового перевозчика

Форма добавления нового перевозчика представляет собой составную часть из нескольких полей ввода:

- поля «Наименование перевозчика», «ФИО водителя», «Паспортные данные водителя», «Марка и номер тягача», «Номер полуприцепа (прицепа)», «Номер телефона водителя» и «Доступность» необходимы для заполнения данных о перевозчике для дальнейшего внесения его в базу данных;
- кнопка «Добавить» необходима для внесения данных о перевозчике в базу данных;
- кнопка «Закрыть» закрывает окно формы добавления нового перевозчика.

Форма «Редактировать запись» представлена на рисунке 13.



The image shows a software window titled "Редактировать запись" (Edit Record). The window has a dark title bar with standard minimize, maximize, and close buttons. The main area is light gray and contains several input fields arranged vertically. Each field is preceded by a label. The labels are: "Наименование перевозчика" (Carrier Name), "ФИО водителя" (Driver's Full Name), "Паспортные данные" (Passport Data), "Номер и марка тягача" (Tractor Number and Brand), "Номер прицепа" (Trailer Number), "Номер телефона" (Phone Number), and "Доступность" (Availability). The first six labels are followed by white rectangular text input boxes. The "Доступность" label is followed by a dropdown menu currently showing "Доступен" (Available). At the bottom of the form, there are two buttons: "Закрыть" (Close) on the left and "Редактировать" (Edit) on the right.

Рисунок 13 — Форма «Редактировать запись»

Форма «Редактировать запись» представляет собой составную часть из нескольких полей ввода и меню:

- поля «Наименование перевозчика», «ФИО водителя», «Паспортные данные водителя», «Марка и номер тягача», «Номер полуприцепа (прицепа)», «Номер телефона водителя» и «Доступность» необходимы для изменения данных о перевозчике для дальнейшего внесения его в базу данных;
- кнопка «Редактировать» необходима для внесения обновленных данных о перевозчике в базу данных;
- кнопка «Закрыть» закрывает окно формы добавления нового перевозчика.

Форма поиска представлена на рисунке 14.

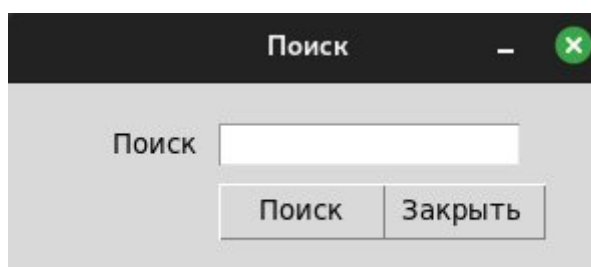


Рисунок 14 — Форма поиска

Форма поиска представляет собой составную часть из одного поля ввода и двух кнопок:

- кнопка «Поиск» реализует поиск по полю «Фамилия имя и отчество»;
- кнопка «Закрыть» закрывает форму.

### 2.5.2 Разработка структуры приложения

После ознакомления с основными на данный момент типами баз данных, была выбрана встраиваемая система управления базами данных SQLite, которая по умолчанию присутствует в библиотеке Python 3.



Слово «Встраиваемый» означает, что SQLite не использует парадигму клиент-сервер. Это значит, что движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется и движок становится составной частью программы.

Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором выполняется программа. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

SQL — это структурированный язык запросов, созданный для того, чтобы получать из базы данных необходимую информацию. SQL в чистом (базовом) виде является информационно-логическим языком, а не языком программирования. Однако стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений, с учетом которых язык уже может рассматриваться в качестве языка программирования. Первый вариант языка SQL был разработан и частично реализован в рамках проекта экспериментальной реляционной СУБД SystemR. SQL предоставляет пользователю достаточно простой и понятный механизм доступа к данным, не связанный с конструированием алгоритма и его описанием на языке программирования высокого уровня. Так, вместо указания того, как необходимо действовать, пользователь с помощью операторов SQL объясняет СУБД, что нужно сделать. Далее СУБД сама анализирует текст запроса и определяет, как именно его выполнять [9, 23].

Tkinter встроен в стандартную библиотеку языка Python. У Tkinter есть несколько преимуществ:

1. Кроссплатформенность - один и тот же код можно использовать на операционных системах семейства Windows, macOS или Linux.
2. Визуальные элементы отображаются через собственные элементы текущей операционной системы. Таким образом, приложения, созданные с помощью Tkinter, выглядят так, как будто их разрабатывали для той платформы, на которой они работают.

Tkinter является простым по сравнению с другими библиотеками. Это отличный инструмент для создания GUI приложений в Python, особенно, когда большую роль играет функциональность и кроссплатформенная скорость.

## ЗАКЛЮЧЕНИЕ

В результате проделанной работы были достигнуты следующие цели:

- был произведён анализ предметной области и сформулированы требования к программе.
- было подробно описано техническое задание к проектируемой системе включающее требования к программе как функционального так и технического характера.
- с помощью языка UML были сформированы функциональные требования к системе в виде диаграммы вариантов использования.
- разработанная диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.
- разработаны диаграммы классов, деятельности и последовательности.
- с помощью языка программирования Python версии 3 и его встроенных библиотек Tkinter и SQLite была частично реализована база данных и клиентское приложение информационной системы транспортной компании.

При полной реализации приложения применение данной системы позволит усовершенствовать и ускорить работу организаций, которые осуществляют грузоперевозки.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Иванова, О. Г. Методы и средства проектирования информационных систем и технологий. Основы UML : учебное пособие / О. Г. Иванова, Ю. Ю. Громов. — Тамбов : Тамбовский государственный технический университет, ЭБС АСВ, 2020. — 80 с. — ISBN 978-5-8265-2308-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: <https://www.iprbookshop.ru/115768.html> (дата обращения: 15.02.2022). — Режим доступа: для авторизир. пользователей
2. Галиаскаров, Э. Г. Анализ и проектирование систем с использованием UML : учебное пособие для вузов / Э. Г. Галиаскаров, А. С. Воробьев. — Москва : Издательство Юрайт, 2022. — 125 с. — (Высшее образование). — ISBN 978-5-534-14903-6. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/497207> (дата обращения: 15.02.2022).
3. Бабич, А. В. Введение в UML : учебное пособие / А. В. Бабич. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 198 с. — ISBN 978-5-4497-0544-0. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: <https://www.iprbookshop.ru/94847.html> (дата обращения: 15.02.2022). — Режим доступа: для авторизир. пользователей
4. Давыдовский, М. А. Проектирование программной системы в UML Designer : учебное пособие / М. А. Давыдовский, М. Н. Никольская. — Москва : Российский университет транспорта (МИИТ), 2019. — 130 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: <https://www.iprbookshop.ru/116069.html> (дата обращения: 17.02.2022). — Режим доступа: для авторизир. пользователей

6. Объектно-ориентированный подход к программированию [Электронный ресурс]. URL: <http://www.intuit.ru/studies/courses/50/50/lecture/1492> (дата обращения: 19.02.2022).

7. Методы структурного анализа и проектирования ПО [Электронный ресурс]. URL: [https://studbooks.net/47730/informatika/metody\\_strukturnogo\\_analiza\\_i\\_proektirovaniya](https://studbooks.net/47730/informatika/metody_strukturnogo_analiza_i_proektirovaniya)

8. Структурный подход к проектированию ИС [Электронный ресурс]. URL: [http://citforum.ru/database/case/glava2\\_1.shtml](http://citforum.ru/database/case/glava2_1.shtml) (дата обращения: 19.02.2022).

9. Петкович, Д. Microsoft SQL Server 2018. Руководство для начинающих / Д. Петкович Пер. с англ. – СПб.: БХВ-Петербург, 2020. – 817 с.

## Листинг программы

```
import tkinter as tk

from tkinter import ttk

import sqlite3


##Класс главного окна

class Main(tk.Frame):

    def __init__(self, root):

        super().__init__(root)

        self.init_main()

        self.db = db

        # Для отображения данных:

        self.view_records()


    def init_main(self):

        toolbar = tk.Frame(bg='#d7d8e0', bd=2)

        toolbar.pack(side=tk.TOP, fill=tk.X)

        self.add_img = tk.PhotoImage(file="add.gif")
```

```
btn_open_dialog = tk.Button(toolbar, text='Добавить перевозчика',  
command=self.open_dialog, bg='#d7d8e0', bd=0,
```

```
compound=tk.TOP, image=self.add_img)
```

```
btn_open_dialog.pack(side=tk.LEFT)
```

```
self.update_img = tk.PhotoImage(file='update.gif')
```

```
btn_edit_dialog = tk.Button(toolbar, text="Редактировать выделенную  
запись", bg='#d7d8e0', bd=0,
```

```
image=self.update_img,
```

```
compound=tk.TOP, command=self.open_update_dialog)
```

```
btn_edit_dialog.pack(side=tk.LEFT)
```

```
self.delete_img = tk.PhotoImage(file='delete.gif')
```

```
btn_delete = tk.Button(toolbar, text='Удалить выделенные записи',  
bg='#d7d8e0', bd=0, image=self.delete_img,
```

```
compound=tk.TOP, command=self.delete_records)
```

```
btn_delete.pack(side=tk.LEFT)
```

```
self.search_img = tk.PhotoImage(file='search.gif')
```

```
btn_search = tk.Button(toolbar, text='Поиск по ФИО', bg='#d7d8e0',  
bd=0, image=self.search_img,
```

```
compound=tk.TOP, command=self.open_search_dialog)
```

```
btn_search.pack(side=tk.LEFT)
```

```

self.refresh_img = tk.PhotoImage(file='refresh.gif')

btn_refresh = tk.Button(toolbar, text='Обновить', bg='#d7d8e0', bd=0,
image=self.refresh_img,

                        compound=tk.TOP, command=self.view_records)

btn_refresh.pack(side=tk.LEFT)


self.tree = ttk.Treeview(self, columns=('ID_Driver', 'DriverName',
'FIO_Driver', 'PassportData',

                        'Car', 'Trailer', 'Driver_Phone', 'Available'),

                        height=15, show='headings')

self.tree.column("ID_Driver", width=30, anchor=tk.CENTER)

self.tree.column("DriverName", width=365, anchor=tk.CENTER)

self.tree.column("FIO_Driver", width=365, anchor=tk.CENTER)

self.tree.column("PassportData", width=365, anchor=tk.CENTER)

self.tree.column("Car", width=150, anchor=tk.CENTER)

self.tree.column("Trailer", width=150, anchor=tk.CENTER)

self.tree.column("Driver_Phone", width=100, anchor=tk.CENTER)

self.tree.column("Available", width=100, anchor=tk.CENTER)


self.tree.heading("ID_Driver", text='ID')

self.tree.heading("DriverName", text='Название перевозчика')

```



```

self.tree.heading("FIO_Driver", text='ФИО водителя')

self.tree.heading("PassportData", text='Паспортные данные')

self.tree.heading("Car", text='Марка и номер тягача')

self.tree.heading("Trailer", text='Номер полуприцепа')

self.tree.heading("Driver_Phone", text='Номер телефона')

self.tree.heading("Available", text='Доступность')


self.tree.pack()

```

# Функция записи данных

```

def records(self, drivename, fio, passdata, car, trailer, driver_phone,
available):

```

```

    self.db.insert_data(drivename, fio, passdata, car, trailer, driver_phone,
available)

```

# Для отображения данных:

```

self.view_records()

```

```

def update_record(self, DriverName, FIO_Driver, PassportData, Car,
Trailer, Driver_Phone, Available):

```

```

    self.db.c.execute('UPDATE carriers SET DriverName=?, FIO_Driver=?,
PassportData=?, Car=?, Trailer=?, '

```

```

'Driver_Phone=?, Available=? WHERE ID=?',

```

[DriverName, FIO\_Driver, PassportData, Car, Trailer,  
Driver\_Phone, Available,

self.tree.set(self.tree.selection()[0], '#1'))

self.db.conn.commit() # сохранить изменения

self.view\_records() # отобразить изменения

def view\_records(self):

self.db.c.execute("SELECT \* FROM carriers") # Отображение в  
тревью

[self.tree.delete(i) for i in self.tree.get\_children()] # Обновление данных

[self.tree.insert("", 'end', values=row) for row in self.db.c.fetchall()]

def delete\_records(self):

for selection\_item in self.tree.selection(): # Нужен цикл для удаления  
нескольких полей

self.db.c.execute("DELETE FROM carriers WHERE id=?",  
(self.tree.set(selection\_item, '#1'),))

self.db.conn.commit()

self.view\_records()

def search\_records(self, FIO\_Driver):

```

FIO_Driver = ('%' + FIO_Driver + '%',)

self.db.c.execute("SELECT * FROM carriers WHERE FIO_Driver
LIKE ?",

                    FIO_Driver) # Обращаемся к базе данных и формируем
SQL запрос

##Отображаем строки в виджете Treeview

#1. Очистим содержимое виджета путём использования генератора
списка

#2. В цикле будем получать строки из метода treeview с помощью
метода get.children

#3. После удалять, применяя метод Delete

[self.tree.delete(i) for i in self.tree.get_children()]

#Отображаем результаты поиска:

[self.tree.insert("", 'end', values=row) for row in self.db.c.fetchall()]

def open_dialog(self):

    Child()

def open_update_dialog(self):

    Update()

def open_search_dialog(self):

    Search()

```

```

class Child(tk.Toplevel):

    def __init__(self):

        super().__init__(root)

        self.init_child()

        self.view = app

    def init_child(self):

        self.title('Добавить перевозчика')

        self.geometry('500x400+400+300')

        self.resizable(False, False)

        # Виджет ввода данных:

        label_drivename = tk.Label(self, text="Наименование перевозчика")

        label_drivename.place(x=50, y=50)

        self.entry_drivename = ttk.Entry(self)

        self.entry_drivename.place(x=250, y=50)      # Наименование
перевозчика

        label_drivename = tk.Label(self, text="ФИО водителя")

```

```
label_drivername.place(x=50, y=90)

self.entry_fio = ttk.Entry(self)

self.entry_fio.place(x=250, y=90) # ФИО Водителя


label_drivername = tk.Label(self, text="Паспортные данные")

label_drivername.place(x=50, y=130)

self.entry_passdata = ttk.Entry(self)

self.entry_passdata.place(x=250, y=130) # Паспортные данные


label_drivername = tk.Label(self, text="Номер и марка тягача")

label_drivername.place(x=50, y=170)

self.entry_car = ttk.Entry(self)

self.entry_car.place(x=250, y=170) # Марка и номер тягача


label_drivername = tk.Label(self, text="Номер прицепа")

label_drivername.place(x=50, y=210)

self.entry_trailer = ttk.Entry(self)

self.entry_trailer.place(x=250, y=210) # Номер прицепа


label_drivername = tk.Label(self, text="Номер телефона")

label_drivername.place(x=50, y=250)

self.entry_driver_phone = ttk.Entry(self)
```

```

self.entry_driver_phone.place(x=250, y=250)

label_drivername = tk.Label(self, text='Доступность')

label_drivername.place(x=50, y=290)

self.combobox = ttk.Combobox(self, values=[u'Доступен', u'Не
доступен'])

self.combobox.current(0)

self.combobox.place(x=250, y=290) # Доступность экипажа

# Кнопка закрытия окна:

btn_cancel = ttk.Button(self, text='Закрыть', command=self.destroy)

btn_cancel.place(x=50, y=350)

# Кнопка "Добавить"

self.btn_ok = ttk.Button(self, text='Добавить')

self.btn_ok.place(x=300, y=350)

self.btn_ok.bind('<Button-1>', lambda event:
self.view.records(self.entry_drivername.get(),
self.entry_fio.get(),
self.entry_passdata.get(),
self.entry_car.get(),
self.entry_trailer.get(),

```

```
self.entry_driver_phone.get(),
self.combobox.get()))
```

```
##END##
```

```
self.grab_set()
```

```
self.focus_set()
```

```
# Создадим класс update, который будет наследоваться от класса
child
```

```
class Update(Child):
```

```
def __init__(self):
```

```
    super().__init__()
```

```
    self.init_edit() # Для отображения пользователю изменений.
Вызываем функцию из конструктора класса update.
```

```
    self.view = app
```

```
def init_edit(self):
```

```
    self.title('Редактировать запись')
```

```
    btn_edit = ttk.Button(self, text='Редактировать')
```

```
    btn_edit.place(x=300, y=350)
```

```

        btn_edit.bind('<Button-1>', lambda event:
self.view.update_record(self.entry_drivename.get(),
                        self.entry_fio.get(),
                        self.entry_passdata.get(),
                        self.entry_car.get(),
                        self.entry_trailer.get(),
                        self.entry_driver_phone.get(),
                        self.combobox.get()))

```

```

        self.btn_ok.destroy() # убрали кнопку "ОК", потому что заменили на
кнопку "Редактировать"

```

```

# Класс базы данных

```

```

class Search(tk.Toplevel):

```

```

    def __init__(self):

```

```

        super().__init__()

```

```

        self.__init_search()

```

```

        self.view = app

```

```

    def __init_search(self):

```

```

        self.title('Поиск')

```



```
self.geometry('300x100+400+300')
```

```
self.resizable(False, False)
```

```
label_search = tk.Label(self, text='Поиск')
```

```
label_search.place(x=50, y=20)
```

```
self.entry_search = ttk.Entry(self)
```

```
self.entry_search.place(x=105, y=20, width=150)
```

```
btn_cancel = ttk.Button(self, text='Закрыть', command=self.destroy)
```

```
btn_cancel.place(x=185, y=50)
```

```
btn_search = ttk.Button(self, text='Поиск')
```

```
btn_search.place(x=105, y=50)
```

```
btn_search.bind('<Button-1>', lambda event:
self.view.search_records(self.entry_search.get()))
```

```
#Чтобы после нажатия кнопки "Поиск" окно закрывалось:
```

```
btn_search.bind('<Button-1>', lambda event: self.destroy(), add='+')
```

```
class DB:
```

```
def __init__(self):
```

```

self.conn = sqlite3.connect('trucking.db')

self.c = self.conn.cursor()

self.c.execute(

    "CREATE TABLE IF NOT EXISTS carriers (id integer primary key,
DriverName text, FIO_Driver text,

    PassportData text, Car text, Trailer text, Driver_Phone text, Available
text)")

self.conn.commit()


def insert_data(self, drivename, fio, passdata, car, trailer, driver_phone,
available):

    self.c.execute(

        "INSERT INTO carriers(DriverName, FIO_Driver, PassportData, Car,
Trailer, Driver_Phone, Available)

        VALUES (?, ?, ?, ?, ?, ?, ?)",

        (drivename, fio, passdata, car, trailer, driver_phone, available))

    self.conn.commit()


if __name__ == "__main__":

    root = tk.Tk()

    # Для того чтобы обращаться к функциям класса db из класса main.
    Необходимо создать экземпляр класса db:

```

```
db = DB()
```

```
app = Main(root)
```

```
app.pack()
```

```
root.title("Перевозчики")
```

```
root.geometry("1650x450+300+200")
```

```
root.resizable(False, False)
```

```
root.mainloop()
```