Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Дисциплина: Технология проектирования программного обеспечения

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ АВТОЗАПРАВОЧНОЙ СТАНЦИИ

БГУИР КП 6-05-0612-03 029 ПЗ

Студент гр. 320601 Яковчик А. Ю.
Руководитель Боброва Т. С.

Минск 2025

РЕФЕРАТ

ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ АВТОЗАПРАВОЧНОЙ СТАНЦИИ: курсовой проект / А. Ю. Яковчик – Минск: БГУИР, 2025, – п.з. – 40 с., чертежей – 1 л. формата А1.

Курсовой проект на тему «Проектирование автоматизированной системы автозаправочной станции» разработан с целью оптимизации процессов обслуживания клиентов на автозаправочных станциях (АЗС) посредством автоматизации операций заправки, оплаты и управления запасами топлива. Система направлена на повышение эффективности работы АЗС путем автоматизации учета топлива, интеграции с платежными системами, оповещения персонала о необходимости пополнения запасов и обеспечения удобного взаимодействия с клиентами через пользовательский интерфейс.

Пояснительная записка к курсовому проекту состоит из введения, трех основных разделов, включающих анализ технического задания и разработку спецификации, проектирование системы с созданием соответствующих диаграмм, реализацию системы, а также заключения, списка использованных источников и приложения, содержащего листинг кода отдельных модулей системы.

Для разработки проекта использован объектно-ориентированный язык проектирования *UML*. С помощью *CASE*-средства *Rational Rose* созданы диаграммы, включая диаграммы классов, последовательности, состояний и активности, которые в совокупности представляют модель автоматизированной системы АЗС. В процессе выполнения курсовой работы составлен глоссарий терминов, относящихся к предметной области, и разработаны чертежи, иллюстрирующие структуру системы.

Проект представляет собой описание создания автоматизированной системы для автозаправочной станции. Смоделированная система, полученная в ходе курсового проектирования, может служить основой для реализации полноценного программного обеспечения для АЗС, упрощающего процессы заправки, учета и взаимодействия с клиентами. Кроме того, проект может использоваться в образовательных целях для изучения языка моделирования *UML* и служить примером проектирования аналогичных автоматизированных систем.

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет информационных технологий и управления

УТВЕРЖДАЮ
Заведующий кафедрой

_____
(подпись)
_____2025г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Яковчику Алексею Юрьевичу

1. Тема проекта Проектирование автоматизированной системы автозаправочной станции

2. Срок сдачи студентом законченного проекта ***6 июня 2025 г.***

3. Исходные данные к проекту: *Разработать программную моделиь автоматизированной системы управления автозаправочной станцией (АЗС) с использованием методов объектно-ориентированного проектирования. В рамках проекта требуется проанализировать предметную область, определить функциональные требования, разработать архитектуру системы и реализовать основные компоненты в виде UML-диаграмм. Система должна моделировать процессы заправки топлива, оплаты, взаимодействия с пользователями (гостями и зарегистрированными), управления оборудованием и формирования отчётности.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):

Введение

1. *Анализ технического задания на курсовое проектирование*
2. *Проектирование системы*
3. *Реализация системы*

Заключение

Список использованных источников

Приложение А (обязательное)

Ведомость курсового проекта

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1 Диаграмма вариантов использования

6. Консультант по проекту (с обозначением разделов проекта) *Боброва Т.С.*

7. Дата выдачи задания *25 февраля 2024 г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

раздел 1 к 15.03 – 15 %;

раздел 2 к 15.04 – 50 %;

раздел 3 к 15.05 – 15 %;

оформление пояснительной записки и графического материала к 18.05 – 20 %

*Защита курсового проекта с **20.05** по **06.06.2025**г.*

Руководитель *Т.С.* Боброва

(подпись)

Задание принял к исполнению *А. Ю. Яковчик*

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные тенденции развития автозаправочных станций (АЗС) требуют высокого уровня автоматизации процессов для обеспечения оперативности обслуживания клиентов, точности учета топлива и оптимизации работы персонала. Автоматизированная система автозаправочной станции играет ключевую роль в повышении эффективности работы АЗС, предоставляя инструменты для управления процессами заправки, оплаты, мониторинга запасов топлива и взаимодействия с клиентами. В этом контексте разработка автоматизированной системы для АЗС становится актуальной задачей, направленной на повышение конкурентоспособности станций и удовлетворенности клиентов.

Данный проект разрабатывался с использованием языка моделирования *UML* (*Unified Modeling Language*) и *CASE*-средства *Rational Rose*.

Язык *UML* – это графический язык моделирования общего назначения, предназначенный для спецификации, визуализации, проектирования и документирования всех артефактов, создаваемых при разработке программных систем. Он используется для проектирования систем и является унифицированным, общепризнанным языком объектно-ориентированного программирования. *UML* содержит стандартный набор диаграмм и нотаций самых разнообразных видов. *UML* принят на вооружение практически всеми крупнейшими компаниями – производителями программного обеспечения (*Microsoft, IBM, Hewlett-Packard, Oracle, Sybase* и др.).

UML, будучи объектно-ориентированным языком моделирования, обладает сходством с методами программирования на современных объектно-ориентированных языках, что делает его подходящим для описания результатов анализа и проектирования. Он позволяет представлять систему с различных точек зрения и освещать различные аспекты ее поведения. Диаграммы *UML* легко осваиваются благодаря своему простому синтаксису, что облегчает их восприятие даже новичками. Кроме того, *UML* предлагает возможность расширения через собственные стереотипы, текстовые и графические, что расширяет его область применения за пределы программной инженерии. *UML* получил широкое распространение и динамично развивается.

Rational Rose – это популярное *CASE*-средство, предоставляющее все необходимые инструменты для создания и анализа моделей *UML*, необходимых для решения задачи данного проекта. Оно включает все

основные виды диаграмм и позволяет автоматически генерировать код на основе разработанных моделей, что значительно ускоряет процесс разработки программного обеспечения, делая его идеальным выбором для реализации данного проекта.

В рамках проекта будет проведен анализ требований к системе, разработан набор диаграмм *UML*, включая диаграммы классов, последовательности, состояний и активности, необходимые для полноценного описания будущей системы.

Основная цель проекта – создать надежную и удобную автоматизированную систему для автозаправочной станции, которая будет отвечать потребностям как клиентов, так и персонала АЗС, обеспечивая высокую степень удовлетворенности пользователей и эффективность работы. Результатом проекта станет подробное описание архитектуры автоматизированной системы, представленное в форме *UML*-моделей, а также прототип системы, демонстрирующий её функциональность и возможности. Это позволит не только оценить эффективность предлагаемой архитектуры, но и предоставить основу для дальнейшего развития и улучшения системы. Также проект направлен на демонстрацию освоения *Rational Rose* и *UML*.

1 АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

1.1 Постановка задачи проектирования

Автозаправочные станции (АЗС) сталкиваются с необходимостью повышения эффективности обслуживания клиентов, точности учета топлива и автоматизации управления операциями. Для решения этих задач требуется разработка автоматизированной системы автозаправочной станции, которая позволит клиентам выбирать тип топлива, объем заправки и способ оплаты через удобный интерфейс, а также получать информацию о совершенных транзакциях и доступных услугах. Система обеспечит персоналу АЗС возможность мониторинга запасов топлива, автоматического формирования заказов на пополнение и управления процессами обслуживания. Цель системы – оптимизация работы АЗС, минимизация времени обслуживания клиентов и повышение точности учета ресурсов.

В системе предусмотрены функции для ввода клиентом данных, необходимых для выполнения заправки, включая выбор типа топлива, объема и способа оплаты (наличные, карта или мобильное приложение). После ввода данных система проверяет наличие топлива в базе данных, отображает доступные варианты и позволяет клиенту подтвердить транзакцию. Система регистрирует операцию, обновляет данные о запасах топлива и формирует чек. Автоматические уведомления о статусе транзакции, низком уровне топлива или необходимости технического обслуживания оборудования отправляются персоналу и, при необходимости, клиентам через интерфейс системы.

При проектировании системы требуется:

- создать иерархию классов системы, отражающую структуру и функциональность АЗС;
- для классов указать стереотипы, соответствующие их роли (например, «контроллер», «сущность», «граница»);
- организовать классы по пакетам (например, пользовательский интерфейс, управление топливом, платежи);
- связать объекты на диаграммах взаимодействия с классами, а сообщения – с соответствующими операциями;
- каждый класс снабдить описанием, включающим краткое описание (ответственность класса), таблицу атрибутов (имя, описание, тип) и таблицу операций (имя, описание, сигнатура);
- построить диаграммы классов, отображающие связи между классами;

– построить диаграммы деятельности для моделирования сложных операций, таких как процесс заправки или обработка платежа, с учетом альтернатив и циклов;

– разработать схему базы данных, включающую таблицы для хранения информации о топливе, транзакциях и клиентах, и отобразить ее на диаграмме «сущность – связь».

При реализации системы необходимо построить диаграммы компонентов для каждого пакета и для системы в целом. Также следует разработать диаграмму размещения, отражающую физическое расположение компонентов системы, включая сервер АЗС, топливные насосы, платежные терминалы и клиентские устройства. Диаграмма размещения должна учитывать связь между оборудованием АЗС и программными компонентами. Проверка корректности модели должна быть выполнена средствами *Rational Rose.*

1.2 Составление глоссария проект

Для описания терминологии предметной области составлен глоссарий проекта. Он служит неформальным словарем системы, обеспечивая единообразное понимание ключевых терминов. Глоссарий приведен в таблице.

Таблица 1 – Глоссарий проекта

Клиент (*Client*)	Пользователь, который взаимодействует с системой для выполнения заправки транспортного средства, оплаты или получения информации о транзакциях.
Оператор АЗС (*Gas Station Operator*)	Сотрудник, использующий систему для управления процессами АЗС, мониторинга запасов топлива и выполнения административных задач.
Топливный насос (*Fuel Pump*)	Устройство, обеспечивающее выдачу топлива клиенту. Объект системы, содержащий информацию о типе топлива и его доступном объеме.
Транзакция (*Transaction*)	Операция заправки или оплаты, выполненная клиентом. Объект системы, содержащий данные о времени, объеме топлива, стоимости и способе оплаты.
Запасы топлива (*Fuel Inventory*)	Совокупность данных о доступных объемах различных типов топлива на АЗС, обновляемая системой в реальном времени.
Платежный терминал (*Payment Terminal*)	Устройство или программный модуль, обеспечивающий обработку платежей клиентов (наличные, карта, мобильное приложение).

1.3 Описание дополнительных спецификаций

Функциональные возможности. Автоматизированная система автозаправочной станции должна обеспечивать удобный доступ к услугам АЗС для клиентов и эффективное управление операциями для персонала. Система выступает посредником между клиентами, оборудованием АЗС и персоналом. Клиенты могут выбирать тип топлива, объем заправки, способ оплаты, просматривать историю транзакций и получать чеки. Персонал имеет доступ к функциям мониторинга запасов топлива, формирования заказов на пополнение и управления процессами обслуживания. Система поддерживает интеграцию с платежными системами для обработки транзакций. Уровень доступа к функциям зависит от роли пользователя (клиент, оператор, администратор).

Удобство использования. Пользовательский интерфейс системы должен быть интуитивно понятным и адаптированным для работы на различных устройствах, включая стационарные компьютеры (*Windows* 7-10, *Mac OS*), мобильные платформы (*Android, iOS*), а также интерфейсы топливных насосов и платежных терминалов. Это обеспечивает удобный доступ к услугам АЗС для клиентов и персонала в любых условиях.

Надежность. Система должна быть доступна 24 часа в сутки, 7 дней в неделю, с максимальным временем простоя не более 5 %. Необходимо регулярное резервное копирование данных о транзакциях и запасах топлива для защиты от потери информации, особенно в условиях высокой нагрузки на АЗС.

Безопасность. Система должна гарантировать защиту данных о транзакциях и операциях от несанкционированного доступа. Все запросы к базе данных должны проходить через системные процедуры, обеспечивающие контроль доступа. Персонал имеет доступ только к информации, необходимой для выполнения обязанностей, без возможности прямого изменения данных в базе. Финансовые транзакции защищаются с использованием современных протоколов шифрования.

Проектные ограничения. Система должна быть разработана на основе реляционной СУБД для эффективного управления данными о топливе, транзакциях и клиентах. Также требуется обеспечить совместимость с оборудованием АЗС, включая топливные насосы и платежные терминалы, что может накладывать ограничения на выбор технологий и протоколов взаимодействия.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1 Создание модели вариантов использования

На рисунке 1 представлена модель диаграммы вариантов использования, созданная на основе поставленной задачи.

В данном курсовом проекте будут участвовать следующие лица:

- **Клиент** – пользователь, взаимодействующий с системой для заправки транспортного средства, оплаты или получения информации о транзакциях.
- **Оператор АЗС** – сотрудник, использующий систему для мониторинга запасов топлива, управления процессами обслуживания и взаимодействия с клиентами.
- **Администратор** – сотрудник, ответственный за настройку системы, управление учетными записями и контроль работы АЗС.
- **Гость (незарегистрированный пользователь)** – пользователь, который не авторизован в системе, но может просматривать базовую информацию (например, доступные виды топлива).
- **Зарегистрированный пользователь** – клиент, имеющий учетную запись в системе, что позволяет сохранять историю транзакций и получать персонализированные услуги.

В рамках проекта выделяются следующие варианты использования, структурированные по ролям пользователей:

Действия клиентов (гостей и зарегистрированных пользователей):

- Регистрация в системе – создание учетной записи для доступа к персонализированным функциям.
- Авторизация – вход в систему с использованием учетных данных.
- Заполнение личных данных – добавление информации о себе (например, контактные данные, предпочтительный способ оплаты).
- Редактирование личных данных – изменение информации в профиле.
- Выход из системы – завершение сеанса.
- Удаление личного аккаунта – деактивация учетной записи.
- Просмотр доступных видов топлива – отображение информации о типах топлива и их наличии.
- Выбор типа топлива – указание желаемого топлива для заправки.
- Указание объема заправки – выбор количества топлива.
- Выбор способа оплаты – оплата через наличные, карту или мобильное приложение.
- Оплата транзакции – завершение платежа через систему.

– Просмотр чека – получение электронного чека после заправки.

– Просмотр истории транзакций – доступ к списку предыдущих операций (доступно только для зарегистрированных пользователей).

– Отправка обратной связи – предоставление отзыва о качестве обслуживания.

Действия оператора АЗС:

– Просмотр состояния топливных насосов – мониторинг работоспособности и доступности насосов.

– Мониторинг запасов топлива – проверка текущих объемов топлива в резервуарах.

– Формирование заказа на пополнение – запрос на доставку топлива при низком уровне запасов.

– Подтверждение транзакций – проверка и регистрация операций, выполненных клиентами.

– Отправка/получение/ответ на сообщение – коммуникация с клиентами или администратором через систему.

– Планирование задач – создание задач для обслуживания оборудования или других операций.

– Пометка выполнения задачи – отметка о завершении запланированных действий.

Действия администратора:

– Создание учетных записей – регистрация новых операторов или администраторов.

– Управление учетными записями – изменение прав доступа или удаление пользователей.

– Настройка системы – конфигурация параметров (например, цены на топливо, доступные способы оплаты).

– Просмотр отчетов – анализ данных о транзакциях, запасах топлива и производительности АЗС.

– Управление топливными насосами – включение/выключение насосов, настройка параметров.

Общие действия:

– Просмотр информации об АЗС – доступ к данным о расположении, часах работы и услугах станции (доступно всем пользователям).

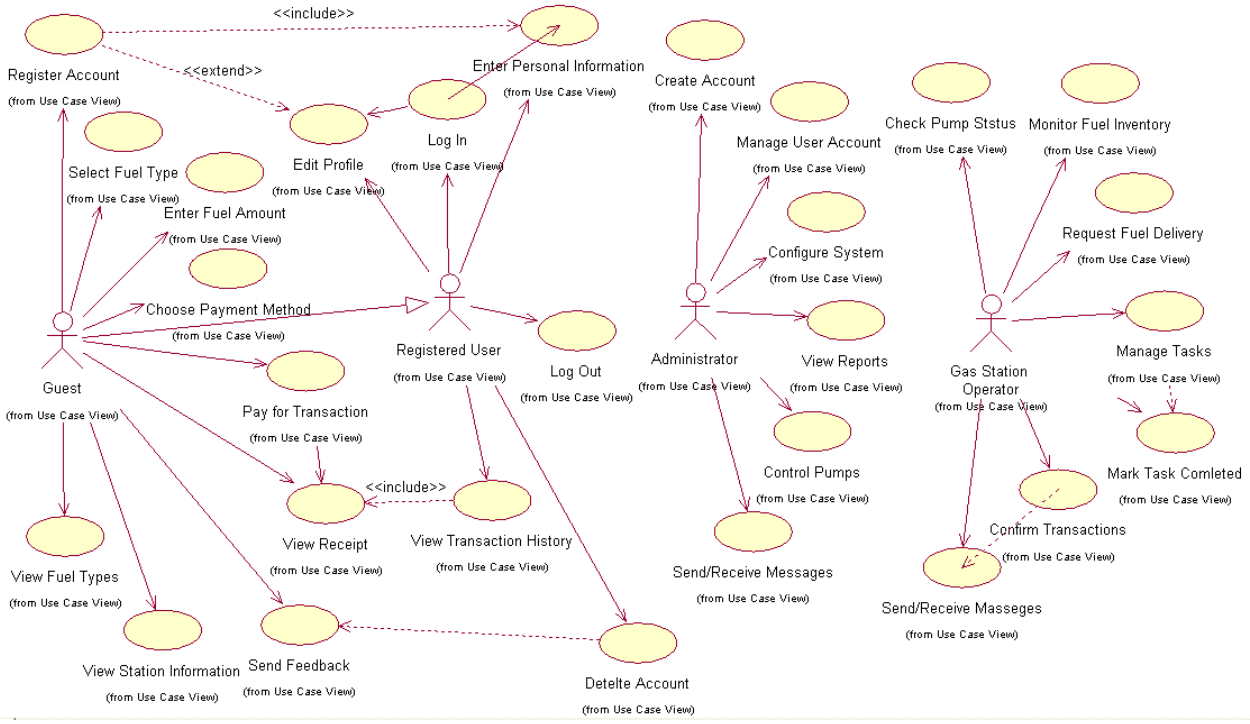

Рисунок 1 – Диаграмма вариантов использования

Данная вариантная диаграмма представляет взаимодействие между различными сценариями использования и актёрами в информационной системе. Она отражает пользовательские требования к системе.

2.2 Добавление описаний к вариантам использования

Вариант использования «Авторизация»

Краткое описание

Когда зарегистрированный пользователь (клиент, оператор или администратор) хочет получить доступ к системе АЗС, он проходит процесс аутентификации, вводя логин и пароль. В случае успешной аутентификации пользователь получает доступ к функционалу, соответствующему его роли.

Основной поток событий

Данный вариант использования начинает выполняться, когда пользователь хочет войти в систему.

1 Пользователь вводит логин в соответствующее поле.

2 Пользователь вводит пароль в соответствующее поле.

3 Пользователь отправляет запрос на вход в систему, нажав кнопку «Войти».

4 Система проверяет корректность введенных данных, сравнивая их с записями в базе данных.

5 Пользователь успешно входит в систему и перенаправляется на главную страницу, соответствующую его роли.

Альтернативные потоки

– Неправильный логин или пароль: Если введенные данные неверны, система отображает сообщение об ошибке с указанием, что логин или пароль некорректны, и предлагает повторить попытку ввода.

– Пустые поля: Если пользователь оставил одно или оба поля пустыми, система отображает сообщение об ошибке с просьбой заполнить все поля.

Предусловия

– Пользователь зарегистрирован в системе.

– Система доступна для входа.

Постусловия

– Пользователь аутентифицирован и имеет доступ к функционалу, соответствующему его роли.

Вариант использования «Заправка транспортного средства»

Краткое описание

Данный вариант использования позволяет клиенту (гостю или зарегистрированному пользователю) выбрать тип топлива, указать объем заправки, выполнить оплату и завершить процесс заправки через интерфейс системы.

Основной поток событий

Данный вариант использования начинает выполняться, когда клиент хочет заправить транспортное средство.

1 Клиент выбирает топливный насос через интерфейс системы.

2 Клиент указывает тип топлива из доступных (например, бензин АИ-95, дизель).

3 Клиент вводит желаемый объем заправки или сумму в денежном эквиваленте.

4 Система проверяет наличие указанного топлива в резервуарах.

5 Клиент выбирает способ оплаты (наличные, карта, мобильное приложение).

6 Система инициирует платеж через платежный терминал или внешнюю платежную систему.

7 После успешной оплаты система активирует топливный насос для выдачи топлива.

8 По завершении заправки система генерирует электронный чек и отображает его клиенту.

9 Клиент получает уведомление о завершении операции.

Альтернативные потоки

– Недостаток топлива: Если выбранного топлива недостаточно, система отображает сообщение об ошибке и предлагает выбрать другой тип топлива или уменьшить объем.

– Ошибка оплаты: Если платеж не прошел (например, недостаточно средств), система уведомляет клиента и предлагает выбрать другой способ оплаты или отменить операцию.

– Техническая неисправность насоса: Если насос неисправен, система отображает сообщение об ошибке и перенаправляет клиента к другому насосу.

Предусловия

– Клиент находится у топливного насоса.

– Система и оборудование АЗС (насосы, терминалы) работоспособны.

Постусловия

– Транспортное средство заправлено, транзакция зарегистрирована в системе, запасы топлива обновлены.

Вариант использования «Мониторинг запасов топлива»

Краткое описание

Оператор АЗС использует систему для проверки текущих запасов топлива в резервуарах и получения уведомлений о необходимости пополнения.

Основной поток событий

Данный вариант использования начинает выполняться, когда оператор хочет проверить состояние запасов топлива.

1 Оператор переходит в раздел мониторинга запасов в интерфейсе системы.

2 Система отображает текущие объемы топлива для каждого типа (например, АИ-92, АИ-95, дизель).

3 Если уровень топлива ниже заданного порога, система выделяет этот тип топлива и предлагает сформировать заказ на пополнение.

4 Оператор подтверждает запрос на пополнение, указывая необходимый объем.

5 Система регистрирует заказ и отправляет уведомление поставщику топлива (через *API* или внутренний модуль).

6 Оператор получает подтверждение о регистрации заказа.

Альтернативные потоки

– Ошибка данных: Если данные о запасах недоступны (например, из-за сбоя датчиков), система отображает сообщение об ошибке и предлагает оператору проверить оборудование.

– Отмена заказа: Если оператор решает не формировать заказ, система возвращает его в раздел мониторинга без изменений.

Предусловия

– Оператор авторизован в системе.

– Система имеет доступ к данным с датчиков резервуаров.

Постусловия

– Оператор проинформирован о состоянии запасов, при необходимости зарегистрирован заказ на пополнение.

2.3 Создание диаграммы последовательности

Для описания процесса «Последовательность заправки не зарегистрированного пользователя» создана диаграмма последовательности, представленная на рисунке 2.

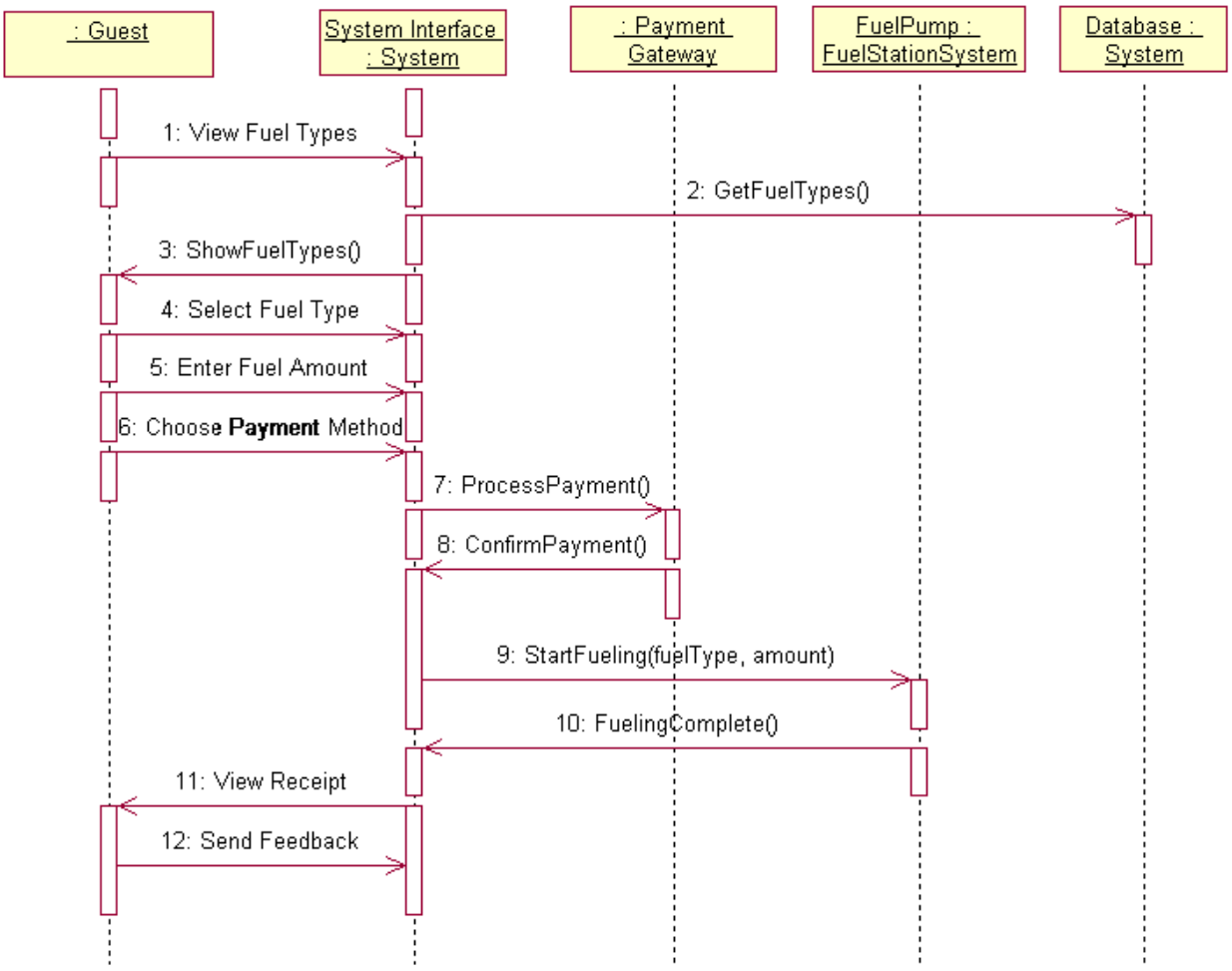

Рисунок 2 – Диаграмма последовательности «Последовательность заправки не зарегистрированного пользователя»

Диаграмма последовательности (Sequence Diagram) показывает взаимодействие между объектами (пользователями и системой) во времени: какие сообщения (действия) передаются и в каком порядке.

Последовательность действий начинается после того, как пользователь (далее Гость) приехал на заправку и остановился у колонки. После чего гость просматривает доступные виды топлива. Затем система запрашивает список видов топлива из базы данных и отображает их. Гость выбирает необходимый вид и желаемый объем топлива. Гость выбирает способ оплаты, после чего система направляет запрос на оплату в платёжную систему. Если платежная система подтверждает оплату, система запускает топливный насос с выбранными параметрами. По окончанию заправки выбранного объема топлива, насос сообщает о завершении заправки в систему. Система предоставляет гостю электронный чек и предлагает оставить отзыв о процессе заправки.

Этот сценарий охватывает последовательность взаимодействий между пользователем и системой без необходимости авторизации. Диаграмма позволяет наглядно отследить каждый шаг, включая взаимодействие с внешними компонентами, такими как платёжный шлюз и топливный насос.

Подобный подход помогает систематизировать бизнес-логику и выявить возможные узкие места в процессе. Использование диаграммы последовательности способствует лучшему пониманию логики работы системы для разработчиков и аналитиков. Кроме того, данная диаграмма может служить основой для моделирования дополнительных функций, например, регистрации пользователя или получения бонусов.

2.4 Создание кооперативной диаграммы

На рисунке 3 продемонстрирована кооперативная диаграмма. Она отражает связи между объектами во время выполнения определённого процесса.

Rational Rose позволяет автоматически генерировать кооперативную диаграмму на основе диаграммы последовательности. Главное взаимодействие происходит с базой данных. все операции в конечном итоге сводятся к запросу системе.

Кооперативная диаграмма, также известная как коммуникационная диаграмма, была разработана для отображения взаимодействия объектов

системы в рамках сценария заправки незарегистрированного пользователя. Данная диаграмма позволяет проследить, какие именно объекты участвуют в процессе, в каком порядке между ними происходят обмены сообщениями, и как осуществляется выполнение операций на разных уровнях системы.

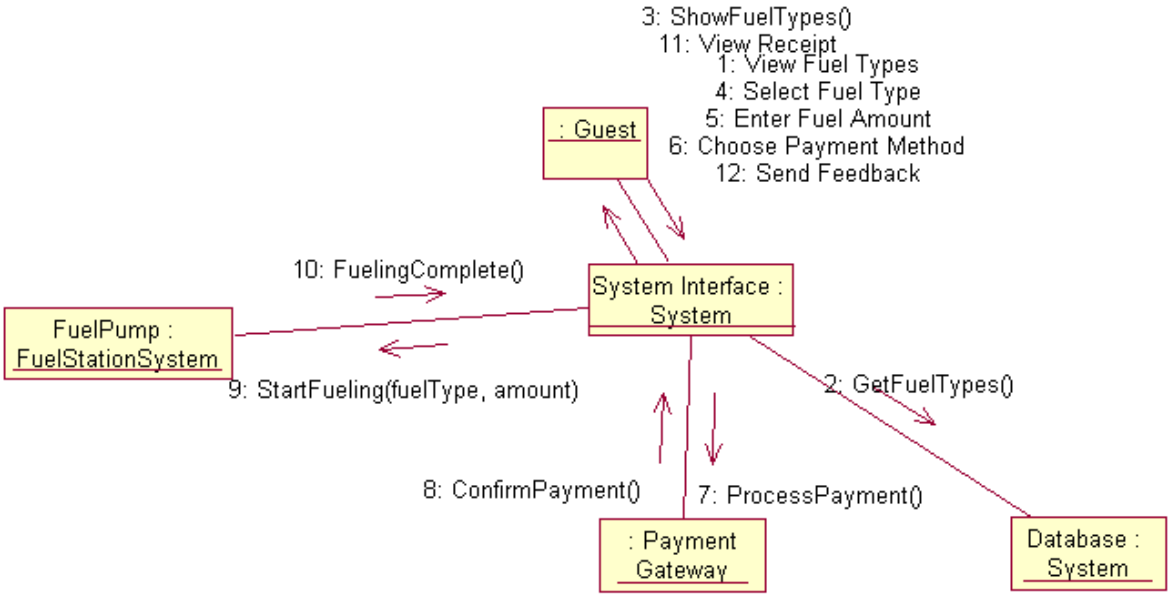

Рисунок 3 – Кооперативная диаграмма «Заправки не зарегистрированного клиента»

В кооперативной диаграмме основной фокус сделан не на времени, как в диаграмме последовательности, а на структуре взаимодействий между объектами. Это позволяет более чётко представить архитектуру обработки запроса, взаимосвязи между объектами и распределение ответственности. Основными участниками в данном взаимодействии выступают объекты: пользователь (гость), интерфейс системы, сервис транзакций, платёжный модуль, топливный насос и модуль генерации чека.

Диаграмма демонстрирует, как пользователь инициирует заправку, взаимодействуя с интерфейсом. Система последовательно обращается к платёжному процессору, после чего активирует насос. По завершении процесса система направляет пользователю подтверждение и предоставляет чек. Каждое сообщение между объектами в диаграмме помечено номером, отражающим порядок вызова, что делает структуру взаимодействия прозрачной и логичной.

Создание кооперативной диаграммы позволило уточнить роли компонентов в реализации бизнес-логики и проанализировать возможные

точки расширения или модификации системы. Она также служит важным инструментом на этапе детального проектирования, облегчая дальнейшую реализацию и отладку взаимодействия между объектами в программной части.

2.5 Создание диаграммы классов

Все объекты, участвующие в системе, представляются с помощью диаграммы классов. Для начала распределим объекты по их типу по папкам.

На рисунке 4 продемонстрировано распределение классов по папкам, в соответствии с их стереотипом.

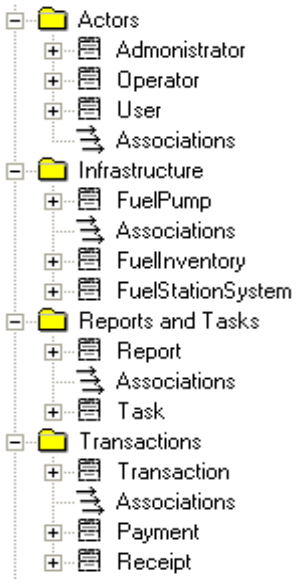

Рисунок 4 – Распределение классов по пактам

Главная диаграмма классов создана для демонстрации взаимодействия между пакетами классов системы и представлена на рисунке 5.

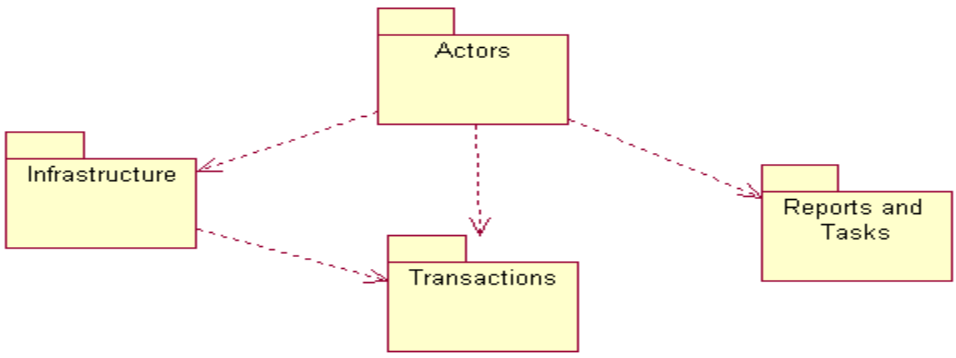

Рисунок 5 – Диаграмма пакетов

Активное взаимодействие происходит между классами управления и сущностей. Классы управления корректируют взаимодействие между сущностями. На рисунке 6 находится основная диаграмма классов.

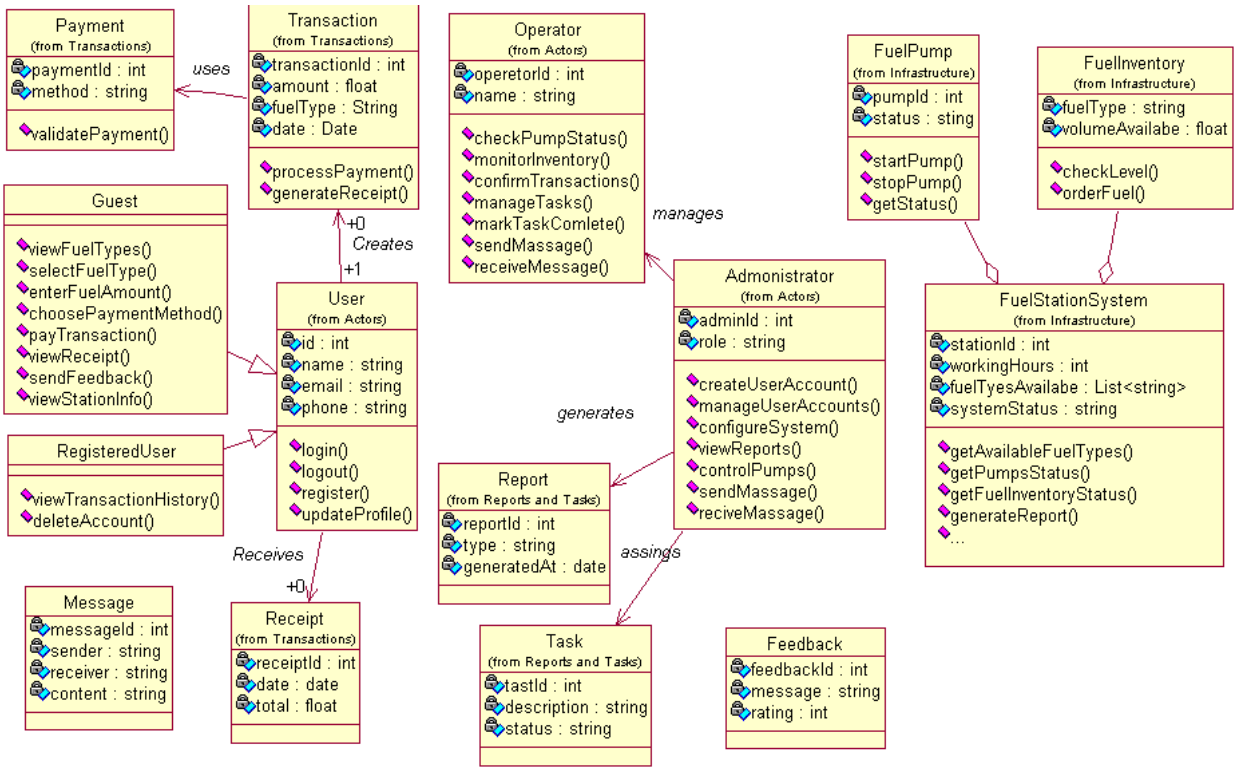

Рисунок 6 – Диаграмма классов

Диаграмма классов является одним из ключевых структурных элементов в объектно-ориентированном проектировании системы автоматизации автозаправочной станции (АЗС). Она отражает статическую структуру системы, задаёт логические связи между основными сущностями, а также определяет внутреннее представление данных и доступные операции для каждого класса. В рамках проекта диаграмма классов выполняет функцию каркаса, на основе которого строится реализация поведения системы и взаимодействия между её компонентами.

На разработанной диаграмме классов выделены иерархии пользователей, включая базовый абстрактный класс *User*, от которого наследуются конкретные типы пользователей — *Guest* (гость) и *RegisteredUser* (зарегистрированный пользователь). Это решение позволяет централизовать общие свойства и методы, обеспечивая повторное использование кода и соблюдение принципов наследования.

Класс *Transaction* представляет собой объект, отвечающий за выполнение операций заправки: хранение информации о типе топлива, объёме, способе оплаты и времени транзакции. Он связан с классами *FuelPump* и *PaymentGateway*, обеспечивающими доступ к физическому оборудованию и платёжным сервисам соответственно. Класс *FuelInventory* реализует контроль запасов топлива, позволяя системе отслеживать уровень топлива по типам и формировать заказы на пополнение.

Класс *Administrator* инкапсулирует функции управления системой: настройка параметров, управление учётными записями, генерация отчётов. Он взаимодействует с *Report* и *Task*, что позволяет реализовать управление заданиями операторов и сбор аналитической информации о работе АЗС.

Все классы связаны между собой с использованием ассоциаций, агрегаций и наследования, что отражает реальные зависимости и потоки данных в системе. Использование множества отношений позволяет задать логические ограничения и роли классов в общей архитектуре.

Диаграмма классов также включает пакетную структуру, распределяя классы по логическим модулям: *Actors*, *Transactions*, *Infrastructure* и *ReportsAndTasks*. Это улучшает читаемость и масштабируемость модели, а также соответствует принципам модульности и разделения ответственности.

Таким образом, диаграмма классов выступает фундаментом архитектурной модели и определяет все основные элементы системы, их атрибуты, поведение и связи. Она обеспечивает переход от концептуальной модели к конкретной реализации и служит основой для автоматической генерации кода.

2.6 Создание диаграммы состояний

Создадим диаграмму состояний для демонстрации форм, которые принимает объект класса «*Transaction*» в процессе выполнения системы. Данная диаграмма представлена на рисунке 7.

На диаграмме состояний, изображенной на рисунке 7, представлены основные состояния объекта *Transaction* в системе. Следуя классическому сценарию, процесс начинается с инициализации оплаты.

После этого, процесс переходит в состояние ожидания оплаты, где система ждёт одобрения оплаты, после чего дает сигнал насосу для заправки. После чего насос даёт сигнал об окончании заправки и клиент получает чек. Если же оплата не поступает или клиент передумал, наступает окончание процесса.

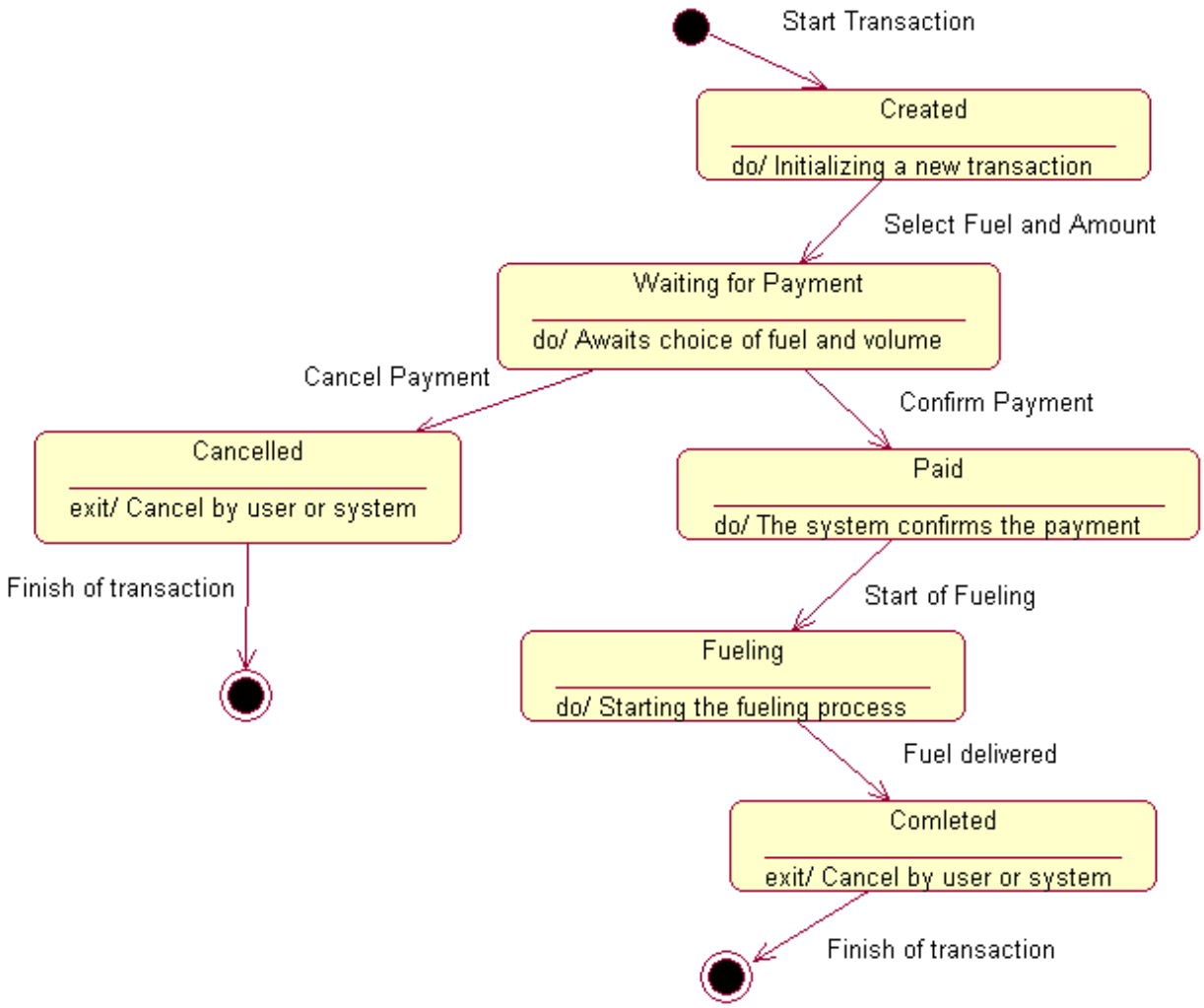

Рисунок 7 – Диаграмма состояний «*Transaction*»

2.7 Создание диаграммы активности

На данной диаграмме будет показан процесс от начала до завершения заправки. Диаграмма представлена на рисунке 8.

Клиент нажимает на интерфейс, после чего просматривает доступные виды топлива (бензин, дизель, газ). Определившись, выбирает необходимый тип топлива и вводит количество. Далее выбирает способ оплаты и оплачивает.

Если система фиксирует одобренный платёж, выводит на экран советующее сообщение и печатает чек. Система уточняет, желает ли клиент оставить связь. Если клиент соглашается, открывается окно, где клиент оставляет обратную связь, после чего активность завершается. Если же

платёжная операция не одобрена и клиент не желает оставлять обратную связь, активность так же принудительно завершается.

Данный процесс построен с учётом логики работы неавторизованного пользователя, что позволяет реализовать заправку без необходимости регистрации. Все действия клиента реализуются через последовательные действия интерфейса, включая выбор параметров и подтверждение транзакции. В диаграмме отражены как стандартный сценарий, так и альтернативные пути развития событий — например, отмена операции при отказе в оплате. Это делает диаграмму универсальной и пригодной для моделирования различных пользовательских ситуаций. Каждое действие отображается в виде отдельного элемента, связанного управляющими переходами, демонстрируя логику системы от инициации до завершения. Диаграмма построена по принципам *UML* и соответствует требованиям к моделированию бизнес-процессов. Такая детализация позволяет разработчикам точно реализовать поведение интерфейса и системы при взаимодействии с пользователем.

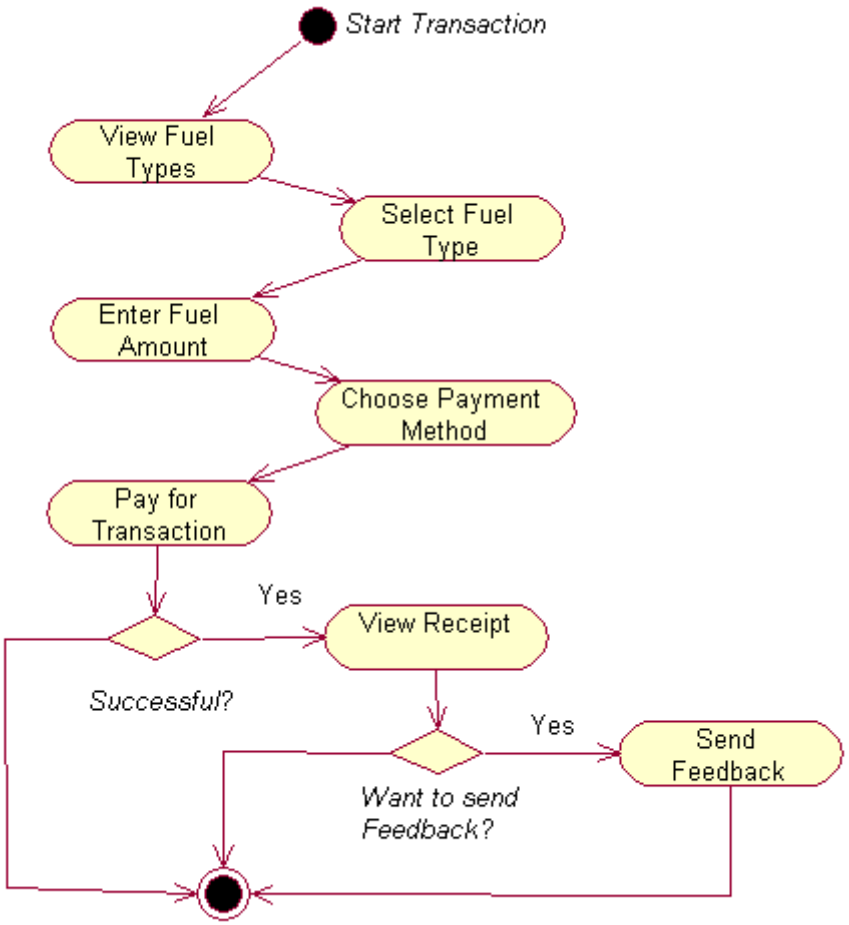

Рисунок 8 – Диаграмма активности «*Transaction*»

3 РЕАЛИЗАЦИЯ СИСТЕМЫ

3.1 Создание диаграммы

Диаграммы компонентов представляют собой визуализацию физического уровня модели системы, отображая компоненты программного обеспечения и их взаимосвязи. Эти диаграммы играют важную роль в начальных этапах разработки и реализации системы.

На диаграммах компонентов можно выделить два основных типа компонентов: исполняемые компоненты и библиотеки кода. Исполняемые компоненты представляют собой части программного обеспечения, которые выполняют определенные функции и задачи в системе. Библиотеки кода представляют собой наборы готовых программных модулей или функций, которые могут быть использованы различными компонентами системы.

Диаграмма компонентов, представленная на рисунке 9, отражает конкретную модель системы, показывая расположение и взаимодействие компонентов на физическом уровне. Эта диаграмма является важным этапом в процессе реализации системы, так как она помогает визуализировать структуру и организацию компонентов программного обеспечения.

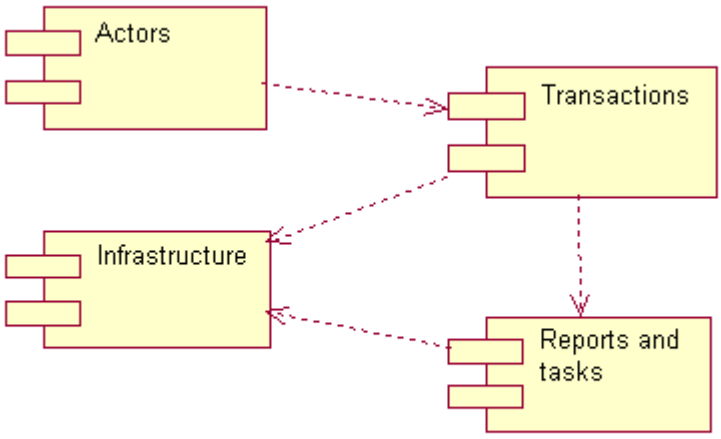

Рисунок 9 – Диаграмма компонентов

Диаграмма является важным инструментом, который помогает программистам лучше понять спецификации и взаимодействие различных элементов системы. Это является неотъемлемой частью процесса разработки,

поскольку обеспечивает правильную организацию написания кода, повышает продуктивность и снижает вероятность возникновения ошибок.

Каждая спецификация имеет свое тело, которое содержит ключевые моменты создания и соответствующий код. Эти тела спецификаций играют важную роль в определении порядка работы. Посредством связей на диаграмме можно определить, с каких тел следует начинать и какие элементы системы являются первоочередными.

Из диаграммы видно, что конечная цель системы заключается в спецификации запросов. Это можно считать своеобразным ядром системы, которому всегда придается высший приоритет. Создание ядра системы должно быть основным фокусом в начале разработки, а остальные части системы могут быть разработаны по мере продвижения ядра. Также стоит уделить особое внимание спецификациям, связанным с обработкой запросов, поскольку они являются ключевыми для функционирования системы.

Диаграммы спецификаций и их взаимодействие обеспечивают четкое представление о структуре системы и ее составляющих. Это позволяет команде разработчиков лучше понимать требования и цели проекта, а также эффективно планировать и координировать работу. Благодаря детальной визуализации и описанию спецификаций, участники проекта могут более точно согласовывать свои действия, минимизировать риски и повысить качество разработки системы.

3.2 Создание диаграммы размещения

Диаграмма размещения применяется для представления общей конфигурации и топологии распределённой программной системы и содержит изображение размещения компонентов по отдельным узлам системы. Кроме того, диаграмма размещения показывает наличие физических соединений, а именно маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Для отображения устройств, на которых будет размещена система, представлена на рисунке 10.

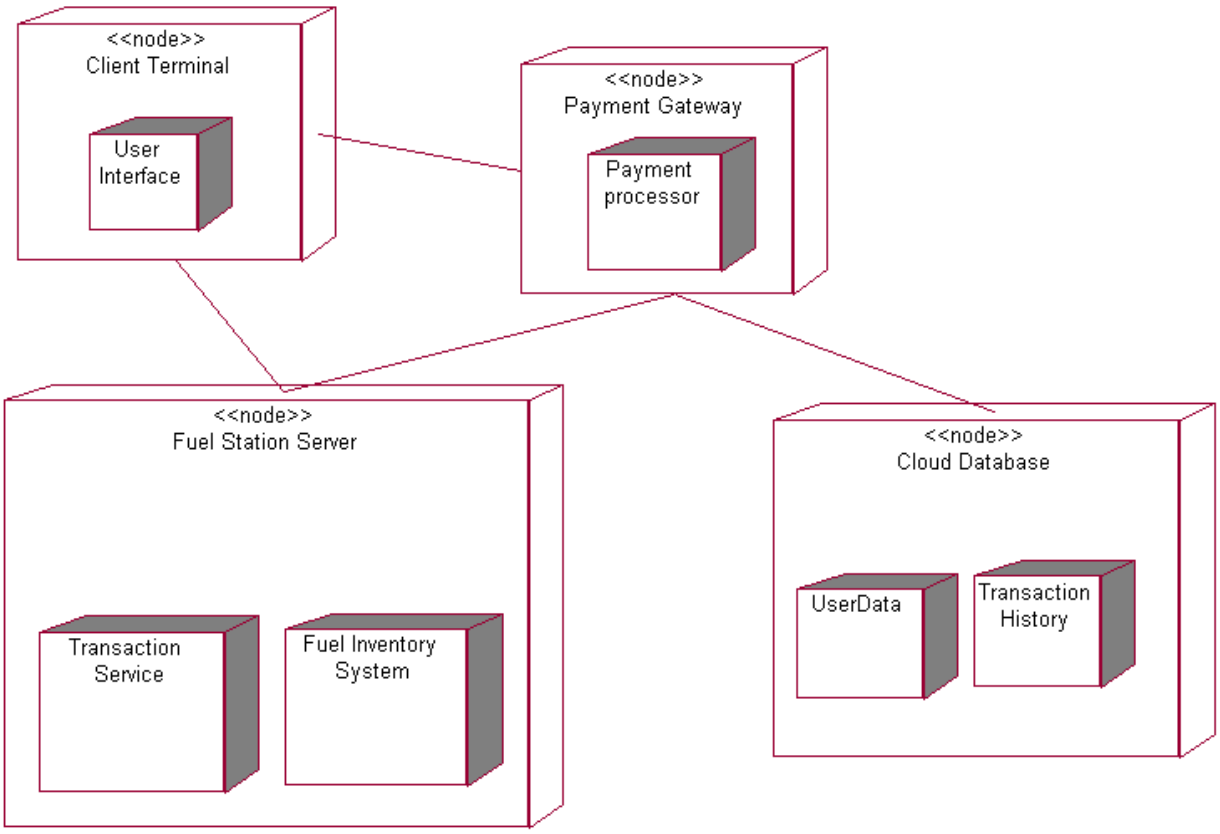

Рисунок 10 – Диаграмма размещения

Для эффективной работы автоматизированной системы управления АЗС в проекте задействованы четыре основных узла, каждый из которых отвечает за выполнение определённых функций.

Первым узлом является *Client Terminal* – клиентское устройство, через которое пользователь взаимодействует с системой. На нём размещён модуль *User Interface*, предоставляющий доступ к операциям выбора топлива, ввода объёма, способа оплаты и получения чека. Этот узел обеспечивает удобный интерфейс и инициализирует основной пользовательский сценарий.

Второй узел – *Fuel Station Server* – выполняет основную бизнес-логику. Здесь находятся модули *Transaction Service* и *Fuel Inventory System.* Первый обрабатывает транзакции и координирует процесс заправки, а второй отслеживает остатки топлива. Этот сервер связывает действия пользователя с физическими ресурсами АЗС.

Третий узел – *Payment Gateway* – представляет собой платёжный шлюз с модулем *Payment Processor*. Он обрабатывает запросы на оплату, обеспечивает безопасность транзакций и взаимодействует с внешними финансовыми системами.

Четвёртым узлом выступает *Cloud Database*, в которой размещены модули *UserData* и *TransactionHistory*. Этот узел отвечает за долговременное хранение информации о пользователях, истории заправок и других сервисных данных. Он позволяет обеспечить доступность и устойчивость данных при масштабировании системы.

Такое распределение компонентов по узлам позволяет системе быть гибкой, масштабируемой и надёжной, обеспечивая стабильную работу всех функций заправочного комплекса и удобство для конечного пользователя.

3.3 Генерация кода *C++*

Воспользуемся возможностью *Rational Rose* для генерации макетов кода для созданных в диаграммах классов. На основе полученных данных созданы классы, в их спецификациях можно указать принадлежность к *C++* коду. Отобразим на рисунке 11 все сгенерированные файлы. На рисунке видно, что программа генерирует заголовочные файлы, облегчая ориентацию.

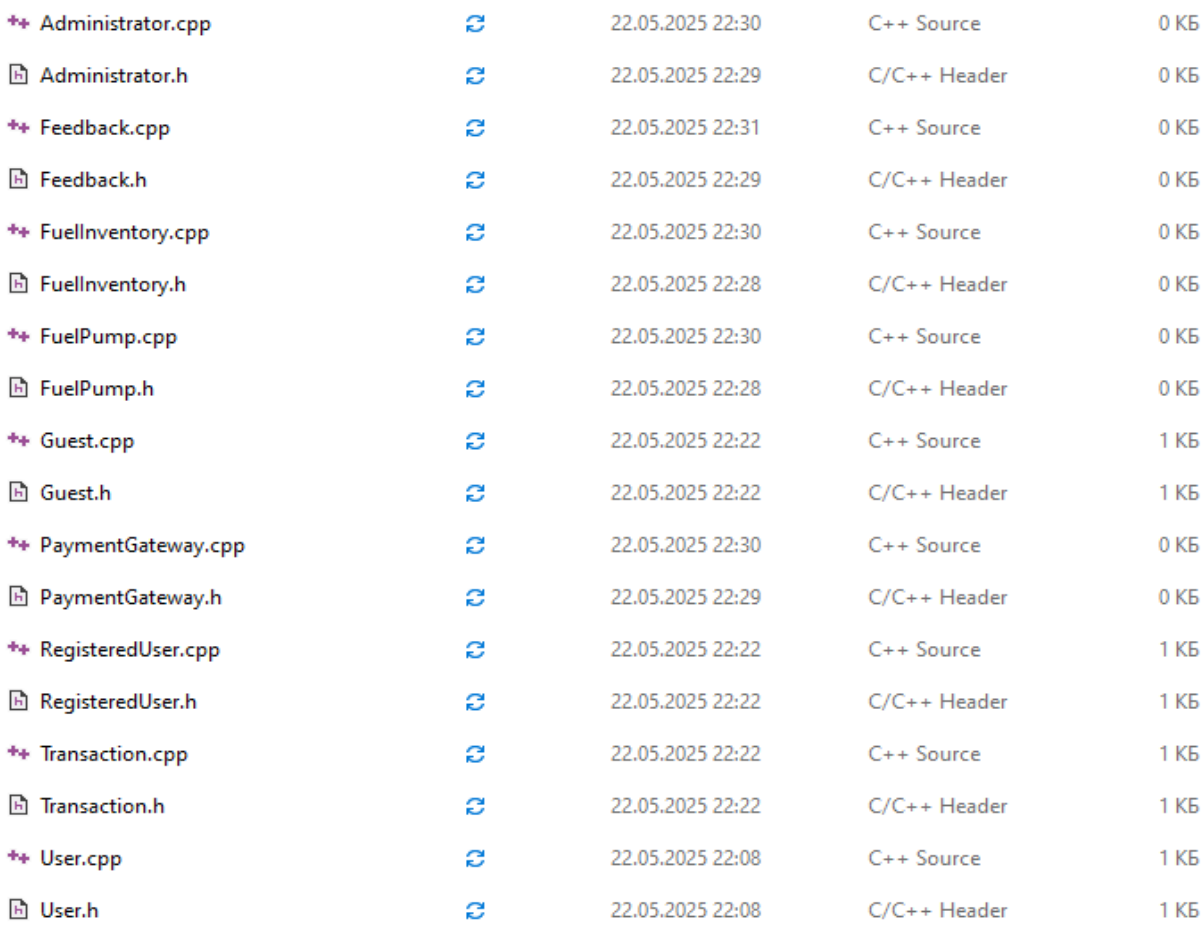

Рисунок 11 – Сгенерированные файлы

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта проанализирован предметный контекст, на основе которого разработана функциональная модель системы Проектирование автоматизированной системы автозаправочной станции.

Для проектирования системы использовались программа *Rational Rose* и язык объектно-ориентированного проектирования *UML*. В процессе работы созданы основные диаграммы, а также составлены описания для них.

Первоначально проведен анализ задачи, определены требования и условия. На основе полученных рекомендаций разработаны диаграммы взаимодействия, компонентов, размещения, активности, состояний, вариантов использования и классов.

Диаграмма вариантов использования отображает основные требования к системе и является первым шагом в процессе проектирования. Она представляет требования к системе с точки зрения пользователя.

Создана диаграмма последовательности для заправки не зарегистрированного пользователя, которая наглядно демонстрирует основные шаги заправки и последовательность действий. Диаграммы последовательности упорядочены по времени, а кооперативные диаграммы акцентируют внимание на взаимосвязях между объектами. Для более ясного представления разработана кооперативная диаграмма заправки не зарегистрированного клиента.

Далее создана диаграмма классов, включая общую диаграмму пакетов и взаимодействий между ними. Основная диаграмма классов разработана для проектирования самих классов и их ассоциаций. На этой диаграмме описаны все классы, которые составляют систему.

Особое внимание уделено диаграмме состояний класса, поскольку вокруг него строится вся система, хотя в основном взаимодействии участвует база данных. Диаграмма является ключевой информационной структурой в системе, и поэтому он заслуживает отдельного рассмотрения.

Для создания кода использована диаграмма компонентов. Спецификации этой диаграммы позволяют наглядно представить общую картину системы и дискретно разбить ее на небольшие компоненты. Используя эти спецификации, мы могли начинать разработку кода частями или делегировать компоненты отдельным группам программистов, обеспечивая их взаимодействие.

Информацию об организации устройств системы представлена на диаграмме размещения.

Использована функция генерации *C++* кода для создания классов в нашей системе. Этот процесс позволил сгенерировать макеты классов, которые затем могли быть дальше развиты и доработаны.

В результате использования *UML* языка построена система, используя различные типы диаграмм. Это позволяет детально спроектировать программу, начиная с первоначальных требований на диаграмме вариантов использования и заканчивая генерацией кода.

Кроме того, в проекте были определены основные роли пользователей, включая гостей, зарегистрированных клиентов, операторов и администраторов, с учётом их прав и сценариев взаимодействия с системой. Для обеспечения структурности и масштабируемости проекта были выделены ключевые пакеты: *Actors*, *Transactions*, *Infrastructure* и *Reports and Tasks*, что позволило логически организовать архитектуру системы.

Каждый элемент системы разрабатывался с учётом принципов объектно-ориентированного подхода, включая инкапсуляцию, наследование и полиморфизм.

Разработка диаграмм взаимодействий позволила выявить связи между объектами и уточнить роли компонентов в реализации функциональности. Система спроектирована с возможностью расширения, благодаря чему в дальнейшем могут быть добавлены новые функции, такие как авторизация через мобильное приложение или интеграция с облачными сервисами аналитики.

Процесс проектирования сопровождался итеративной проверкой логики между уровнями моделей, что обеспечило целостность архитектуры. В целом, реализация проекта с использованием *UML*-диаграмм и *Rational Rose* позволила создать гибкую, расширяемую и понятную модель автоматизированной системы автозаправочной станции.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Вендров, А. М. Объектно-ориентированный анализ и проектирование с использованием языка *UML* и *Rational Rose* / А.М. Вендров. – М. : Москва, 2004. – 54 с.

[2] Боггс, У. UML и Rational Rose / У. Боггс, М. Боггс. – Москва : ДМК Пресс, 2002. – 510 с.

[3] СТП 01-2017. Стандарт предприятия. Дипломные проекты (работы). Общие требования. - Минск: БГУИР, 2017. - 169 с.

[4] UML – это просто. Диаграммы последовательностей. [Электронный ресурс]. – Режим доступа: http://www.larin.in/archives/

[5] Буч, Г. Введение в UML от создателей языка / Г. Буч, Дж. Рамбо, И. Якобсон. – Москва : ДМК Пресс, 2015. – 496 с.

[6] Киммел, П. UML. Основы визуального анализа и проектирования / П. Киммел. – Москва : НТ Пресс, 2008. – 272 с.

[7] Белов, В.В. Проектирование информационных систем: Учебник / В.В. Белов. – Москва : Академия, 2018. – 144 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг реализации классов

Transaction.cpp

```
#include "Transaction.h"

#include <iostream>

#include <iomanip>

Transaction::Transaction(int id, const std::string& fuel, float amt, float price, const std::string& method, const std::string& date)

    : transactionId(id), fuelType(fuel), amount(amt), pricePerLiter(price),

      paymentMethod(method), transactionDate(date), isPaid(false) {}


void Transaction::process() {

    std::cout << "Processing transaction #" << transactionId << "...\n";

    isPaid = true; // имитируем успешную оплату

}


void Transaction::generateReceipt() const {

    std::cout << "\n=== RECEIPT ===\n";

    std::cout << "Transaction ID: " << transactionId << "\n";

    std::cout << "Fuel Type: " << fuelType << "\n";

    std::cout << "Amount: " << amount << " L\n";

    std::cout << "Price per Liter: " << pricePerLiter << " USD\n";

    std::cout << "Total: " << std::fixed << std::setprecision(2) << calculateTotal() << " USD\n";

    std::cout << "Payment Method: " << paymentMethod << "\n";

    std::cout << "Date: " << transactionDate << "\n";
```

```
        std::cout << "Status: " << (isPaid ? "Paid" : "Not Paid")
<< "\n";
        std::cout << "================\n";
    }

    float Transaction::calculateTotal() const {
        return amount * pricePerLiter;
    }

    bool Transaction::getPaymentStatus() const {
        return isPaid;
    }

    int Transaction::getId() const {
        return transactionId;
    }

    std::string Transaction::getFuelType() const {
        return fuelType;
    }

    float Transaction::getAmount() const {
        return amount;
    }

    std::string Transaction::getDate() const {
        return transactionDate;
    }
```

Transaction.h

```
#ifndef TRANSACTION_H
#define TRANSACTION_H
```

```
#include <string>

class Transaction {
private:
    int transactionId;
    std::string fuelType;
    float amount;
    float pricePerLiter;
    std::string paymentMethod;
    std::string transactionDate;
    bool isPaid;

public:
    Transaction(int id, const std::string& fuel, float amount,
float price, const std::string& method, const std::string& date);

    void process();
    void generateReceipt() const;
    float calculateTotal() const;
    bool getPaymentStatus() const;

    // Геттеры
    int getId() const;
    std::string getFuelType() const;
    float getAmount() const;
    std::string getDate() const;
};

#endif // TRANSACTION_H
```

Administrator.h

```
#ifndef ADMINISTRATOR_H
#define ADMINISTRATOR_H


#include <string>


class Administrator {
private:
    std::string name;
    int adminId;
```

```
public:
    Administrator(const std::string& name, int id);

    void createUser();
    void manageAccounts();
    void configureSystem();
    void generateReport();
};

#endif // ADMINISTRATOR_H
```

Administrator.cpp

```
#include "Administrator.h"
#include <iostream>

Administrator::Administrator(const std::string& name, int id)
    : name(name), adminId(id) {}

void Administrator::createUser() {
    std::cout << "[Administrator] Creating new user account...\n";
}

void Administrator::manageAccounts() {
    std::cout << "[Administrator] Managing existing user accounts...\n";
}

void Administrator::configureSystem() {
    std::cout << "[Administrator] Configuring fuel station system settings...\n";
}
```

```
void Administrator::generateReport() {
    std::cout << "[Administrator] Generating performance and transaction report...\n";
}
```

GasStationOperator.h

```
#ifndef GASSTATIONOPERATOR_H
#define GASSTATIONOPERATOR_H

#include <string>

class GasStationOperator {
private:
    std::string name;
    int operatorId;

public:
    GasStationOperator(const std::string& name, int id);

    void checkPumpStatus();
    void monitorFuelInventory();
    void requestFuelDelivery();
    void confirmTransactions();
    void manageTasks();
    void markTaskCompleted();
    void sendMessageToAdmin(const std::string& message);
};

#endif // GASSTATIONOPERATOR_H
```

GasStationOperator.cpp

```
#include "GasStationOperator.h"
#include <iostream>

GasStationOperator::GasStationOperator(const std::string& name,
int id)
    : name(name), operatorId(id) {}

void GasStationOperator::checkPumpStatus() {
    std::cout << "[Operator] Checking fuel pump status...\n";
}

void GasStationOperator::monitorFuelInventory() {
    std::cout << "[Operator] Monitoring fuel inventory
levels...\n";
}

void GasStationOperator::requestFuelDelivery() {
    std::cout << "[Operator] Requesting new fuel delivery...\n";
}

void GasStationOperator::confirmTransactions() {
    std::cout << "[Operator] Confirming completed
transactions...\n";
}

void GasStationOperator::manageTasks() {
    std::cout << "[Operator] Managing assigned operational
tasks...\n";
}

void GasStationOperator::markTaskCompleted() {
```

```
    std::cout << "[Operator] Marking task as completed...\n";
}

void GasStationOperator::sendMessageToAdmin(const std::string& message) {
    std::cout << "[Operator] Sending message to administrator: " << message << "\n";
}
```

Main.cpp

```
#include <iostream>
#include <string>
#include "Transaction.h"
#include "Administrator.h"
#include "GasStationOperator.h"
#include "Guest.h"
#include "RegisteredUser.h"

int main() {
    std::cout << "=== Gas Station Automation System ===\n\n";

    // Администратор
    Administrator admin("Ivan Petrov", 1);
    admin.createUser();
    admin.configureSystem();
    admin.generateReport();

    std::cout << "\n";

    // Оператор АЗС
    GasStationOperator operator1("Sergey Ivanov", 101);
```

```
    operator1.checkPumpStatus();
    operator1.monitorFuelInventory();
    operator1.confirmTransactions();

    std::cout << "\n";

    // Гость заправляется
    Guest guest;
    guest.viewFuelTypes();
    guest.selectFuelType("Diesel");
    guest.enterFuelAmount(20);
    guest.choosePaymentMethod("Card");
    guest.payTransaction();
    guest.viewReceipt();
    guest.sendFeedback("Everything was great!");

    // Зарегистрированный пользователь
    RegisteredUser user;
    user.viewFuelTypes();
    user.viewTransactionHistory();
    user.deleteAccount();
    // Транзакция
    Transaction tx(1001, "Diesel", 20.0f, 1.75f, "Card", "2025-
05-23");
    tx.process();
    tx.generateReceipt();

    std::cout << "\n=== Simulation Complete ===\n";

    return 0;
}
```

ВЕДОМОСТЬ КУРСОВОГО ПРОЕКТА

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 6-05-0612-03 029 ПЗ	Пояснительная записка	40с.
	Графические документы	
ГУИР.000000.001 ПД	Диаграмма вариантов	
	использования	Формат А1

Изм.	Л	№ докум.	Подп.	Дата	БГУИР КП 6-05-0612-03 029 ПЗ			
Разраб.		Яковчик			Проектирование автоматизированной системы автозаправочной станции	Лист	Лист	Листов
Провер.		Боброва				Т	40	40
Т.контр.						Кафедра ИТАС гр. 320601		
Н.контр.								
Утв.								
Реценз.								