

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
(НИЯУ МИФИ)
ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ
КАФЕДРА КИБЕРНЕТИКИ

На правах рукописи

УДК 004.4

ЧУДНОВЕЦ И.В.

ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
ТЕРМИНАЛОВ САМООБСЛУЖИВАНИЯ ДЛЯ АЗС

Выпускная квалификационная работа бакалавра

Направление подготовки 01.03.02 Прикладная математика и информатика

Книга 1

Выпускная квалификационная
работа защищена

«__» _____ 20 __ г.

Оценка _____

Секретарь ГЭК _____

г. Москва

2021



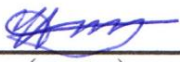
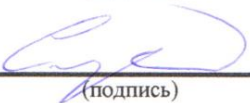

**Институт
интеллектуальных кибернетических систем
Кафедра №22 «Кибернетика»**

Направление подготовки 01.03.02 Прикладная математика и информатика

Пояснительная записка

к ВКР на тему:

**Проектирование и разработка программного обеспечения терминалов
самообслуживания для АЗС**

Группа	<u>Б17-511</u>	
Студент	<u></u> (подпись)	<u>Чудновец И.В.</u> (ФИО)
Руководитель	<u></u> (подпись)	<u>Смирнов Д.А.</u> (ФИО)
Научный консультант	<u></u> (подпись)	<u>Коновалов Р.В.</u> (ФИО)

Москва 2021




Институт	Интеллектуальных кибернетических систем	Кафедра Группа	№22 «Кибернетика» Б17-511
----------	--	-------------------	------------------------------

Специальность (направление)	Прикладная математика и информатика (01.03.02)	«Утверждаю» Зав. кафедрой
--------------------------------	--	------------------------------

Загребаев А.М.
(подпись)
«__» _____ 20__ г.

ЗАДАНИЕ НА ДИПЛОМНУЮ РАБОТУ

(выпускную квалификационную работу ВКР)

1. Фамилия, имя, отчество студента
Чудновец Иван Владимирович
(ФИО) 
(подпись)
2. Тема работы (ВКР) « Проектирование и разработка программного
обеспечения терминалов самообслуживания для АЗС »
3. Срок сдачи студентом готовой работы: 04 июня 2021 г.
4. Место выполнения ООО "Экспертек ИБС"
5. Руководитель работы
Смирнов Д.А.
(ФИО) Директор дирекции
по развитию
розничных решений
(ПО), ООО
"Экспертек ИБС"
(уч. степень, должность) 
(подпись)
6. Консультант работы
Коновалов Р.В.
(ФИО) Старший
преподаватель
(уч. степень, должность) 
(подпись)

1. Аналитическая часть

Анализ требований заказчика на разрабатываемое программное обеспечение.
Функциональные возможности клиентского GUI.
Хранение и обработка данных.
Требования к ОС.

2. Теоретическая часть

Выбор архитектуры ИС.
Анализ различных протоколов обмена данными.
Выбор технологий для разработки.

3. Технологическая часть

Отбор данных, используемых при работе ТСО.
Моделирование бизнес-процессов и бизнес-логики взаимодействия пользователя с интерфейсом ТСО на основе ТЗ.
Проектирование БД REST-сервиса.
Разработка алгоритмов обработки и хранения данных на стороне REST-сервиса.
Проектирование модулей ИС.

4. Практическая часть

Реализация HTTP процедур обмена данными в соответствии с выбранным протоколом REST.
Разработка компонентов ИС.
Демонстрация работы компонентов системы.

Дата выдачи задания «15» февраля 2021 г.

РЕФЕРАТ

Пояснительная записка содержит 188 страниц, 2 книги, 104 рисунка. Количество использованных источников – 16.

БИЗНЕС-ПРОЦЕССЫ, БИЗНЕС-ЛОГИКА, СЕРВИСЫ, REST, SOAP, API, HTTP-МЕТОДЫ, JSON, БД, МОДЕЛЬ, СЕРИАЛИЗАТОР, КОНТРОЛЛЕР, ИНТЕРФЕЙС, GUI

Целью данной работы является определение методов подхода к реализации программного обеспечения, анализ функциональности, разработка архитектуры и пример последующей реализации модулей информационной системы для терминалов саообслуживания автозаправочной станции (АЗС).

В первом разделе основного тома происходит анализ требований заказчика на разрабатываемое программное обеспечение.

Во втором разделе основного тома осуществляется выбор архитектуры ИС, анализ различных протоколов обмена данными и выбор технологий для разработки.

В третьем разделе основного тома производится отбор данных, используемых ТСО, моделирование и проектирование всех модулей ИС, с учётом особенностей форматов данных и разработка эффективных алгоритмов для работы с ними.

В четвёртом разделе основного тома описана реализация компонентов ИС (HTTP-процедур обмена данными, REST-сервиса, графического интерфейса для ТСО).

В заключении подводятся итоги проведённой работы с кратким описанием результатов по каждому разделу.

Книга приложений служит расширением основного тома, где приведено большое количество скриншотов программы, а также её исходный код.

СОДЕРЖАНИЕ

Книга 1

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	8
ВВЕДЕНИЕ	11
Раздел 1. Постановка задачи.....	13
1.1 Анализ требований заказчика на разрабатываемое программное обеспечение	13
1.1.1 Функциональные возможности клиентского GUI.....	14
1.1.2 Хранение и обработка данных	19
1.1.3 Требования к ОС.....	20
Раздел 2. Описание используемых технологий	21
2.1 Выбор архитектуры ИС	21
2.1.1 Сервис-ориентированная архитектура (микросервисная архитектура)	21
2.1.2 Монолитная архитектура	22
2.2 Анализ различных протоколов обмена данными	23
2.2.1 SOAP веб-сервисы	23
2.2.2 REST веб-сервисы	24
2.3 Выбор технологий для разработки.....	26
2.3.1 ЯП и фреймворки для WEB-разработки	26
2.3.2 Анализ особенностей фреймворков Django и DRF для разработки веб-служб REST	27
2.3.3 ЯП и фреймворки для разработки GUI приложения.....	28
2.3.4 Анализ особенностей фреймворка PyQt5 для разработки GUI приложения.....	28
Раздел 3. Проектирование архитектуры компонентов ИС	30
3.1 Отбор данных, используемых при работе ТСО	30
3.2 Моделирование бизнес-процессов и бизнес-логики взаимодействия пользователя с интерфейсом ТСО на основе ТЗ.....	31
3.3 Проектирование БД REST-сервиса	35
3.4 Разработка алгоритмов обработки и хранения данных на стороне REST-сервиса	36

3.5 Проектирование модулей ИС	37
3.5.1 Проектирование архитектуры REST-сервиса в соответствии с паттерном MVC	37
3.5.2 Проектирование архитектуры графического интерфейса (GUI) для ТСО	38
Раздел 4. Программная реализация компонентов ИС	39
4.1 Реализация HTTP процедур обмена данными в соответствии с выбранным протоколом REST	39
4.2 Разработка компонентов ИС	40
4.2.1 Реализация REST-сервиса.....	40
4.2.2 Реализация графического интерфейса (GUI) для ТСО	40
4.3 Демонстрация работы компонентов системы	44
ЗАКЛЮЧЕНИЕ.....	47
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	48
Книга 2. Приложения	

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке к ВКР применены следующие сокращения и обозначения:

АЗС – автозаправочная станция

БД – база данных

БК – банковская карта

БП – бизнес-процесс

Валидатор – устройство, принимающее купюры, купюроприёмник

Веб-сервис – идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами, а также HTML-документ сайта, отображаемый браузером пользователя

Интерфейс – в компьютерной и вычислительной технике под интерфейсом понимают элементы, обеспечивающие взаимодействие аппаратных и программных средств между собой и с человеком

ИС – информационная система

Контроллер – компонент MVC, который интерпретирует действия пользователя, оповещая модель о необходимости изменений

Модель – компонент MVC, который предоставляет данные и реагирует на команды контроллера, изменяя своё состояние

НП – нефтепродукт

Паттерн – в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста

ППО – прикладное программное обеспечение

Представление – компонент MVC, который отвечает за отображение данных модели пользователю, реагируя на изменения модели

ПО – программное обеспечение

Сериализатор – класс (в программировании), извлекающий данные из модели и форматирующий их в формат JSON

Сканер – устройство ввода, которое, анализируя какой-либо объект (обычно изображение, текст), создаёт его цифровое изображение

ТК – топливная карта. Идентична БК (по возможностям оплаты товаров и услуг).

ТРК – топливно-раздаточная колонка

ТСО (терминал самообслуживания) – многозначный термин означающий в зависимости от контекста:

1) Устройство, рассчитанное на приём платежей и осуществление всевозможных платежей и переводов

2) Приложение, установленное на одноимённое устройство (см. 1)), для приёма платежей клиентов, желающих приобрести нефтепродукт или сопутствующие товары на АЗС

В дальнейшем рамках ВКР мы будем рассматривать ТСО в смысле 2). ППО, установленное на терминале – AutoGasComplex

ТУ – терминальное устройство, pos-терминал, устройство для оплаты банковской картой (БК)

Фреймворк – библиотека, задающая архитектуру программного компонента

ШК – штрих-код

ЯП – язык программирования

API – Application Programming Interface (программный интерфейс приложения)

Backend – программно-аппаратная часть сервиса

Django – свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC

DRF – это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта

GUI – graphical user interface, система средств для взаимодействия пользователя с компьютером, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т. п.)

HTML – стандартизированный язык разметки веб-страниц во Всемирной паутине

HTTP – протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате «HTML», в настоящий момент используется для передачи произвольных данных. Основой HTTP является технология «клиент-сервер», то есть предполагается существование:

- Потребителей (клиентов), которые инициируют соединение и посылают запрос
- Поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

HTTP-метод – последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом

JSON – текстовый формат обмена данными, основанный на JavaScript

MVC – Model-View-Controller, схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода

REST – архитектурный стиль взаимодействия компонентов распределённого приложения в сети

SOAP – протокол обмена структурированными сообщениями в распределённой вычислительной среде

ВВЕДЕНИЕ

Автозаправочная станция (АЗС) - комплекс оборудования на придорожной территории, предназначенный для заправки топливом транспортных средств. Наиболее распространены АЗС, заправляющие автотранспорт традиционными сортами углеводородного топлива — бензином и дизельным топливом (бензозаправочные станции). Менее распространёнными являются Автомобильная ГазоНаполнительная Компрессорная Станция (АГНКС) — заправка сжатым природным газом (CNG) и Автомобильная ГазоЗаправочная Станция (АГЗС) — заправка сжиженным нефтяным газом (LPG). В рамках ВКР будут рассмотрены АЗС, осуществляющие продажу бензина (92, 95, 98) и дизельного топлива.

На современных автозаправочных станциях зачастую сервис не ограничивается продажей топлива. Часто на таких станциях имеется небольшой магазин, реже — закусочная, а также банкомат, мойка и тому подобное. В США распространены трак-стопы или travel center в которых собственно заправка является лишь частью комплекса включающего в себя услуги стоянки для большегрузных автомобилей дальнобойщиков, центры отдыха и досуга, кафе и магазины. Таким образом, для автоматизации и учёта продаж НП и других товаров на АЗС необходимо удобное ПО.

До середины нулевых оплата заказа на АЗС осуществлялась исключительно через кассы, в которых требовалось участие продавца. К началу 10-х годов XXI века был замечен устойчивый рост использования терминалов самообслуживания (ТСО) в различных сферах бизнеса, в частности на АЗС. На текущий момент ТСО успешно продолжает внедряться для ускорения процесса оплаты.

ТСО представляет собой довольно простое устройство, позволяющее принимать денежные средства для оплаты различного рода услуг, наиболее популярными из которых являются плата за мобильную связь, Интернет и погашение кредитов. Устройства выполнены в форме металлических стоек, оснащённых повышенной защитой от физических взломов, что объясняется режимом их эксплуатации. Они имеют один или два экрана невысокого разрешения, купюроприемник и принтер для печати чеков. Для потребителя терминал – это, прежде всего - удобство, то достижение техники, которое позволяет пополнить счет мобильного телефона или оплатить заказ быстро и удобно.

Устройство ТСО предполагает разработку и установку программного обеспечения, заточенного под нужды конкретной АЗС или сети. Целью данной ВКР является создание ПО, которое будет обслуживать ТСО.

Рассмотрим взаимодействие пользователя с ТСО, расположенным на АЗС. ППО, установленное на ТСО выполняет функции приёма платежей клиентов, желающих

приобрести нефтепродукт (НП) или сопутствующие товары на АЗС. В сеансе работы пользователя с ТСО ППО часто обменивается данными со сторонними сервисами, для извлечения актуальной информации о ценах и видах НП, а также доступных топливораздаточных колонках (ТРК) из БД. Таким образом, для полноценной работы всей программной системы в целом можно выделить 3 основных компонента: графический интерфейс для ТСО (GUI), протокол обмена данными, сервис с БД.

Целью данной работы является определение методов подхода к реализации программного обеспечения, анализ функциональности, разработка архитектуры и пример последующей реализации модулей информационной системы для терминалов саобслуживания автозаправочной станции (АЗС).

Раздел 1. Постановка задачи

1.1 Анализ требований заказчика на разрабатываемое программное обеспечение

ТСО состоит из двух частей:

- клиентский GUI,
- сервисный GUI.

Первый предназначен для работы клиента с терминалом, второй – кассира или другого работника АЗС. В рамках ВКР рассматривается проектирование и разработка клиентского GUI.

В рамках ТЗ заказчик предоставил альбом экранных форм (пункт 1.1.1 и приложение А), который состоит из скриншотов, требуемых к реализации экранов. На этих картинках детально отображается взаимное расположение желаемых кнопок, надписей и изображений. Также были представлены функциональные требования (возможности) разрабатываемого ПО.

Не смотря на подробно описанные требования заказчика относительно прототипа интерфейса ИС, вопросы разработки экранных форм (подбор цветов элементов интерфейса, размеров, шрифтов) не являются основными и не рассматриваются в ВКР. Другими словами, на данный момент нет необходимости доводить интерфейс приложения до максимальной схожести с альбомом экранных форм. Этим в дальнейшем будет заниматься отдел Frontend разработки.

Значимыми же сейчас являются задачи реализации функциональных требований (пункт 1.1.1) к ИС. Ключевыми подзадачами в данном случае будут являться:

- Моделирование бизнес-процессов и бизнес-логики взаимодействия пользователя с интерфейсом ТСО (подраздел 3.2);
- Выбор архитектуры ИС (подраздел 2.1);
- Проектирование и реализация компонентов ИС (разделы 3 и 4).

Также отдельным пунктом рассматривается работа с данными, которые являются ключевыми для обработки и хранения информации о заказах НП, а также актуальной информации о видах НП и ценах на них. Более подробно требования по работе с данными указаны в пункте 1.1.2.

Для уменьшения трудностей в случае возможного расширения проекта необходимо уделить особое внимание деталям бизнес-логики в проекте. Под этим подразумевается рассмотрение последовательности процессов в системе в зависимости от действий пользователя, а также сбор и хранение необходимой информации во время сеанса клиента.

Важно рассмотреть все возможные случаи и сценарии, чтобы обеспечить правильное функционирование ТСО в любом состоянии, независимо от аппаратных поломок и сбоев в работе.

1.1.1 Функциональные возможности клиентского GUI

Функциональные требования – требования к разрабатываемой системе, которые определяют её назначение и основные возможности. По сути, это и есть поставленные задачи перед разработчиками, это то, что ожидает получить заказчик от приложения на финальных этапах жизненного цикла ПО.

Далее в данном пункте ВКР представлены функциональные требования заказчика в текстовом виде, именно в таком формате они были изложены в качестве вводных данных, на основе которых требуется разработать ИС для работы на ТСО.

Для того, чтобы исключить размытость требований и чётко понять акценты заказчика по отношению к разрабатываемому ПО для ТСО был предоставлен альбом экранных форм. Альбом экранных форм составляет набор скриншотов экранов ожидаемого приложения. Эти скриншоты определяют “скелет” экранов графической программы, служат прототипом GUI.

Так, рисунок 1.1 представляет из себя возможную версию Главного экрана приложения. Основная задача для этого экрана заключается в информировании пользователя о неполадках на ТСО (рисунок 1.2) (отказ работы сканера, отказ работы валидатора, отказ работы ТУ, закончилась бумага, закончилась краска, нет связи с банковским процессингом, нет связи с топливным процессингом или нет связи с БД) и формирования возможностей взаимодействия пользователя с приложением (например, отображение не всех возможных кнопок, а лишь части, в зависимости от текущего состояния терминала). Так, в случаях полностью функционирующего ТСО надпись на главном экране отсутствует вовсе. Если же отсутствует связь с одним из процессингов (топливным или банковским), то это наложит ограничения на оплату топливной и банковской картами соответственно. Об этом должно быть сообщено в виде надписи на информационном поле: “В настоящее время оплата топливной картой недоступна” или “В настоящее время оплата банковской картой недоступна”. Однако в вышеуказанных случаях остаётся возможность оплаты наличными или другой картой. Если же нельзя оплатить заказ ничем (состояние ТСО таково, что не работает купюроприёмник и POS-терминал), то отображается лишь кнопка “Информация”, а надпись оповещает “В настоящее время заправка не осуществляется”. Более подробно бизнес-логика формирования надписи на Главном экране рассмотрена на рисунке Б.1 приложения Б.1.

Также главный экран содержит информацию о текущих дате и времени в формате “ДД.ММ.ГГГГ ЧЧ:ММ”, она расположена в левом верхнем углу. Нижняя полоска предназначена для отображения номера ТСО, адреса АЗС и телефона горячей линии в случае некорректной работы терминала.



Рисунок 1.1 – Главный экран в начальном состоянии, не ограничивающем заправку на ТСО



Рисунок 1.2 – Главный экран в ограниченном состоянии

Описанные ранее текстовые элементы не реагируют на действия пользователи, они лишь отображают необходимую для правильной работы приложения информацию пользователю. Такие элементы называются “*Информационными*”. Ниже представлены информационные элементы Главного экрана:

- Информационные элементы:
 - Дата в формате ДД.ММ.ГГГГ;
 - Время в формате ЧЧ:ММ;
 - Информационная строка с № ТСО, Адресом ТСО, Телефоном службы поддержки;
 - Информационное поле об ограничениях работы с ТСО, в котором указаны основные ошибки, возникшие в процессе работы приложения.

Помимо информационных имеются также “*Управляющие элементы*”. Данные объекты на экране позволяют пользователю взаимодействовать с программой: переходить на последующие экраны, делать выбор в заказе, вводить необходимую информацию.

- Управляющие элементы:
 - Кнопка [Заправка] для перехода в режим заказа НП;
 - Кнопка [Печать чека/Перевод сдачи] для перехода в режим печати чека и распоряжения сдачей;
 - Кнопка [Информация] для перехода к ознакомлению с офертой предоставления услуг.

Всего в ИС планируется разработать 14 экранов. Для удобства разделим их на 3 непересекающиеся группы. Первая группа из 9 экранов служит для основной функции приложения – приёма и обработки заказа НП:

1. Главный экран (описан выше);
2. Информационный экран (рисунок А.1 в приложении А) служит для информирования клиента о событиях, происходящих на ТСО и имеющих отношение к клиенту, а также для индикации работы системы при проведении внутренних процессов. Наглядным примером является отображение надписи “Загрузка” в качестве поясняющего процесс сообщения клиенту. Также данный экран должен отображаться после оформления заказа НП или Других услуг (вторая группа из 4 экранов ниже) и полностью введённой информации;
3. Экран ошибок (рисунок А.2 в приложении А) отображает надпись-сообщение об ошибке. Со всеми возможными ошибками можно ознакомиться в рисунке 3.13;
4. Экран выбора ТРК (рисунок А.3 в приложении А) открывается при нажатии кнопки [Заправка] на Главном экране и отображает имеющиеся колонки (ТРК) на АЗС с их актуальными статусами (свободна, занята, недоступна) на данный момент;
5. Экран выбора НП (пистолета) (рисунок А.4 в приложении А) открывается после выбора ТРК на Экране выбора ТРК и отображает имеющиеся виды НП (92, 95, 98 бензин, дизель) на АЗС с их актуальными ценами за литр;
6. Экран выбора способа оплаты (рисунок А.5 в приложении А) открывается после выбора НП и отображает возможные виды оплаты (банковской, топливной картой или наличными) в зависимости от текущего состояния (ограничений) ТСО;
7. Экран ввода реквизитов клиента (рисунок А.6 в приложении А) открывается после выбора способа оплаты и служит формой для ввода номера телефона и/или адреса электронной почты. Ввод данной информации не является обязательным. При желании его можно пропустить;
8. Экран оплат (наличные) (рисунок А.7 в приложении А) открывается после Экрана ввода реквизитов клиента (в случае выбора способа оплаты наличными на Экране выбора способа оплаты) и отображает количество принятых купюр купюроприёмником, а также общую внесённую сумму денег в рублях. Поле “Ориентировочное кол-во литров” должен пересчитывать нужный объём в соответствии с ценой выбранного НП на Экране выбранного НП;

9. Экран оплат (БК/ТК) (рисунок А.8 в приложении А) открывается после Экрана ввода реквизитов клиента (в случае выбора способа оплаты банковской или топливной картой на Экране выбора способа оплаты) и отображает количество введенных рублей (сумма вводится с помощью виртуальной клавиатуры). Поле “Ввести литры” должно пересчитывать нужный объем в соответствии с ценой выбранного НП на Экране выбранного НП. Должна быть предусмотрена возможность ввода объема в литрах с клавиатуры и пересчета требуемой суммы в рублях.

Вторая группа из 4 экранов, описанных далее расширяет функционал ПО для ТСО до возможности перевода остатка суммы денег, в случае недолива топлива. Перевод можно осуществить по номеру телефона или на электронный кошелек. Данная услуга в ИС именуется “Другая услуга”.

1. Экран сканирования ШК (рисунок А.9 в приложении А) открывается при нажатии кнопки [Печать чека/Перевод сдачи] на «Главном экране» и выполняет функцию формы ввода клиентом штрих-кода с чека, который выдаётся уже после налива НП. До налива выдаётся *квитанция* об оплате, которая просто фиксирует принятый заказ. Чек же хранит информацию о недоливе, который мог произойти в процессе выдачи оплаченного объема НП. Более подробно БП выдачи ценных бумаг клиенту в процессе оплаты НП изображён на средней ветви в рисунке Б.2 приложения Б.1;
2. Экран перевода сдачи (выбор услуги) (рисунок А.10 в приложении А) открывается после Экрана сканирования ШК (в случае если ШК введенный на Экране сканирования ШК действителен и по тому заказу есть остаток). Данный экран даёт возможность перевести средства на номер телефона или веб-кошелек;
3. Экран перевода сдачи (по номеру телефона) (рисунок А.11 в приложении А) открывается после выбора сотового оператора на Экране перевода сдачи и служит формой для ввода номера мобильного телефона, на счёт которого будет переведена вся оставшаяся сумма остатка;
4. Экран перевода сдачи (на кошелек Элекснет) (рисунок А.12 в приложении А) открывается после выбора электронного кошелька на Экране перевода сдачи и служит формой для ввода номера электронного кошелька, на счёт которого будет переведена вся оставшаяся сумма остатка.

Третья группа состоит всего из 1 экрана и доступна всегда вне зависимости от текущего состояния (ограничений) ТСО. Экран информации (рисунок А.13 в приложении А)

отображается после нажатия кнопки [Информация] на «Главном экране» и даёт возможность пользователю ознакомиться с юридическими документами (офертой и публичной офертой), отражающих условия договора между покупателем и АЗС.

Более подробно об информационных и управляющих элементах соответствующих экранов можно прочитать в приложении А.

1.1.2 Хранение и обработка данных

В качестве средства для хранения информации заказчик указал реляционную БД. В список необходимых для хранения данных входят:

- Данные для формирования заказа:
 - Виды НП и актуальные цены на них;
 - Имеющиеся ТРК с их актуальными состояниями (занята, свободна, недоступна).
- Данные для сохранения заказа:
 - № ТСО;
 - № заказа;
 - Время и дата заказа;
 - ТРК № n;
 - НП № n;
 - Объём литров;
 - Сумма в рублях.
- Данные для логирования (ограничения ТСО):
 - Отказ работы сканера;
 - Отказ работы валидатора;
 - Отказ работы ТУ;
 - Закончилась бумага;
 - Закончилась краска.

Большим преимуществом будет продумать возможность поиска заказов по ШК в БД, для быстрой выдачи сдачи и осуществления Других услуг (пункт 1.1.1 Экран сканирования ШК).

Из вышеперечисленных требований по работе с данными становятся ясны основные подзадачи:

- Отбор данных, используемых при работе ТСО (подраздел 3.1);
- Подобрать протокол передачи данных (подраздел 2.2);

- Спроектировать БД (подраздел 3.3);
- Разработать алгоритмы обработки и хранения данных (подраздел 3.4).

1.1.3 Требования к ОС

Требования заказчика к операционной системе (ОС) были однозначны в пользу Astra Linux [1]. Причинами являлось то, что данная ОС основана на базе ядра Linux и была создана для всесторонней защиты информации и разработки защищённых автоматизированных систем. В российских силовых ведомствах, спецслужбах и государственных органах она широко востребована, т.к. обеспечивает степень защиты обрабатываемой информации до уровня государственной тайны «особой важности» включительно. Сертифицирована в системах сертификации средств защиты информации Минобороны, ФСТЭК и ФСБ России. Единый реестр российских программ Минкомсвязи России включил её в свой список.

Иными словами, Astra Linux основана на классе ОС семейства Debian и является полусвободной (без права декомпиляции). Первое означает, что в качестве средств для разработки можно выбирать те, которые поддерживают разработку «под Linux». Например, можно выбирать фреймворки и библиотеки, адаптированные под Linux разработку. Второе является большим плюсом в экономической стороне вопроса.

Раздел 2. Описание используемых технологий

2.1 Выбор архитектуры ИС

2.1.1 Сервис-ориентированная архитектура (микросервисная архитектура)

Сервис-ориентированная архитектура [2] – это способ организации программного приложения, при котором его функциональность представляется в виде набора сервисов. Сервис-ориентированная архитектура является одним из шагов для создания промежуточного программного обеспечения (middleware). Сервисно-ориентированная архитектура позволяет создавать распределённое программное обеспечение, состоящее из набора независимых сервисов. Благодаря сервисно-ориентированной архитектуре могут создаваться приложения, являющиеся композицией нескольких сервисов.

Описание веб-сервиса включает его название, расположение и требования к осуществлению процесса обмена данными.

Веб-сервис – программный компонент, идентифицируемый веб-адресом и выполняющие операции, продекларированные в поддерживаемом им интерфейсе. Другие программные компоненты взаимодействуют с веб-сервисами посредством обмена сообщениями.

Веб-сервисы в настоящее время становятся стандартом для интеграции распределённых систем программных приложений. Создание программных приложений, которые способны использовать веб-сервисы, недостаточно для полной поддержки бизнес-процессов. Организация взаимодействующих друг с другом веб-сервисов в сети позволяет реализовывать сложные рабочие процессы.

Характеристики сервисов:

- Слабая связность. Сервисы обычно независимы друг от друга и обладают минимумом информации о других сервисах;
- Автономность. Каждый сервис самодостаточен и способен к независимому функционированию (кроме управляющего сервиса сети веб-сервисов);
- Абстракция. Сервис предоставляет клиенту только интерфейс. Внутренние детали реализации скрыты от потребителя;
- Отсутствие состояния. Сервис не хранит состояние и не учитывает историю предыдущих обращений к нему;
- Композиция. Сервисы спроектированы с возможностью быть объединёнными в сеть веб-сервисов. В такой сети веб-сервисов всегда существует управляющий сервис, координирующий работу остальных сервисов;

- Обнаружение. Сервис должен обладать описанием. После обработки описания существует возможность обратиться к сервису без дополнительной настройки и конфигурации;
- Отсутствие идентичности. При обращении к разным экземплярам сервиса с одним контрактом результат не зависит от того, к какому именно экземпляру произошло обращение;
- Обращение через интерфейс. Веб-сервисы поддерживают интерфейсы с определенным перечнем операций.

Распространены следующие два типа веб-сервисов:

- SOAP веб-сервисы,
- REST-сервисы.

2.1.2 Монолитная архитектура

Приложение, в основе которого лежит монолитная архитектура (монолит) [3] чаще всего состоит из одного модуля, в котором расположены все компоненты приложения.

- Основными достоинствами монолитной архитектуры являются:
 - Простота реализации. Не требуется тратить время на продумывание межпроцессного взаимодействия и хитрой архитектуры – всё помещается в одном модуле;
 - Удобность развёртывания – достаточно запустить один скрипт, который загрузит модуль и запустит приложение.
- Из недостатков можно выделить:
 - Заметное усложнение разработки с ростом проекта. Сильная связанность между различными компонентами программы не позволяет быстро вносить изменения. Большинство изменений в коде потребует внесения корректив в смежных местах, которые используют эти компоненты;
 - Сложности при масштабировании. В монолитной архитектуре не получится расширить лишь часть компонентов. Все части системы необходимо будет масштабировать одновременно.

Таким образом, вышеописанный анализ оказывается в пользу микросервисной архитектуры. Она обеспечивает слабую связанность между сервисами, что позволяет разрабатывать различные модули системы по отдельности. Чёткие границы между сервисами дают возможность с лёгкостью заменить устаревший компонент системы на более подходящий, использующий новые технологии. При этом сам процесс разработки замедляется незначительно. Т.к. актуальность моей ВКР является вкладом в ИТ сообщество

и включает ускорение разработки новых программ, а также расширение клиентского сервиса, то возможность комбинирования различных модулей нескольких программ для сборки готового решения является особым преимуществом.

Ещё микросервисная архитектура лучше справляется с локальными сбоями сервисов. Так, хорошо распределённая система справится с выходом из строя одного сервиса, в то время как для монолита это станет серьёзной трудностью.

2.2 Анализ различных протоколов обмена данными

2.2.1 SOAP веб-сервисы

Протокол SOAP – протокол обмена структурированными сообщениями в распределённой вычислительной среде. SOAP-сообщение представляет собой XML-документ. Сообщение состоит из трех основных элементов: конверт (SOAP Envelope), заголовок (SOAP Header) и тело (SOAP Body).

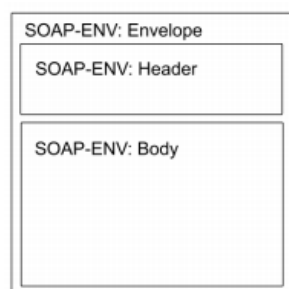


Рисунок 2.1 – SOAP-конверт.

Заголовок – это дочерний элемент конверта. Не обязательный.

Элемент *Body* обязательно записывается сразу за элементом *Header*, если он есть в сообщении, или первым в SOAP-сообщении, если заголовок отсутствует. В элемент *Body* можно вложить произвольные элементы, спецификация никак не определяет их структуру. Определен только один стандартный элемент, который может быть в теле сообщения - *Fault*, содержащий сообщение об ошибке. Протокол SOAP не различает вызов процедуры и ответ на него, а просто определяет формат послания (message) в виде документа XML. Послание может содержать вызов процедуры, ответ на него, запрос на выполнение каких-то других действий или просто текст. Спецификацию SOAP не интересует содержимое послания, она задает только его оформление. SOAP основан на языке XML и расширяет некоторый протокол прикладного уровня — HTTP, FTP, SMTP и т.д. Как правило чаще всего используется HTTP. Вместо использования HTTP для запроса HTML страницы, которая будет показана в браузере, SOAP отправляет посредством HTTP-запроса XML-сообщение и получает результат в HTTP отклике. Для правильной обработки XML-сообщения процесс-«слушатель» HTTP (напр. Apache или Microsoft IIS) должен предоставить SOAP процессор, или, другими словами, должен иметь возможность

обработать XML.

2.2.2 REST веб-сервисы

REST — метод взаимодействия компонентов приложения с использованием протокола HTTP для вызова процедуры. При этом необходимые данные передаются в качестве параметров запроса. Этот способ является альтернативой более сложным методам, таким как SOAP, CORBA и RPC. Передача состояния представления введен и определен в 2000 году Роем Филдингом в его кандидатской диссертации «Архитектурные стили и проектирование архитектур сетевого программного обеспечения». Веб-сервисы REST являются веб-сервисами, реализуемые с использованием HTTP и принципов REST. Как правило, Web-сервис RESTful определяет URI основного ресурса, поддерживаемые MIME-типы представления/ответа и операции.

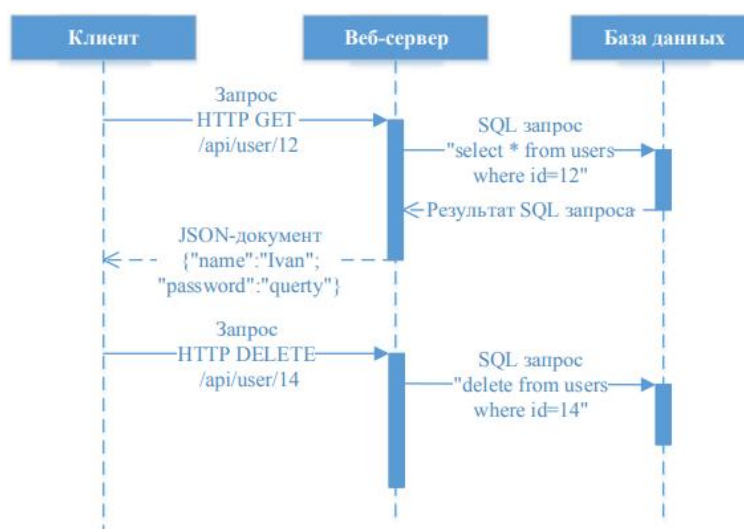


Рисунок 2.2 – REST веб-сервис

Ресурс может являться практически любым понятным и значимым адресуемым объектом. Представление ресурса — это обычно документ, отражающий текущее или требуемое состояние ресурса. Ресурсы обычно представляются документами в форматах XML или JSON. REST и горизонтальный подход. Стратегия, опирающаяся на горизонтальный подход к протоколам, наиболее радикальна. Слово «горизонтальный» означает в данном случае сохранение существующего уровня, без выстраивания уровней поверх него. Предполагается отказаться от разработки новых протоколов, а использовать несколько хорошо проверенных, считая, что для работы с объектами вполне достаточно уметь выполнять четыре типа действий: создание (Creation), восстановление (Retrieval), изменение (Update) и уничтожение (Destruction). Из этих действий получается так называемый «шаблон проектирования» CRUD. Протокол Hypertext Transfer Protocol определяет методы GET/PUT/POST/DELETE, которые и реализуют шаблон CRUD. Аббревиатура (шаблон) CRUD обозначает перечень основных операций с объектом: create

(создать), read (считать, загрузить), update (обновить, изменить, отредактировать) и delete (удалить).

Разработчики SOAP веб-сервисов создают свой собственный перечень имен существительных и глаголов (например getUsers(), savePurchaseOrder(...)) для обозначения операций веб-сервиса. В этом смысле SOAP реализуют сервисный принцип работы, в соответствии с которым главную роль при взаимодействии со службами играют их методы. В основе архитектуры REST веб-сервисов лежит ресурсный (объектный) подход. REST веб-сервисы разрабатываются согласно следующим принципам:

1. Возвращайте любые данные по их идентификатору;
2. Use standard methods — используйте стандарты, имеется в виду, экономьте свои силы и деньги заказчика, используйте стандартные методы HTTP;
3. Одни и те же данные можно вернуть различных форматах. Например, в XML или JSON для последующей программной обработки;
4. Передача данных без сохранения состояния. При обращении к REST-сервису не учитываются результаты ранее выполненных операций. REST приложение не сохраняет никакого состояния сессии на стороне сервера. Вся информация, необходимая для выполнения запроса, передается в самом запросе.

Таблица 2.1 – Сравнение REST и SOAP веб-сервисов ^[2]

Критерий	REST	SOAP
Назначение	Инкапсулирует бизнес-логику	Доступ к ресурсам/данным
Методология разработки	Объектно-ориентированная (сервисно-ориентированная)	Ресурсно-ориентированная
Независимость от языка программирования	+	+
Независимость от платформы	+	+
Независимость от транспортного протокола	+	- (только HTTP)
Стандартизирован	+	-
Необходимость использования специальных средств разработки	+	-
Протокол сообщений	XML	XML, JSON или любой типа MIME

Размер сообщений	Большой из-за наличия служебных данных	Сравнительно небольшой
Производительность	Ниже	Выше

Исходя из сравнения выше, становится ясно, что выбор REST протокола был сделан неслучайно. REST-API более прост для разработки протокола взаимодействия и обеспечивает большую производительность, нежели SOAP. Несмотря на отсутствие строгих стандартов, «шаблон проектирования» CRUD даёт возможность REST вполне успешно конкурировать со стандартизированным SOAP протоколом.

2.3 Выбор технологий для разработки

2.3.1 ЯП и фреймворки для WEB-разработки

Python длительное время не теряет своей актуальности и на данный момент развился в динамичный, гибкий и очень мощный язык программирования. Многие разработчики предпочитают использовать для работы именно его, а не такие традиционные варианты, как C++ и Java. Веб-разработка — также является сильной стороной этого языка.

Вот несколько примеров [4] ведущих сайтов, использующих ЯП Python в своей работе:

- Instagram — очень популярная социальная сеть с ежедневной посещаемостью более 4 миллионов пользователей. Python — основная технология, которая использовалась при ее создании.
- Spotify — это приложение задает тон в индустрии стриминга музыки. Оно создано с использованием Python.
- Disqus — это плагин для комментирования записей на сайтах. Ежемесячно с его помощью оставляют около 50 миллионов комментариев в сети. Основной ЯП, использовавшийся при создании этого плагина — Python.

Если посмотреть на приведенные примеры, становится ясно, что Python прекрасно подходит для создания самых разнообразных веб-проектов.

Но, как и у любого ЯП, у Python есть и свои недостатки, не позволяющие применять его в некоторых проектах.

- Преимущества использования Python для веб-разработки:
 - простота использования,
 - легкость изучения,
 - отлично подходит для визуализации данных,

- легкость чтения,
- несравненная гибкость,
- асинхронное программирование;
- минусы использования Python в веб-разработке:
 - ограничения скорости,
 - проблемы с потоками.

По большому счету, преимущества языка Python перевешивают его недостатки. И он, благодаря своей непревзойденной гибкости, простоте использования и модульности, для многих разработчиков остается самым предпочтительным вариантом выбора.

2.3.2 Анализ особенностей фреймворков Django и DRF для разработки веб-служб REST

Системная архитектура — это организация и структура распределения элементов информационной системы, то есть это то, как логически распределена на наша система и как эти элементы взаимодействуют между собой. Если не следовать какой-то архитектуре, то в дальнейшем будет очень сложно вносить какие-то изменения и дорабатывать этот код. Поэтому люди начали придумывать некие паттерны разработки.

Собственно, один из них мы и рассмотрим — MVC (Model-View-Controller) [5]. Это схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента. Это сделано для того, чтобы если поменялся какой-то из компонентов, мы могли его поменять, и это изменение не затронуло Model-View. Давайте рассмотрим, как происходит процесс обращения к клиенту.



Рисунок 2.3 – паттерн MVC

Клиент делает запрос к нашему приложению. Он попадает в контроллер. Контроллер как-то модифицирует данные с помощью модели. После этого модель сохраняет эти данные в базу. После этого модель предоставляет эти данные view.

View как-то формирует ответ и отдает этот ответ клиенту. Собственно, Controller. Контроллер отвечает за бизнес-логику, контроллер использует модели, для того чтобы как-то изменять, или добавлять, или удалять данные и представления, или view, чтобы их отображать. Собственно, в контроллере сосредоточена вся бизнес-логика нашей системы. View, или представление, формирует ответ в конкретном формате, то есть во View попадают некие данные, и он их как-то отображает. Модель отвечает за данные и правила их обработки, то есть все, что ты делаем с данными, мы делаем в моделях.

Именно этот паттерн используют фреймворки Django и DRF, используемые в данной работе.

2.3.3 ЯП и фреймворки для разработки GUI приложения

GUI - graphical user interface, графический интерфейс пользователя позволяет пользователю взаимодействовать с программой без лишних усилий. При разработке графического приложений решаются вопросы интуитивной понятности интерфейса (взаимного расположения кнопок и иных виджетов).

Помимо дизайна требуется задумываться о скорости отрисовки графики и отображения элементов окна приложения. В случаях простых приложений, которые не являются высоконагруженными и не используют большое количество ресурсов компьютера, с этим не возникает особых проблем. Для таких приложений оказываются эффективными высокоуровневые языки типа Python, а удачными фреймворками оказываются PyQt, Kyvi и WxPython.

Когда же система становится более нагруженной и начинает обрабатывать большое количество запросов, становится распределённой, то высокоуровневые ЯП начинают проигрывать более низкоуровневым в скорости, производительности и управлении памятью. Здесь в игру вступают Java и C++.

Выбирая между высокоуровневым и низкоуровневым ЯП, стоит учитывать сроки и скорость разработки в рамках которых требуется реализовать ИС. Так, слишком большая и нагруженная ИС, написанная на C++ потребует большего времени на разработку, нежели легковесный прототип ИС, реализованный с помощью Python.

Получается, что в силу ограниченности временных ресурсов и отсутствии критических нагрузок на систему разумнее использовать более высокоуровневый ЯП – Python.

2.3.4 Анализ особенностей фреймворка PyQt5 для разработки GUI приложения

Qt [6] – фреймворк для разработки кроссплатформенного, написанный на языке C++. Для ЯП Python существует библиотека-обёртка PyQt [7], которая позволяет сохранять преимущества быстрого действия и надёжности C++-го фреймворка, совмещая при этом

простоту синтаксиса Python и быстроту разработки.

Qt содержит более 45 виджетов, основными из которых являются: QPushButton (кнопка), QLabel (простой текст или картинка), QLineEdit (поле ввода текстовой строки), QDoubleSpinBox (счётчик двойной точности для работы с десятичными числами). Компоновка (сохранение взаимного расположения элементов при изменении размеров окна) осуществляется с помощью менеджеров компоновки: QHBoxLayout – горизонтальная, QVBoxLayout – вертикальная, QGridLayout – табличная. Также Qt позволяет создавать собственные виджеты, тем самым обеспечивая гибкую разработку.

Кроссплатформенность является значимым преимуществом Qt. Осуществляется поддержка большого числа ОС, среди которых: Linux и другие UNIX-подобные ОС, Mac OS X и Windows. Этот плюс позволяет разработчикам не задумываться над особенностями конкретной ОС, а с писать и разворачивать свои приложения на разных ОС, без необходимости переписывания исходного кода.

Ещё одним немаловажным преимуществом PyQt является наличие бесплатной лицензии GPL.

Быстрая разработка достигается с помощью такого удобного инструмента как Qt Designer (Qt Creator). Он входит в состав фреймворка PyQt и является дизайнером графического интерфейса пользователя. Иными словами, этот инструмент позволяет программисту избавиться от монотонного программирования создания однотипных виджетов, составляющих основу любого окна в Qt. Вместо этого можно воспользоваться удобным интерфейсом Qt Designer'a и просто редактировать окно приложения. Утилита ruic позволяет легко создавать Python код из файлов Qt Designer'a. Именно это преимущество и делает фреймворк PyQt столь удобным для быстрого прототипирования графического интерфейса.

Для моей ВКР возможность создания быстрого прототипа является необходимостью, в силу требований заказчика (раздел 1). Заказчик конкретно указал, что создание “красивого” интерфейса не стоит рассматривать на данном этапе, этим в дальнейшем всерьёз будет заниматься отдел Frontend-разработки. Сейчас же нужно сфокусироваться на вопросах проектирования и работы с данными (раздел 3).

Таким образом, вышеописанный анализ особенностей фреймворка PyQt5 для разработки GUI приложения демонстрирует, что данный фреймворк подходит как нельзя лучше для реализации задач части быстрого создания графического интерфейса. Это является серьёзным аргументом в пользу выбора именно этого инструмента для разработки.

Раздел 3. Проектирование архитектуры компонентов ИС

3.1 Отбор данных, используемых при работе ТСО

Для корректных обработки и хранения необходимых данных, указанных заказчиком в подразделе 1.1.2 нас будут интересовать все данные используемые при работе ТСО. Отберём эти данные на основе информации из ТЗ (раздел 1).

Данные будем условно разделять на 2 потока: символом «←» обозначается поток данных из Backend'а ТСО в Frontend ТСО, символом «→» – поток данных из Frontend'а ТСО в Backend ТСО.

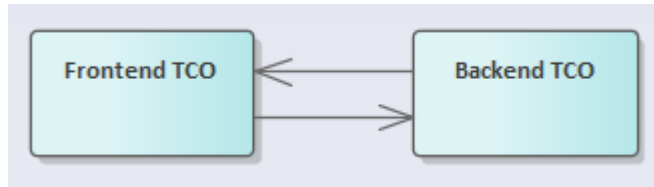


Рисунок 3.1 – Две категории данных в ТСО

- Ограничения ТСО:

Поток данных из Frontend'а ТСО в Backend ТСО «→»:

- Отказ работы сканера;
- Отказ работы валидатора;
- Отказ работы ТУ;
- Закончилась бумага;
- Закончилась краска;

Поток данных из Backend'а ТСО в Frontend ТСО «←»:

- Отпуск топлива временно невозможен;

- Данные по ТРК:

Поток данных из Backend'а ТСО в Frontend ТСО «←»:

- Состояние колонки;

- Данные по НП:

Поток данных из Backend'а ТСО в Frontend ТСО «←»:

- Вид НП;
- Цена НП;

- Данные по заказу:

Поток данных из Frontend'а ТСО в Backend ТСО «→»:

- Реквизиты клиента;

- Данные по заказу НП:

Поток данных из Frontend'а ТСО в Backend ТСО «→»:

- № ТСО;

- № заказа;
- Время и дата заказа;
- ТРК № n;
- НП № n;
- Объём литров;
- Сумма в рублях;
- Реквизиты клиента;
- Данные по считанному штрих-коду:

Поток данных из Frontend'а TCO в Backend TCO «→»:

 - ID штрих-кода;

Поток данных из Backend'а TCO в Frontend TCO «←»:

 - Ответ сервера;
- Данные по другим услугам:

Поток данных из Frontend'а TCO в Backend TCO «→»:

 - № TCO;
 - № заказа;
 - Время и дата заказа;
 - Выбор оператора № n;
 - Выбор электронного кошелька;
 - Сумма в рублях;

Поток данных из Backend'а TCO в Frontend TCO «←»:

 - Список доступных операторов и кошелька.

На основе отобранных данных реализованы HTTP процедуры обмена данными (подраздел 4.1) в соответствии с выбранным протоколом в подразделе 2.2.

3.2 Моделирование бизнес-процессов и бизнес-логики взаимодействия пользователя с интерфейсом TCO на основе ТЗ

Опорным этапом при проектировании ИС является моделирование её бизнес-логики. Бизнес-процессы, происходящие в бизнес-логике, позволяют детально рассмотреть всё, что происходит в системе. Жизненный цикл правильно смоделированной системы будет значительно дольше, чем той ИС, где этап моделирования отсутствовал или был рассмотрен поверхностно.

На основе информации из ТЗ, был изучен интерфейс TCO и отображена информация, которую использует приложение (подраздел 3.1) для правильной работы. Была смоделирована бизнес-логика взаимодействия пользователя с интерфейсом TCO.

Результаты моделирования представлены в виде диаграмм активности (Activity Diagram) UML ниже, а также в приложении Б.1.

Всего в ИС смоделировано 15 бизнес-процессов. БП можно условно разделить на 3 пересекающиеся категории. Первая наиболее обширная категория включает 14 БП. Они отображены на рисунках Б.2-Б.12 приложения Б.1 и 3.1, 3.2, 3.3 текущего подраздела. Суть данных бизнес-процессов сводится к осуществлению правильных переходов между экранами в зависимости от введенной пользователем информации и нажатых кнопок.

Для примера изучим рисунок 3.1. Рассматриваемый БП объясняет бизнес-логику Главного экрана при нажатии на различные кнопки на экране. Начальное состояние экрана готово к взаимодействию с пользователем. Была выбрана надпись заголовка окна или её отсутствие в зависимости от ограничений ТСО (рисунок Б.1 приложения Б.1).

Следующим этапом будет принятие решения об отображении необходимых кнопок. В случае, когда есть связь с БД, а также доступен хотя бы один вид оплаты (банковской/топливной картой, наличными) отображаются все кнопки. В противном случае отображается лишь кнопка “Информация”.

Нажатие на кнопку “Заправка” означает переход на Информационный экран и продолжение БП из рисунка Б.2 приложения Б.1 (верхняя ветвь).

Нажатие на кнопку “Печать чека/перевод сдачи” запускает процедуру “Другая услуга” (подраздел 1.1.1) и открывает Экран сканирования ШК (рисунок Б.12 приложения Б.1).

Нажатие на кнопку “Информация” открывает Экран информации (рисунок Б.6 приложения Б.1).

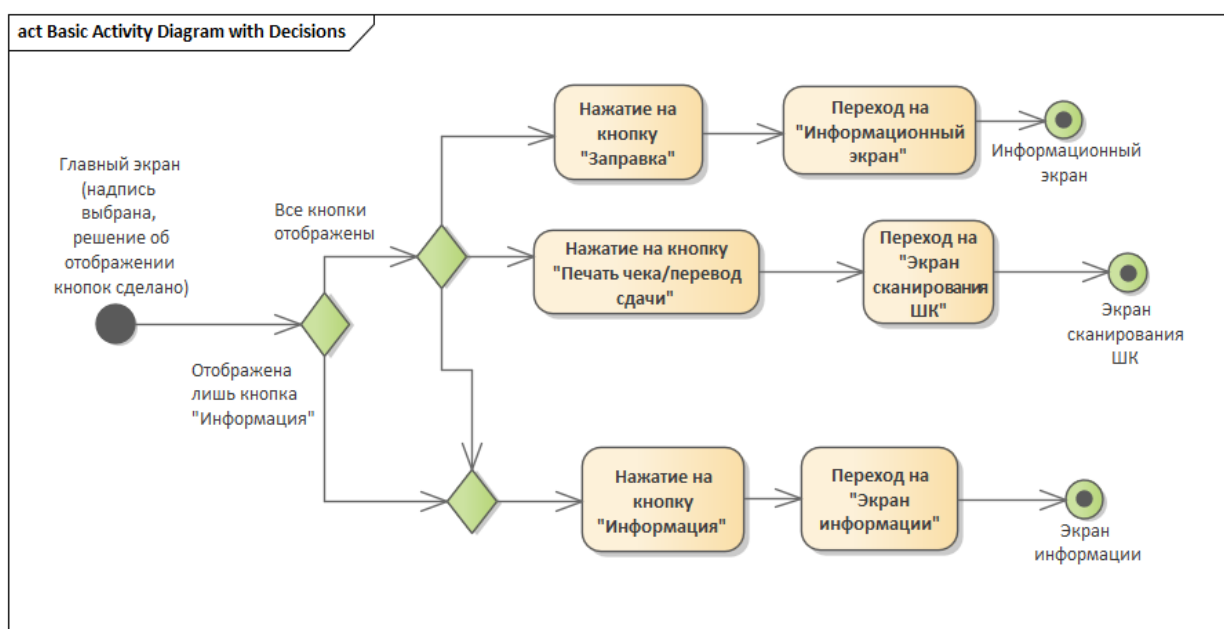


Рисунок 3.1 – Главный экран в состоянии, готовом ко взаимодействию с пользователем

Во второй категории состоит выделить 6 наиболее характерных БП, которые отвечают за формирование элементов графического интерфейса (надписей, кнопок, изображений, пересчёт счётчиков не введённой величины). Происходит это на основе накопленной информации с предыдущих экранов или информации, замкнутой на ТСО. Бизнес-логика данных БП продемонстрирована на рисунках Б.1, Б.4, Б.7, Б.8 приложения Б.1 и 3.2, 3.3 данного подраздела.

В качестве примера рассмотрим БП представленный на рисунке 3.2. На данной схеме отображена бизнес-логика Экрана оплат банковской или топливной картой (БК/ТК). Чтобы иметь визуальное представление об описанном экране см. рисунок А.8 в приложении А. После введения суммы заказа в рублях проверяется была ли превышена максимальная сумма заказа. Для возможности перевода на счёт мобильного телефона остатка в случае недолива топлива *максимальная сумма* заказа ограничена 15000 рублей. Если сумма не была превышена, то происходит пересчёт не объёма в литрах в соответствии с выбранным НП на Экране выбора НП (пистолета). Далее бизнес-процесс возвращается в начало.

В случае введения объёма в литрах сразу происходит пересчёт нужной суммы в рублях. И уже это полученное значение сравнивается с максимальной суммой. Если сумма не превышена бизнес-логика возвращается в начало. Для обеих форм ввода введение значений (суммы рублей или литров), превышающих максимум в рублях, означает возврат предыдущих допустимых значений в формы ввода.

Когда происходит нажатие пользователем на кнопку “Оплатить” происходит проверка, что значения счётчиков отличны от нуля. Если это является истиной осуществляется переход на Информационный экран, где далее продолжается бизнес-логика рисунка Б.2 приложения Б (нижняя ветвь). Когда же в счётчики не были внесены нужные суммы – ничего не происходит.

Особое внимание стоит уделить переходу с Экрана оплат (БК/ТК) на Главный экран. Это событие происходит в 2-х возможных сценариях. Первый запускается после нажатия на кнопку “Выход”. В этом случае происходит немедленный возврат. Второй же начинается после того как таймер в левом верхнем углу завершает отсчёт. В качестве времени допустимого нахождения на экране выбрано 120 секунд или 2 минуты. Этого времени вполне достаточно, чтобы осуществить требуемые действия на экране и продолжить бизнес-процесс.

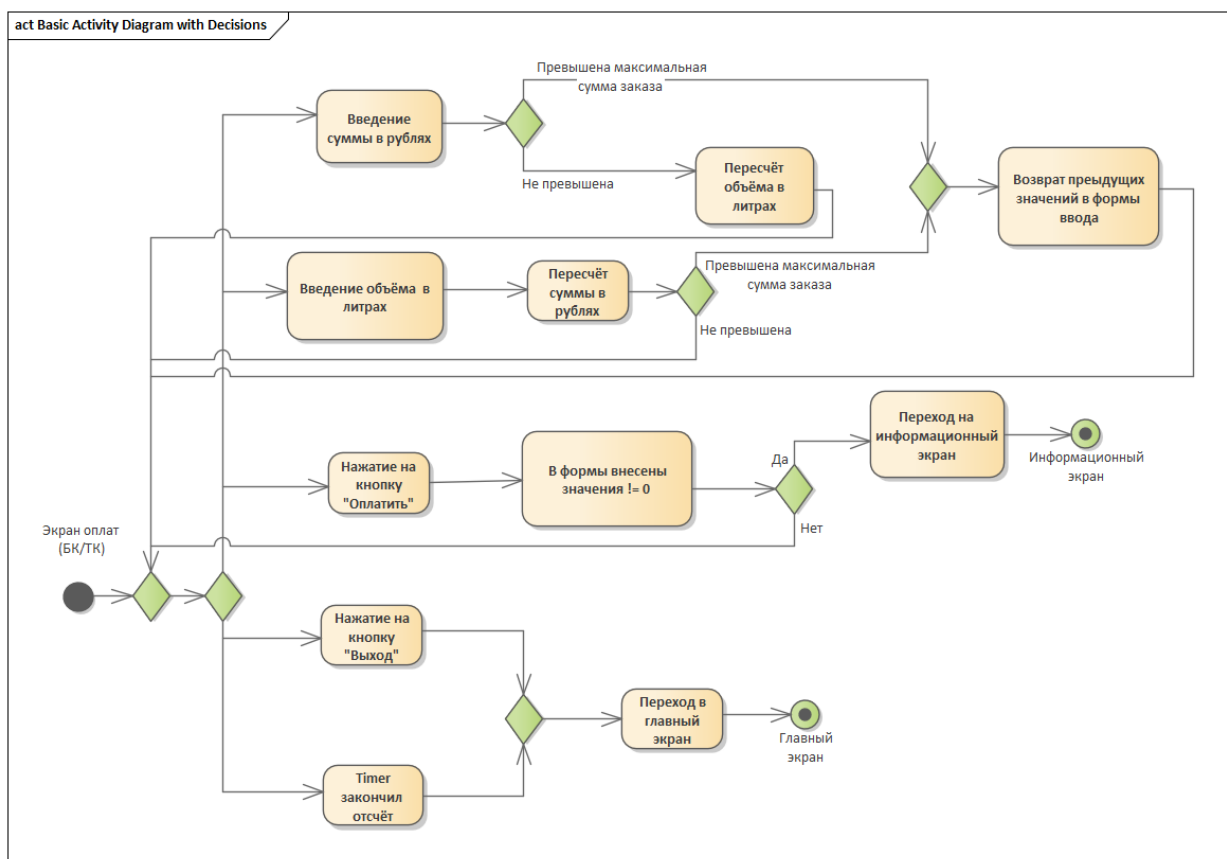


Рисунок 3.2 – Бизнес-логика экрана оплат (БК/ТК)

Переход на Главный экран в 2-х сценариях должен быть реализован не только на Экране оплат (БК/ТК), описанном выше. Тоже самое происходит и на рисунках Б.3-Б.12 из приложения Б.1, а также 3.2, 3.3 из данного подраздела.

Третья категория БП осуществляет отправку HTTP запроса (подраздел 4.1) на REST-сервер. С этими БП можно ознакомиться на рисунках Б.2, Б.4, Б.10-Б.12 приложения Б.1 и 3.3 данного подраздела.

Разберём пример из рисунка 3.3 (рисунок А.3 приложения А). Здесь представлена бизнес-логика экрана выбора ТРК после того как верхняя ветвь БП Информационного экрана была завершена кодом 200 (рисунок Б.2 приложения Б.1). Также на предыдущем экране был отправлен HTTP-запрос GET /pumps, возвращающий список ТРК с их актуальными состояниями. Именно эти элементы из списка нужно отобразить в виде кнопок на экране выбора ТРК.

Далее пользователь выбирает одну из колонок и если она оказывается недоступной, то ничего не происходит. Бизнес-логика возвращается в самое начало БП. Если же выбранная ТРК свободна/занята, то будет отправлен HTTP-запрос GET /petrols для получения информации по НП и ценам на них. Далее произойдёт переход на БП из рисунка Б.4 приложения Б.1 в случае возврата запросом статус-кода 200. Когда запрос возвращает код 503, то происходит переход на Экран ошибок (рисунок Б.8 приложение Б.1), где

отображается надпись “Нет связи с ТРК. Заправка невозможна!”.

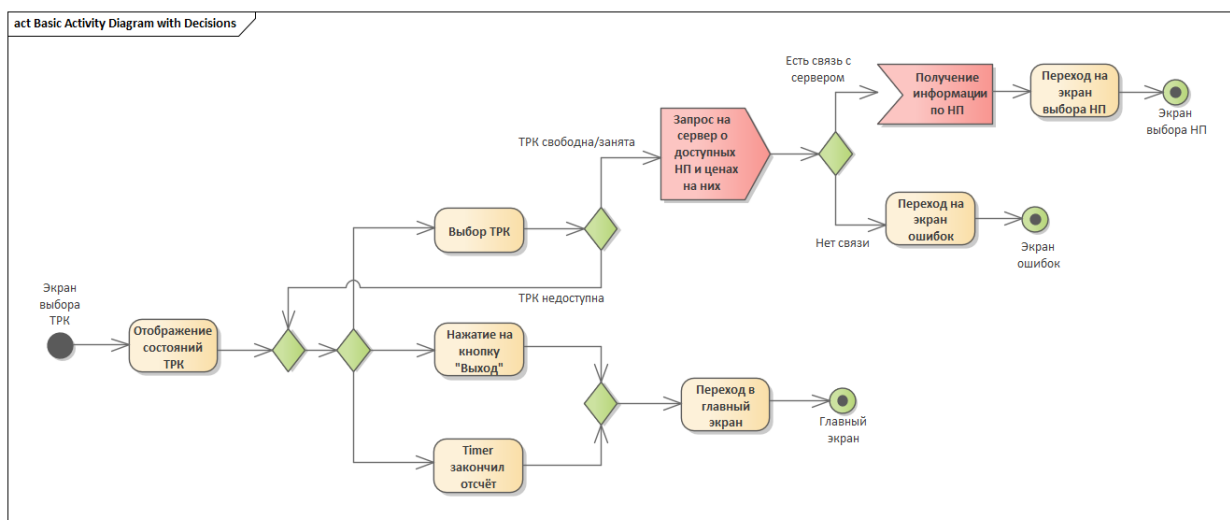


Рисунок 3.3 – Бизнес-логика экрана выбора ТРК

С оставшимися БП в ИС можно ознакомиться в приложении Б.1.

3.3 Проектирование БД REST-сервиса

В подразделе 4.1 представлена реализация протокола обмена данными между модулями системы, на основе бизнес-логики из подраздела 3.2. Необходимо чётко понимать, как стоит хранить эти данные REST-серверу.

В данном разделе представлена схема спроектированной базы данных.

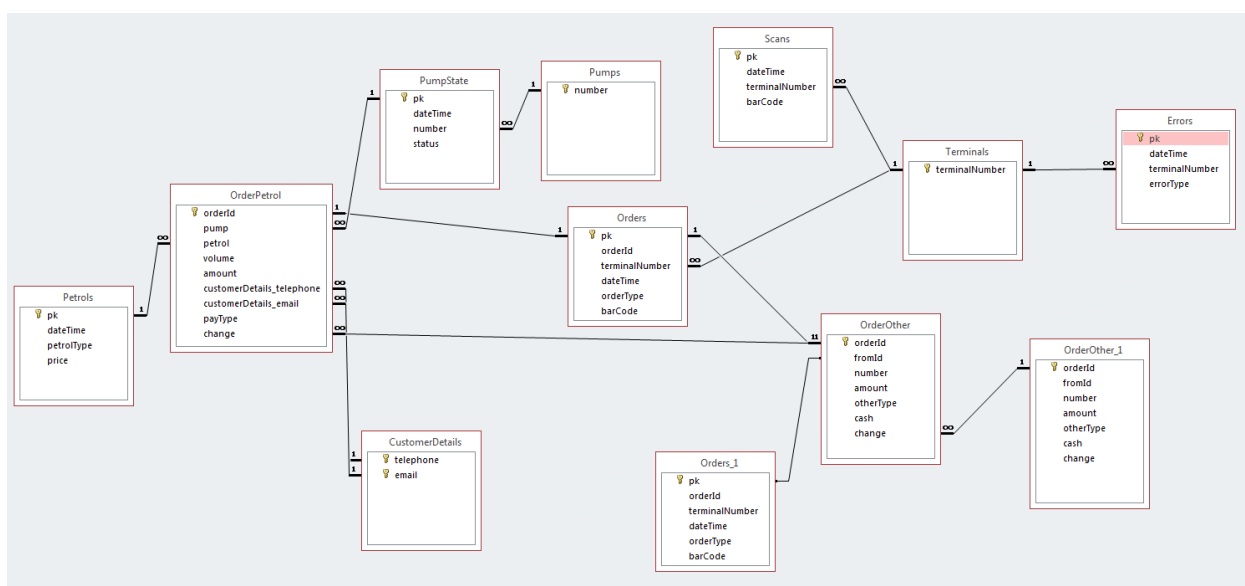


Рисунок 3.4 – Схема БД.

Всего в базе 10 таблиц. Большинство из таблиц представляют собой сущности с полями, которые просто дублируют поля JSON объектов из запросов (в таблице Petrols

хранятся объекты со схемой Petrol, CustomerDetails хранит одноимённый объект и т.д.). Однако часть таблиц создана для удобной связи между объектами. Например, таблица Orders хранит общую информацию о заказах OrderOther и OrderPetrol. В таблицах OrderOther и OrderPetrol введено поле change, отражающее была ли произведена выплата сдачи по данному заказу. По умолчанию значение поля NULL. В случае, если по заказу происходит выплата сдачи, то поле change принимает значение первичного ключа заказа OrderOther, который осуществлял этот перевод.

Между таблицами установлены различные виды связей. Связь один-ко-многим мы можем наблюдать между таблицами Terminals и Errors. Связь один-к-одному продемонстрирована между таблицами Orders и OrderOther. Также можно заметить неочевидную связь между двумя экземплярами одной сущности OrderOther. Это замыкание поля change на самой сущности, на случай множественного перевода сдачи.

3.4 Разработка алгоритмов обработки и хранения данных на стороне REST-сервиса

Для того чтобы REST-сервис быстро принимал запросы и отвечал на них, необходимо продумать алгоритмы обработки и хранения данных. Необходимо рассмотреть все случаи, которые могут возникнуть в процессе обработки HTTP-запросов.

Простейший из алгоритмов продемонстрирован ниже (рисунок 3.5). В нём акцент делается на наличии или отсутствии связи с сервером. В зависимости от этого отправляются различные статус коды (200 или 503).

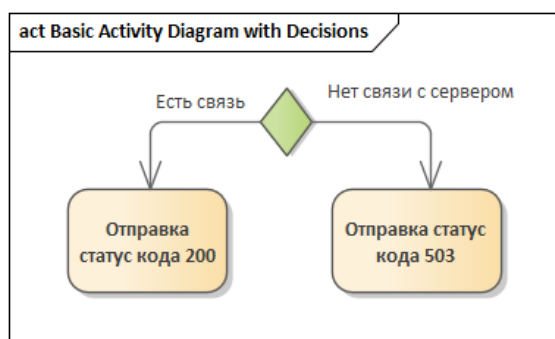


Рисунок 3.5 – Алгоритм обработки запроса GET /session

Одной из первостепенных задач в обработке запросов является задача проверки типов полей при сериализации в формат JSON и десериализации из формата JSON обратно в структуру данных. Так, в алгоритме на рисунке Б.13 приложения Б.2, поле errorList должно быть обязательно типа list, а terminalNumber – int. Запрос POST /session отправляет данные серверу о состоянии терминала. Для корректной работы системы необходимо знать

о слабых местах оборудования, которое установлено на АЗС.

Для правильного ответа на GET запрос часто бывает нужно обращаться к БД. В алгоритме на рисунке Б.14 приложения Б.2 формируется список актуальных цен на все виды НП, которые есть в БД.

Алгоритм на рисунке Б.15 приложения Б.2 похож на алгоритм из рисунка Б.14 приложения Б.2, однако здесь мы работаем с другой таблицей и формируем список ТРК с их состояниями на данный момент.

Алгоритм на рисунке Б.16 приложения Б.2 отвечает за приём данных, полученных в ходе процедуры обработки штрих-кода из чека об оплате товара. Нужно найти заказ по штрих-коду в базе и отправить клиенту сумму, доступную для снятия.

В предыдущем алгоритме использовались функции `isActive` и `countCash`. Первая проверяет найденный заказ на то, была ли сдача по нему уже выплачена или штрих-код ещё не использовался. Функция же `countCash` вычисляет остаток после перевода сдачи (возможен случай, что пользователь не станет переводить всю оставшуюся сумму за раз, а захочет сделать это за несколько операций). Алгоритмы функций отображены на рисунках Б.17 и Б.18 приложения Б.2 соответственно.

3.5 Проектирование модулей ИС

3.5.1 Проектирование архитектуры REST-сервиса в соответствии с паттерном MVC

В разделе 2.3.2 были описаны основные особенности архитектуры приложений, написанных с использованием фреймворков Django и DRF. Таким образом, для написания веб-сервиса нам потребуется реализовать классы моделей, сериализаторов и контроллеров.

Модели являются Python обёртками над базой данных. С помощью несложных методов классов и ООП стиля программирования мы имеем возможность с лёгкостью осуществлять нетривиальные SQL запросы. Все классы моделей в проекте наследуются от базового класса `models.Model` (рисунок Б.19 приложения Б.3).

Класс сериализатора `serializers.ModelSerializer` является потомком класса `serializers.Serializer` и является более высокоуровневым. Иногда (в 2 случаях на рисунке Б.21 приложения Б.3) его поведение полностью нас устраивало, однако значительно чаще (Рисунок Б.20 приложения Б.3) приходилось переопределять поведение более низкоуровневого класса `serializers.Serializer` для обеспечения необходимого полноценной сериализации сложных объектов, полученных из протокола REST-API.

Классы контроллеры, наследуются от класса `ApiView` и переопределяют стандартные HTTP-методы GET, POST и т.д. (Рисунок Б.22 приложения Б.3).

3.5.2 Проектирование архитектуры графического интерфейса (GUI) для ТСО

Данный пункт посвящён проектированию архитектуры графического модуля ИС. По большей части здесь обсуждаются вопросы подбора классов-инструментов, а также некоторые значимые детали реализации БП из подраздела 3.2.

Ранее в пункте 2.3.4 мы выбрали фреймворк PyQt5 для разработки графического интерфейса для ТСО. Этот выбор позволит нам не разрабатывать все окна приложения с нуля, а воспользоваться готовыми решениями других разработчиков. Продуманная система классов в PyQt5 определяет “каркас” – основу любого графического приложения.

Основные виджеты, которые используются в экранах являются классами из модуля QtWidgets. Это:

- QLabel – виджет для текста или картинки;
- QLineEdit – форма ввода текстовой строки;
- QPushButton – кнопка;
- QDoubleSpinBox – счётчик с точностью до второй цифры после запятой;
- QHBoxLayout, QVBoxLayout, QGridLayout – горизонтальный, вертикальный и сеточный компоновщики;
- QSpacerItem – виджет для выравнивания других виджетов.

Более детальный пример формирования окна с помощью виджетов представлен в пункте 4.2.2.

Для работы со временем используется класс QTimer из модуля QtCore. Во многих экранах, где используется таймер, в конструкторе вызывается метод, инициализирующий этот таймер – `init_timer()`.

В некоторых случаях необходимо описывать сложную бизнес-логику в отдельном классе Logic – наследника QObject модуля QtCore. Объект данного класса хранится в атрибуте `self.logic` экрана. Этот объект будет принадлежать отдельному потоку (класс QThread из модуля QtCore). Дополнительный поток необходим, дабы не нагружать основной графический поток и не мешать пользователю работать с экраном. Когда новый поток запускается, он испускает сигнал `started`, после чего вызывается связанный метод-слот `run` класса Logic, который реализует ту самую непростую бизнес-логику. Для упрощения программирования и минимизации дублирования кода инициализация класса Logic обёрнута в метод `init_logic`.

Раздел 4. Программная реализация компонентов ИС

В данном разделе представлены реализации HTTP-процедур обмена данными, REST-сервиса, а также графического интерфейса (GUI) для ТСО.

Ниже представлены листинги основных файлов, которые отражают обработку HTTP-запросов, особенности администрирования и работы с базой данных, а также логику работы приложения.

4.1 Реализация HTTP процедур обмена данными в соответствии с выбранным протоколом REST

В данном разделе приведено краткое пояснение к реализации процедур передачи данных в соответствии с основными принципами REST. В качестве среды для разработки REST процедур и схем передаваемых объектов была выбрана платформа Swagger. Она предоставляет широкий спектр возможности для удобной разработки REST-API.

Ниже (приложение В.1) представлены результаты реализации процедур взаимодействия.

Исходный код интерфейса написан с помощью формата сериализации данных Yaml.

Список реализованных методов REST-API:

- POST /session для отправки текущего состояния ТСО на сервер;
- GET /session для проверки связи ТСО с сервером и ТРК;
- GET /pumps для получения текущих состояний ТРК;
- GET /petrols для получения имеющихся на данный момент видов НП и цен на них;
- POST /orderpetrol для отправки информации по заказу НП на сервер;
- POST /orderother для отправки информации по заказу других услуг (перевода сдачи) на сервер;
- POST /scan для отправки ШК на сервер и получении информации по заказу с заданным ШК.

Схемы данных, используемые для передачи данных представлены в приложении В.2.

4.2 Разработка компонентов ИС

В данном разделе приведены краткие пояснения к реализации REST-сервиса (приложение В.3), написанного на ЯП Python с использованием фреймворков Django и DRF, а также графического интерфейса (GUI) для ТСО (приложение В.4), реализованного на ЯП

Python с помощью фреймворка PyQt5. Как было сказано ранее проект имеет сложную архитектуру и состоит из нескольких файлов.

4.2.1 Реализация REST-сервиса

Проект Django (у нас это папка RESTWS) включает следующие основные файлы:

- `manage.py` – программный файл с кодом утилиты, который позволяет запускать сервер, а также создавать и выполнять миграции моделей;
- `settings.py` – модуль с настройками проекта;
- `urls.py` – модуль с маршрутами уровня проекта.

Основное приложение проекта (Python пакет TSO в нашем проекте) содержит важные модули:

- `admin.py` – модуль административных настроек и классов редакторов;
- `models.py` – файл с моделями;
- `views.py` – файл с представлениями (контроллерами);
- `serializers.py` – модуль в котором хранятся сериализаторы (программные коды, отвечающие за сериализацию и десериализацию структур данных в формат JSON и обратно);
- `urls.py` – модуль с маршрутами уровня приложения.

4.2.2 Реализация графического интерфейса (GUI) для TCO

Все файлы графического модуля стоит разделить на 2 категории: первая категория файлов предназначена для программирования отдельных окон приложения. Реализация каждого окна будет храниться в отдельном файле. В каждом файле будет определён класс окна, наследующийся от класса `QMainWindow` модуля `QtWidgets`. Таким образом, программные реализации всех 14 экранов, представленных в ТЗ заказчика в пункте 1.1.1 будут храниться в следующих Python файлах:

1. Главный экран (рисунок 1.1 в пункте 1.1.1) – `MainScreen.py`
2. Информационный экран (рисунок А.1 в приложении А) – `InfoScreen.py`
3. Экран ошибок (рисунок А.2 в приложении А) – `ErrorScreen.py`
4. Экран выбора ТРК (рисунок А.3 в приложении А) – `PumpsScreen.py`
5. Экран выбора НП (пистолета) (рисунок А.4 в приложении А) – `PetrolsScreen.py`
6. Экран выбора способа оплаты (рисунок А.5 в приложении А) – `PaymentMethodScreen.py`
7. Экран ввода реквизитов клиента (рисунок А.6 в приложении А) – `CustomerDetailsScreen.py`
8. Экран оплат (наличные) (рисунок А.7 в приложении А) – `PaymentScreen.py`

9. Экран оплат (БК/ТК) (рисунок А.8 в приложении А) – `PaymentScreenCard.py`
10. Экран сканирования ШК (рисунок А.9 в приложении А) – `ScanScreen.py`
11. Экран перевода сдачи (выбор услуги) (рисунок А.10 в приложении А) – `OrderOtherSimcardScreen.py`
12. Экран перевода сдачи (по номеру телефона) (рисунок А.11 в приложении А) – `OrderOtherTelephoneScreen.py`
13. Экран перевода сдачи (на кошелек Элекснет) (рисунок А.12 в приложении А) – `OrderOtherWalletScreen.py`
14. Экран информации (рисунок А.13 в приложении А) – `InformationScreen.py`

Во всех экранах значимая часть бизнес-логики сокрыта в трёх основных методах: в конструкторе (магический метод `__init__`), в методе `setupUI` и `retranslateUI`. В конструкторе вызывается конструктор класса родителя `QMainWindow`, для соблюдения основной концепции ООП – наследования. Далее в конструкторе происходит вызов `setupUI()` – для создания и верного расположения основных виджетов на экране. Вызов `retranslateUI()` происходит в конце метода `setupUI`, когда все основные виджеты уже были созданы и инициализированы. `retranslateUI` создан для правильного отображения надписей на виджетах.

Чтобы лучше разобраться с тем как применять данные инструменты в разработке окон рассмотрим пример программирования Главного экрана (рисунок 1.1 в пункте 1.1.1). В методе `setupUI` создадим объект центрального виджета (объект класса `QWidget` с родителем – объектом класса окна (`self`)), который будет родителем последующих виджетов, отражающих желаемое расположение виджетов. Созданный объект сохраним как атрибут класса окна `self.centralwidget`. Внизу экрана находятся 3 надписи, равномерно распределённых по горизонтали. Чтобы добиться подобного нам потребуется 3 объекта класса `QLabel` (`self.label_2`, `self.label_3`, `self.label_4`) с родителем `self.centralwidget`. Данные `label`’ы необходимо поместить в горизонтальный компоновщик (`self.horizontalLayout` – объект класса `QHBoxLayout`) с помощью метода `addWidget` (`self.horizontalLayout.addWidget(self.label_2)`). Далее создадим вертикальный компоновщик (`self.verticalLayout`) объект класса `QVBoxLayout`. В нём разместим сверху вниз: заголовок с ограничениями (атрибут `self.label` – объект `QLabel`), 4 кнопки (`self.pushButton` (“Заправка”), `self.pushButton_2` (“Печать чека\nПеревод сдачи”), `self.pushButton_3` (“Другие услуги”), `self.pushButton_4` (“Информация”) – объекты `QPushButton`), вертикальный виджет для выравнивания (переменная `spacerItem2` – объект класса `QSpacerItem`). Сделаем это с помощью методов `addWidget` (для размещения виджетов) и `addItem` (для размещения объекта `QSpacerItem`). Создадим второй вертикальный компоновщик (атрибут

`self.verticalLayout_2`) и поместим в него первый вертикальный компоновщик (`self.verticalLayout`). Далее обратим внимание на верхнюю полосу на экране. На ней расположены текущая дата и время, смещенные влево. Реализуем надпись с помощью объекта `QLabel` (атрибут `self.label_5`), смещение с помощью ещё одного горизонтального объекта `QSpacerItem` (переменная `spacerItem`). Свяжем их вместе с помощью горизонтального компоновщика (атрибут `self.horizontalLayout_2` – объект класса `QHBoxLayout`). Теперь задумаемся о взаимном расположении всех виджетов на экране. Объединим все блоки виджетов сверху вниз: `self.horizontalLayout_2`, вертикальный объект `QSpacerItem` (переменная `spacerItem1`), `self.verticalLayout_2` в один вертикальный компоновщик (атрибут `self.verticalLayout_3`). Чтобы компоновка была равномерной и по вертикали, и по горизонтали нам потребуется последний горизонтальный компоновщик (атрибут `self.horizontalLayout_3`). В него мы поместим вертикальный компоновщик `self.verticalLayout_3`. И уже наш последний горизонтальный компоновщик (`self.horizontalLayout_3`) должен иметь в качестве родителя центральный виджет, который мы создавали вначале (`self.centralwidget`). Аналогично формируется взаимное расположение виджетов и в других окнах в методе `setupUI`.

Также в конструкторе должен быть словарь `_dictButtons` для хранения пар (ключ : значение) вида (источник сигнала : кортеж из текстового названия экрана и класса экрана). Здесь источником сигнала может быть нажатая кнопка, таймер закончивший отсчёт – любой объект, после которого логически ожидается переход на следующий экран.

Ещё важное назначение конструктора – связь между объектами источниками сигналов и слотами-функциями. Наиболее часто встречающимся слотом является метод `showScreen`. Ему стоит уделить особое внимание.

Метод `showScreen` есть во всех классах экранов и отвечает он за правильную логику перехода (отображения следующего экрана). Т.е. всегда, когда все необходимые бизнес-процессы на экране были завершены, встаёт вопрос о переходе на следующий экран. Для этого источник сигнала испускает свой сигнал, который ранее был связан с методом `showScreen`. Уже в слоте с помощью метода `sender()` класса `QObject` определяется чей именно был сигнал. В зависимости от источника подбирается нужный следующий экран с помощью словарика `_dictButtons`, а также формируются данные (атрибут `self.data`), которые стоит собрать с текущего экрана, добавить к уже имеющимся и передать дальше, для последующих экранов. Чаще всего данные обновляются с помощью метода `update_data`.

Вторая категория файлов объединяет общую логику для нескольких экранов, в отдельные файлы. Это позволит удобно пользоваться смежными функциями для различных

окон (описанных выше) с помощью классического выражения `import`. Так, в реализации присутствуют следующие 4 файла из второй категории:

1. `DatetimeLabel.py` – файл в котором реализованы функции работы с датой и временем в ИС.

Здесь функция `get_datetime(format)` принимает один из двух возможных форматов времени “ГГГГ-ММ-ДДТЧЧ:ММ”, “ДД.ММ.ГГГГ\пЧЧ:ММ” и в зависимости от переданного аргумента выдаёт строку с текущими датой и временем в системе в нужном виде. Первый используется для формирования JSON структуры по заказу НП в классах экранов оплат (наличными и банковской картой) `PaymentScreen.py` и `PaymentScreenCard.py`, второй же используется на главном экране в левом верхнем углу (в классе `MainScreen`). Функция `time_before_signal` используется только в классе `MainScreen` для правильной начальной инициализации таймера на количество секунд, оставшееся до ближайшей смены счётчика минут. Например, если окно Главного экрана открылось в 11 часов 56 минут 32 секунды (рисунок В.28 приложения В.6), то функция `time_before_signal` вернёт 28 секунд – через такое время необходимо будет обновить число минут до 57. По истечению указанных 28 секунд, таймер испустит сигнал `timeout` после чего связанный с ним слот `update_datetime` обновит время с помощью третьей функции модуля `DatetimeLabel.py` `set_current_datetime`, а также запустит таймер снова, но теперь уже на 60 секунд.

Также в данном модуле есть глобальные переменные `TIMER_DELAY` и `SLEEP_DELAY`. Переменная `TIMER_DELAY` хранит значение секунд, отведённое пользователю на ознакомление с экраном и принятие решения на дальнейшее взаимодействие. Обычно это 120 секунд, но при желании можно увеличить или уменьшить указанный временной интервал. Эта переменная используется во всех экранах, которые должны иметь возможность возврата на Главный экран через определённое количество секунд. Функция `set_current_time` принимает на вход аргумент `QLabel`, в котором будет обновляться число оставшихся секунд, и аргумент `decrease` типа `int`, указывающий сколько секунд уже прошло. Более подробно об этих экранах было рассказано выше в подразделе 3.2 при описании бизнес-логики на рисунке 3.2. Другая переменная `SLEEP_DELAY` хранит значение секунд, отведённое на HTTP-запрос (подраздел 4.1) к серверу (пункт 4.2.1). Обычно 3-х секунд более чем достаточно. Эта переменная используется в экранах из третьей категории бизнес-процессов, указанной в подразделе 3.2.

2. `general_functions.py` – файл, содержащий лишь одну функцию `create_inscription`.

Эта функция принимает текстовую информацию для пользователя в аргументе `inscription` и создаёт словарь с одним одноимённым ключом. Она часто используется в файле `InfoScreen.py` для формирования правильной надписи на

экране ошибок ErrorScreen (рисунок А.2 приложения А).

3. TSO_state.py – файл, в котором представлена реализация класса TSO_State, отвечающего за состояние ТСО (ограничения из пункта 1.1.2). Ограничения здесь представлены в виде атрибутов класса. В случае если ТСО имеет какое-то ограничение, то соответствующая переменная-атрибут будет принимать значение False. Например, когда не работает купюроприёмник, атрибут self.CURRENCYDETECTOR == False.

Также в качестве атрибута выступает номер заказа для данного сеанса (self.id). В конструкторе класса TSO_state значение по умолчанию для идентификатора заказа – 1. Дефолтное значение используется в случаях, когда ИС запускается на данном ТСО впервые и нет файла log.txt, в котором бы хранилось последнее значение заказа. Если же это не первый заказ, который формируется на данном ТСО, то статический метод create_id попытается открыть файл log.txt и прочитать предыдущее значение заказа. Если попытка закончится неудачно, то файл перезапишется со значением id по умолчанию. Метод create_id возвращает актуальный id заказа для данного сеанса. Метод update_id нужен для обновления id заказа, после завершения оконченного заказа (для корректного логирования, нет разницы был ли он успешен).

4. log.txt – файл, хранящий значение предыдущего заказа.

4.3 Демонстрация работы компонентов системы

Здесь описана краткая суть работы графического интерфейса (GUI) при взаимодействии с пользователем. Также отображена значимая часть работы REST-сервиса при обработке запросов пользователя. Реализована основная функциональность, указанная в ТЗ из пункта 1.1.1.

Скриншоты работы REST-сервиса представлены в приложении В.5, скриншоты работы графического интерфейса (GUI) представлены в приложении В.6. На них продемонстрированы основные тонкости БП-ов из подраздела 3.2.

Так, рисунки В.28-В.30 из приложения В.6 отображают результат бизнес-процесса из рисунка Б.1 приложения Б.1. На рисунке В.28 – Главный экран в начальном состоянии, не ограничивающем заправку на ТСО (нет информационного поля сверху экрана). На рисунке В.29 – Главный экран в ограниченном состоянии (не работает POS-терминал), о чём свидетельствует надпись сверху “В настоящее время оплата доступна только наличными”. Рисунок В.30 отображает Главный экран в ограниченном состоянии (не работает POS-терминал и купюроприёмник). Это значит, что ни одна из услуг недоступна, присутствует лишь кнопка “Информация”.

Рисунки В.31-В.45 приложения В.6 показывают возможный сеанс клиента, при формировании заказа на НП. Клиент, он же пользователь, нажал на кнопку “Заправка”, когда Главный экран был в состоянии из рисунка В.28 (всё работало безотказно). После этого отобразился Информационный экран с надписью “Загрузка” (Рисунок В.31). К сожалению, сервер оказался недоступен, о чём немедленно было сообщено пользователю на рисунке В.32. После того, как связь с сервером была восстановлена, Информационный экран перешёл на Экран выбора ТРК, где были отображены доступные ТРК с их текущими статусами (занята, свободна, недоступна) (Рисунок В.33). Пользователь попытался выбрать 4-ю колонку, но она оказалась недоступной (Рисунок В.34). После этого пользователь выбрал 3-ю колонку (Рисунок В.35). Далее на Экране выбора НП был выбран 95-ый бензин (Рисунок В.36) по цене 49.79 руб./литр. Открывшийся Экране выбора способа оплаты предлагает выбрать способ оплаты НП. Т.к. ограничений нет, отображаются все возможные виды оплат. Пользователь выбирает оплату наличными (Рисунок В.37). Затем на Экране ввода реквизитов клиента были введены следующие данные: мобильный – 89170932736, почта – My@email.ru (Рисунок В.38). На рисунках В.39-4.42 отображено взаимодействие пользователя с Экраном оплаты наличными. Рисунок В.39 – начальное состояние экрана, рисунок В.40 эмитирует внесение 100-рублёвой купюры. Рисунок В.41 показывает обновление счётчиков после внесения 1 купюры. На рисунке В.42 – отображена окончательная информация о заказе НП для данного сеанса. Рисунки В.43-В.45 демонстрируют последовательную смену надписей на Информационном экране. Рисунок В.43 – Надпись “Подождите, идёт печать” на Информационном экране информирует клиента о печати квитанции заказа. Рисунок В.44 – Надпись “Возьмите напечатанную квитанцию заказа” на Информационном экране оповещает клиента об окончании процесса печати. Рисунок В.45 – Надпись “ТРК занята. Ожидайте пока ТРК освободится.” на Информационном экране отображает ответ сервера с кодом 400 и пояснением “Pump is BUSY now!”. Клиенту остаётся лишь подождать, пока стоящая перед ним машина закончит заправку.

Рисунки В.46, В.47 отображают работу REST-сервиса во время отправки заказа. На рисунке В.46 – В БД на сервере в этот момент последняя запись для ТРК №3 соответствует статусу BUSY (ТРК ещё не освободилась). Рисунок В.47 имитирует окончание заправки впереди стоящего клиента должно завершиться записью в таблицу Pump states. Для этого была создана актуальная запись для ТРК №3 со статусом FREE.

После того, как ТРК №3 освободилась, Информационный экран отреагировал на это и продолжил БП оплаты наличными. На рисунке В.48 отображена надпись “Подождите, идёт печать”. Рисунок В.49 отображает последнее действие перед возвратом на Главный

экран – оповещение клиента о необходимости взять напечатанный чек.

Рисунки В.50-4.52 показывают, как сформированный заказ обработал REST-сервис. Рисунок В.50 – Для сервера окончание бизнес-процесса означает появление записи для выбранной ТРК со статусом BUSY (мы заняли ТРК №3). Также в таблице Orders появляется запись о заказе НП (тип PETROL) (Рисунок В.51). Заказ 42 имеет ID 131 был осуществлён с терминала №5 19 мая 2021 в 12:06. Автоматически сгенерированный штрих-код является уникальным и однозначно идентифицирует заказ в БД. В таблице Order petrols появляется запись, дополняющая информацию о данном заказе НП (Рисунок В.52). Запись о заказе имеет тот же ID 131, хранит состояние ТРК из таблицы Pump states, вид НП из таблицы Petrols, налитый объём, заплаченную сумму в рублях, телефон и e-mail клиента, а также тип оплаты. Поле “Перевод сдачи” не заполнено, т.к., во-первых, налитый объём соответствует оплаченной сумме, а, во-вторых, процедура перевода сдачи даже не была осуществлена.

Рассмотрим отличия бизнес-логики программы при выборе оплаты банковской картой на Экране выбора способа оплаты (Рисунок В.53). Вместо Экрана оплаты (наличные) открывается Экран оплаты (БК/ТК) (Рисунок В.54). На экране осуществляется автоматический перевод суммы в рублях на соответствующий объём в литрах для выбранного НП (OR95 : 49.79). Переход на Информационный экран происходит после нажатия на кнопку “Оплатить”. Рисунок В.55 – Бизнес-логика данного экрана соответствует нижней ветви на рисунке Б.2 приложения Б.1. Надпись “Следуйте инструкциям на пин-паде” говорит о необходимости продолжить оплату картой через POS-терминал. Транзакция может быть успешной и неуспешной. В целях имитации работы терминала созданы две соответствующие кнопки в правом верхнем углу. Предположим, что транзакция закончилась успехом. Рисунок В.56 – В случае успеха бизнес-логика Информационного экрана далее соответствует оплате наличными (Рисунок В.43, средняя ветвь бизнес-логики на рисунке Б.2 приложения Б.1). В случае неуспешной транзакции при оплате картой, происходит переход на Экран ошибок (Рисунок В.57). Рисунок В.58 – Надпись “Ошибка оплаты” на Экране информации отображает статус транзакции на POS-терминале. Рисунок В.59 – Экраны, имеющие в левом верхнем углу надпись “Возврат в главное меню через:” со счётчиком секунд автоматически возвращают пользователя на Главный экран по истечению времени. Также кнопка “Выход” позволяет перейти на Главный экран, не дожидаясь окончания времени.

ЗАКЛЮЧЕНИЕ

Данная работа посвящена определению методов подхода к реализации программного обеспечения, анализу функциональности, разработке архитектуры и примера последующей реализации модулей информационной системы для терминалов саобслуживания автозаправочной станции (АЗС).

В первом разделе основного тома был проведён анализ требований заказчика на разрабатываемое программное обеспечение.

Во втором разделе основного тома по результатам анализа были осуществлены выбор архитектуры, произведён анализ различных протоколов обмена данными, а также выбор технологий для разработки.

В третьем разделе основного тома был произведён отбор данных, используемых ТСО, смоделированы и спроектированы все модули ИС, с учётом особенностей форматов данных и разработаны эффективные алгоритмы для работы с ними.

В четвёртом разделе основного тома была описана реализация компонентов ИС (HTTP-процедур обмена данными, REST-сервиса, графического интерфейса для ТСО).

В результате работы:

- были написаны «листинги» исходных кодов REST-сервиса и графического интерфейса для ТСО
- определена структура БД ИС
- промоделировано заполнение информацией БД в ИС, поступившей со стороны клиента веб-сервису, продемонстрирована работа всех модулей системы.

Цель работы достигнута, поставленные задачи выполнены. В данный момент разработанный макет проходит стресс-тестирование на экспериментальном терминале.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Astra Linux [Электронный ресурс] // Википедия. Свободная энциклопедия. URL: https://ru.wikipedia.org/wiki/Astra_Linux (дата обращения: 28.05.2021)
2. Банокин П. И. Проектирование программных приложений. Национальный исследовательский Томский политехнический университет. – Томск, 2012. – 92 с.
3. Монолитная vs Микросервисная архитектура [Электронный ресурс] // proglib.io URL: <https://proglib.io/p/monolitnaya-vs-mikroservisnaya-arhitektura-2019-09-16/> (дата обращения: 19.05.2021)
4. Использование Python в веб-разработке: плюсы и минусы [Электронный ресурс] // pythonist.ru URL: <https://pythonist.ru/ispolzovanie-python-v-veb-razrabotke-plyusy-i-minusy/> (дата обращения: 17.05.2021)
5. Создание Web-сервисов на Python [Электронный ресурс] // coursera.org URL: <https://www.coursera.org/learn/python-for-web> (дата обращения: 10.05.2021)
6. Qt [Электронный ресурс] // Википедия. Свободная энциклопедия. URL: <https://ru.wikipedia.org/wiki/Qt> (дата обращения: 22.05.2021)
7. PyQt [Электронный ресурс] // Википедия. Свободная энциклопедия. URL: <https://ru.wikipedia.org/wiki/PyQt> (дата обращения: 22.05.2021)
8. Назаров С. В. Архитектура и проектирование программных систем. – М.: Издательский центр «ИНФРА-М», 2013. – 413 с.
9. Коптенок Е. В., Трунников М. В., Сухарев Е. А. [и др.] Применение UML-диаграмм для проектирования программных комплексов. // Молодой ученый. 2020. № 19 (309). С. 133-135.
10. Моделирование бизнес-процессов [Электронный ресурс] // youtube.com URL: <https://www.youtube.com/watch?v=JAsQEgUojMk/> (дата обращения: 13.03.2021)
11. Моделирование бизнеса — IDEF, UML, ARIS [Электронный ресурс] // analytics.infozone.pro URL: <https://analytics.infozone.pro/business-modeling-idef-uml-aris/> (дата обращения: 25.02.2021)
12. Дронов В. А. Django 3.0. Практика создания веб-сайтов на Python. – СПб: БХВ-Петербург, 2021. – 704 с.
13. Welcome to Python.org [Электронный ресурс] // python.org URL: <https://www.python.org/> (дата обращения: 11.02.2021)
14. Документация Django [Электронный ресурс] // docs.djangoproject.com URL: <https://docs.djangoproject.com/en/3.1/> (дата обращения: 11.02.2021)
15. Django REST framework [Электронный ресурс] // django-rest-framework.org

URL: <https://www.django-rest-framework.org/> (дата обращения: 11.02.2021)

16. Прохорёнок Н. А. Python 3 и PyQt 5. Разработка приложений. – СПб: БХВ-Петербург, 2016. – 832 с.