

**СОДЕРЖАНИЕ**

[В](#_Toc74752608)[[ВЕДЕНИЕ 6](#_Toc74752608)](#_Toc75092388)

[1ТЕОРЕТИЧЕСКИЙ АНАЛИЗ ИСПОЛЬЗОВАНИЯ ПЛАТЕЖНЫХ СИСТЕМ В СЕРВИСАХ ЭЛЕКТРОННОЙ КОММЕРЦИИ 8](#_Toc74752609)

[1.1 Описание предметной области 8](#_Toc74752610)

[1.1.1 Электронная коммерция 8](#_Toc74752611)

[1.1.2 Платежный шлюз 9](#_Toc74752612)

[1.1.3 Программный интерфейс 9](#_Toc74752613)

[1.2 Описание платежного шлюза TranzAxis Payment Gateway 10](#_Toc74752614)

[1.2.1 Общее описание платежного шлюза TranzAxis Payment Gateway 10](#_Toc74752615)

[1.2.2 Основные функции и возможности TranzAxis Payment Gateway 10](#_Toc74752616)

[1.2.3 Архитектура системы TranzAxis Payment Gateway 12](#_Toc74752617)

[1.2.4 Основные объекты TranzAxis Payment Gateway 16](#_Toc74752618)

[1.3 Обзор существующих решений 21](#_Toc74752619)

[1.4 Цель и задачи выпускной квалификационной работы 22](#_Toc74752620)

[1.5 Выводы по первой главе 22](#_Toc74752621)

[2 ПРОЕКТНЫЕ РЕШЕНИЯ РАЗРАБОТКИ ПРОГРАММНОГО ИНТЕРФЕЙСА ДЛЯ ВЗАИМОДЕЙСТВИЯ ИНТЕРНЕТ-МАГАЗИНОВ С ПЛАТЕЖНОЙ СИСТЕМОЙ TRANZAXIS PAYMENT GATEWAY 24](#_Toc74752622)

[2.1 Описание функциональных требований, предъявляемых к разрабатываемому программному интерфейсу 24](#_Toc74752623)

[2.2 Описание выбранной платформы для разработки программного интерфейса 25](#_Toc74752624)

[2.3 Структура программного интерфейса 26](#_Toc74752625)

[2.4 Схема обработки запросов программного интерфейса 34](#_Toc74752626)

[2.5 Основные модули и функции работы программного интерфейса 38](#_Toc74752627)

[2.6 Выводы по второй главе 44](#_Toc74752628)

[3 РЕЗУЛЬТАТЫ ОПЫТНОЙ ЭКСПЛУАТАЦИИ ПРОГРАММНОГО ИНТЕРФЕЙСА ДЛЯ ВЗАИМОДЕЙСТВИЯ ИНТЕРНЕТ-МАГАЗИНОВ С ПЛАТЕЖНОЙ СИСТЕМОЙ TRANZAXIS PAYMENT GATEWAY 45](#_Toc74752629)

[3.1 Описание программного интерфейса 45](#_Toc74752630)

[3.2 Пример взаимодействия программного интерфейса с TranzAxis Payment Gateway 56](#_Toc74752631)

[3.3 Выводы по третьей главе 58](#_Toc74752632)

[ЗАКЛЮЧЕНИЕ 59](#_Toc74752633)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 61](#_Toc74752634)

[ПРИЛОЖЕНИЕ А 62](#_Toc74752635)

[ПРИЛОЖЕНИЕ Б 69](#_Toc74752635)

[ПРИЛОЖЕНИЕ В 77](#_Toc74752635)

ВВЕДЕНИЕ

До появления интернета люди приобретали товары в местных торговых комплексах, что в свою очередь отнимало много времени на поиски качественного и недорогого товара, но появление электронной коммерции стало важнейшим достижением человечества в сфере высоких технологий. Системы электронных платежей экономят время и деньги пользователей, так как теперь им доступны все рынки мира, а стоимость товара в интернет-магазинах, как правило, намного ниже, чем в обычных. В таком случае представители онлайн-трейдинга работают напрямую с производителями продукции и им не надо переплачивать за содержание торговой площади. Еще одна заслуга интернет-магазинов в том, что благодаря им появились электронные системы оплаты и электронные кошельки.

Система электронных платежей – технология осуществления торговой деятельности между двумя компаниями, позволяющая совершать финансовые транзакции через интернет. Система электронных платежей позволяет производить оплату через интернет без участия посредника. В настоящее время использование данных систем очень популярно, так как позволяет совершать финансовые операции через интернет в режиме реального времени, что в свою очередь экономит много времени. Примерами систем электронной коммерции являются интернет-магазины, интернет-банки.

Интернет-магазины очень востребованы на данный момент, но для того чтобы они функционировали, необходимо произвести процесс обработки транзакций торгово-сервисного предприятия при взаимодействии с платежной системой. После выбора пользователем необходимых товаров, ему необходимо произвести оплату, для этого он вводит данные своей банковской карты и они отправляются в платежную систему. Процесс передачи данных осуществляется за счет программного интерфейса, который может связать множество интернет-магазинов с одной платежной системой, обеспечивая при этом настолько эффективный обмен информацией, что пользователи этого даже не замечают.

Актуальность данной работы заключается в создании системы автоматизации процесса приема электронных платежей, для того чтобы торгово-сервисным предприятиям было удобно взаимодействовать с платежным шлюзом TranzAxis Payment Gateway.

Целью выпускной квалификационной работы является совершенствование процесса взаимодействия платежных сервисов с системой TranzAxis Payment Gateway, за счет разработки программного интерфейса REST API. Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать использование платежных систем в сервисах электронной коммерции, изучить систему TranzAxis Payment Gateway.
2. Cпроектировать и разработать программный интерфейс для взаимодействия интернет-магазинов с платежной системой TranzAxis Payment Gateway.
3. Произвести опытную эксплуатацию программного интерфейса для взаимодействия интернет-магазинов с платежной системой TranzAxis Payment Gateway.

# ТЕОРЕТИЧЕСКИЙ АНАЛИЗ ИСПОЛЬЗОВАНИЯ ПЛАТЕЖНЫХ СИСТЕМ В СЕРВИСАХ ЭЛЕКТРОННОЙ КОММЕРЦИИ

## Описание предметной области

# Электронная коммерция

Электронная коммерция – осуществление торговой деятельности, позволяющей совершать финансовые транзакций через интернет, при помощи электронных средств платежа (банковская карта, электронный кошелек) [1]. Для воплощения в жизнь электронной коммерции в сети интернет, необходимо реализовать три основных системы:

1. Платежная система, позволяющая обрабатывать финансовые транзакции.
2. Платежный сервис, который позволяет клиенту выбрать перечень товаров и услуг.
3. Программный интерфейс, позволяющий интернет приложению взаимодействовать с платежным шлюзом.

Электронная коммерция, в зависимости от участников сделки, подразумевает различные виды взаимодействия:

1. Бизнес для потребителя (B2C) – схема коммерческого взаимодействия между торгово-сервисным предприятием и потребителем (физическим лицом). Где покупателем является потребитель, а продавцом – компания, организация.
2. Бизнес для бизнеса (B2B) – схема коммерческого взаимодействия между двумя компаниями.
3. Потребитель для потребителя (С2С) – схема коммерческого взаимодействия между двумя потребителями, что подразумевает собой осуществление сделок между физическими лицами.
4. Бизнес для правительства (B2G) – схема коммерческого взаимодействия, при которой роль продавца на себя берет юридическое лицо, а покупателем является государственная инстанция.
5. Потребитель для бизнеса (C2B) – схема коммерческого взаимодействия между потребителем и компанией. В данном случае покупатели сами определяют цену на товар, в результате голосования, а продавцов ищет торговая площадка, которая выступает в роли посредника.

# Платежный шлюз

Платежный шлюз – программный продукт, предназначенный для обработки финансовых операций. Платежный шлюз обеспечивает безопасное взаимодействие платежного сервиса с банком или процессором платежей, за счет шифрования конфиденциальных данных.

Главной задачей платежного шлюза является – безопасно и бесперебойно осуществлять каждую транзакцию между мерчантом и банком-эквайером в минимальные сроки [2]. Пользователю платежного сервиса необходимо лишь ввести необходимые данные о карте, которую ему выдал банк, на веб странице или в приложении, после чего эта информация отправляется в платежный шлюз.

# Программный интерфейс

Программный интерфейс (API) – программный продукт, представляющий из себя набор методов или команд, которые позволяют взаимодействовать между собой двум различным системам. Использование программного интерфейса обладает рядом преимуществ:

1. Обеспечение программных продуктов возможностью адаптировать другие сервисы.
2. Уменьшение затрат на разработку программных продуктов.
3. Быстрый и эффективный обмен информацией.

Программный интерфейс является посредником, который связывает части программной системы и передает необходимые данные. Так как API позволяет получить доступ к информации, они подразделяются на категории:

1. Частные API. Частные программные интерфейсы доступны определенному кругу пользователей, например, разработчикам компании, у которых есть одновременный доступ к необходимым данным. Использование частных API защищает конфиденциальные данные компании и упрощает совместную работу.
2. Открытые API. Общедоступные программные интерфейсы, которые предоставляют доступ к проприетарным программным продуктам.

## Описание платежного шлюза TranzAxis Payment Gateway

### ***Общее описание платежного шлюза TranzAxis Payment Gateway***

Система TranzAxis Payment Gateway предоставляет мерчантам широкий набор платежных методов, современные стандарты и тенденции электронной коммерции, которые базируются на универсальном механизме описания и исполнения сценариев сделок. Для удобного взаимодействия с платежным шлюзом, весь функционал системы доступен через программный интерфейс. TranzAxis Payment Gateway легко интегрируется в IT-инфраструктуру клиента, также в системе реализован сбор, обработка и сверка расчетной информации от эквайеров [3]. Система поддерживает мультиарендность – возможность обслуживания нескольких независимых платежных агрегаторов в рамках одной инсталляции приложения. Обобщенная схема работы системы обработки финансовых транзакций представлена на рисунке 1.1.

схема API

Рисунок 1.1 – Обобщенная схема работы системы обработки финансовых транзакций

### **Основные функции и возможности TranzAxis Payment Gateway**

Система TranzAxis Payment Gateway предоставляет мерчантам широкий функционал, позволяющий совершать финансовые операции. Ниже будут рассмотрены основные возможности платежного шлюза.

TranzAxis Payment Gateway позволяет обработать множество различных сценариев сделки. При совершении заказов, одним из важных этапов является аутентификация – подтверждение логина и пароля пользователя, которые передаются в зашифрованном виде, что в свою очередь обеспечивает безопасность выполнения транзакций. Так же система позволяет обработать спорные ситуации, возникшие при совершении сделки и взаимодействовать с участниками процесса [4]. Основной задачей платежного шлюза является обработка финансовых транзакций – операция, совершенная путем оплаты банковской картой.

Система поддерживает различные классы ордеров: ордер на продажу, инвойс на продажу, ордер на перевод денежных средств, ордер на аутентификацию, ордер на пополнение счета, инвойс на пополнение счета, ордер на оплату вендору, инвойс на оплату вендору, ордер на выплату клиенту, ордер подписки. Клиенты могут определять собственные классы ордеров.

В платежном шлюзе поддерживаются интерфейсы с операторами платежных методов для запроса информации о доплатах/DCC – это разница между суммой, взимаемой с плательщика, и суммой, которая передается получателю. Доплата либо увеличивает сумму, которую отдает плательщик "доплата сверху", либо уменьшает сумму, которую получает получатель "доплата внутри" [5]. В некоторых случаях допустимы оба варианта, тогда вариант выбирает клиент (при этом следует установить такую ставку, что при одинаковой сумме получателя получалась одинаковая сумма плательщика). Величина доплаты удерживается эквайером. Эквайер учитывает доплаты как собственный доход и может делиться этим доходом с ТСП и агрегатом. Также система осуществляет все транзакции жизненного цикла сделок: покупка, платеж, кредит, отмена, возврат товара, простая, предварительная и инкрементальная авторизация, финансовое представление.

Tranz Axis Payment Gateway поддерживает программы платежных систем, основанные на протоколах 3-D Secure – дополнительная аутентификация пользователя, которая позволяет убедиться в том, что платеж совершает владелец карты и избежать мошеннических операций. При совершении оплаты, пользователь получает SMS-сообщение с одноразовым кодом подтверждения, который ему необходимо отправить для дальнейшего выполнения операции по оплате. Система поддерживает 3-D Secure v1.x (Visa Secure, Masterсard Secure Code, Masterсard Identity Check, American Express Safe Key, J/Secure и Mir Accept) иEMV 3-D Securev2.х (Visa Secure, Masterсard Identity Check, Mir Accept 2.х) [6].

Система поддерживает работу с сервисами токенизации – возможность совершения платежей пользователем при помощи идентификатора, который можно использовать вместо данных банковской карты. Данные сервисы также позволяют совершать оплату покупок при помощи смартфона. Система позволяет взаимодействовать с сервисами ApplePay и GooglePay.

TranzAxis Payment Gateway предоставляет сервис безопасного хранения платежных реквизитов, а также возможность выполнения операций с использованием сохраненных платежных реквизитов (подписки на выполнение платежей, оплаты в рассрочку, автоматических платежей).

TranzAxis Payment Gateway поддерживает множество интернет-магазинов на одной инстанции системы. Для каждого интернет-магазина можно индивидуально настраивать параметры и правила исполнения и сопровождения сделок (использование технологии 3-D Secure разных версий в разных сценариях, возможность выполнения операций отмены, возврата товара, сверки итогов).

TranzAxis Payment Gateway предоставляет возможность обслуживания нескольких независимых платежных агрегаторов в рамках одной инсталляции приложения (мультиарендность) [7]. Для каждого агрегатора можно использовать индивидуальные и общие настройки. Также система позволяет информировать клиентов по электронной почте и SMS, и предоставляет возможность информирования ТСП, эквайеров посредством интерфейсов и по электронной почте.

### **Архитектура системы TranzAxis Payment Gateway**

В TranzAxis Payment Gateway бизнес-транзакции исполняются в составе ордеров (сделок). Исполнение ордера представляет собой бизнес-сценарий, то есть последовательность шагов со всеми возможными разветвлениями и вариантами. Сценарий исполнения ордера позволяет определять участников сделки и очередность взаимодействия с ними, а также выполнять информационные, авторизационные и финансовые транзакции с помощью различных платежных методов, взаимодействовать с внешними сервисами, извещать и запрашивать участников. В TranzAxis Payment Gateway реализован универсальный механизм исполнения ордеров, который обеспечивает возможность исполнения сложных сценариев с несколькими участниками, множеством шагов и разветвленной логикой [8].

TranzAxis Payment Gateway предоставляет широкий набор готовых настраиваемых сценариев ордеров для выполнения покупок, P2P-переводов, платежей (включая подписки, платеж в рассрочку, погашение кредита, автоматические платежи). Поставляемые в составе системы инструменты позволяют настраивать и изменять существующие сценарии сделок, а также конструировать новые сценарии. Например, типовой сценарий ордера на выполнение коммунального платежа включает этапы:

1. Создание клиентом ордера на платеж.
2. Выбор клиентом платежного метода из списка доступных.
3. Получение платежных реквизитов от клиента (ввод информации по карте на платежной странице, получение реквизитов из электронного кошелька, получение реквизитов от сервиса ApplePay, GooglePay).
4. Выбор оператора платежного метода и эквайера.
5. Оценка риска ордера с учетом выбранного платежного метода, параметров ордера, информации о плательщике или получателе.
6. Получение дополнительных реквизитов платежа от клиента (адрес квартиры, номер лицевого счета, показания счетчиков).
7. Проверка реквизитов платежа через API получателя.
8. Получение курсов валют (сервис DCC).
9. Расчет эквайринговой комиссии за проведение платежа.
10. Отображение клиенту информации о возможности конвертации суммы платежа и о взимании комиссии за проведение платежа.
11. Проведение аутентификации клиента (3-D Secure или другого типа), обеспечивающей проверку подлинности клиента.
12. Исполнение финансовой транзакции через интерфейс с оператором платежного метода (эквайером).
13. Передача данных платежа получателю.
14. Формирование и отображение клиенту информации о результатах выполнения платежа, отправка чека клиенту по электронной почте.

TranzAxis Payment Gateway состоит из следующих компонентов:

1. TranzAxis Payment Gateway Frontend Server – включает веб-приложения:
   * платежная страница (HPP), API Gateway и портал предприятий, реализующие взаимодействие с клиентом и ТСП;
   * UL Middleware, обеспечивающее обработку запросов UL 3DS Self-TestPlatform в рамках тестирования и сертификации ПП по протоколу EMV 3-D Secure v2.х.
2. TranzAxis Payment Gateway Backend Server – реализует исполнение ордеров, интерфейсы для взаимодействия с внешними системами, хранение настроек и данных. Компонент BES в приложении TranzAxis Payment Gateway разработан на базе технологической платформы RadixWare. Сервер приложений RadixWare Server реализует бизнес логику системы. Задачи хранения данных решаются средствами сервера БД Oracle. Платформа RadixWare обеспечивает:
3. Быстродействие и производительность. Эффективность обработки сценариев электронной коммерции обеспечивается использованием высокопроизводительного асинхронного ядра обработки APE. Благодаря APE в системе отсутствует синхронное ожидание ресурсов, длительных операций или ответов внешних систем, а также сводится к минимуму конкуренция в БД.
4. Транзакционную целостность. Целостность гарантируется использованием промышленной БД, которая обеспечивает надежное и консистентное хранение информации.
5. Отказоустойчивость. Сервер приложений обеспечивает высокий уровень отказоустойчивости и гарантирует работоспособность системы в целом, при возможном отказе ее отдельных компонентов, модулей или целых инстанций. В системе отсутствуют узлы, выход которых из строя приводит систему в неработоспособное состояние (т. е. отсутствует единая точка отказа).
6. Непрерывность эксплуатации. Платформа ориентирована на критически важные системы с самым высоким уровнем доступности. В архитектуру заложена возможность установки обновлений без остановки работы системы.
7. Масштабируемость. Масштабирование производительности в TranzAxis Payment Gateway обеспечивается следующими особенностями системы:
   * + - возможность организации серверного звена на кластере неограниченного размера. За счет пропорционального увеличения аппаратных ресурсов (наращивания процессорных мощностей и увеличения количества узлов кластера) достигается близкое к линейному масштабирование вычислительной мощности системы;
       - многопоточная обработка данных, позволяющая каждой серверной инстанции максимально эффективно использовать ресурсы многопроцессорного многоядерного сервера;
       - технология многоуровневого кеширования, позволяющая уменьшить нагрузку на сервер БД и повысить скорость работы с данными.
8. Кастомизируемость. В системе предусмотрены средства кастомизации, которые позволяют пользователю самостоятельно (без участия вендора) определять и переопределять многие аспекты поведения системы, в частности, посредством разработки пользовательских функций, кастомных действий и запросов, реализации интерфейсов средствами подсистемы Service Bus.
9. Возможности для интеграции. Благодаря поддержке API система легко интегрируется в IT-инфраструктуру клиента. Платформа способна взаимодействовать с другими ПП Компании, с основными провайдерами различных дополнительных сервисов и прочими системами, представленными на рынке.
10. Безопасность и аудит. Для обеспечения безопасности работы в TranzAxis Payment Gateway используются встроенные средства разграничения доступа. Полномочия пользователя в отношении объектов TranzAxis Payment Gateway определяются через механизм ролей. Роли характеризуются множеством доступных объектов и доступными операциями над объектами (просмотр/изменение/исполнение и т. д.). Для обеспечения дополнительной безопасности при управлении полномочиями пользователя/группы пользователей может использоваться механизм двойного контроля. Регистрация действий пользователей в TranzAxis Payment Gateway обеспечивается внутренними средствами аудита. Предусмотренные средства мониторинга позволяют отслеживать текущее состояние работы системы, регистрировать список событий в системе и изменения, производимые в БД.

### **Основные объекты TranzAxis Payment Gateway**

Ниже будут рассмотрены основные объекты платежного шлюза TranzAxis Payment Gateway [9].

*Торгово-сервисное предприятие*

Торгово-сервисное предприятие (ТСП) – программный комплекс, принимающий оплату в виде цифровых платежных методов за товары или услуги. Объект «Торгово-сервисное предприятие» описывает в системе торгово-сервисное предприятие (ТСП) или его подразделения. Торгово-сервисное предприятие принадлежит агрегатору. Объекты торгово-сервисного предприятия образуют древовидную иерархию. Иерархия влияет на доступ следующим образом:

1. Доступ операторов портала к ТСП распространяется на дочерние ТСП.
2. Доступ родительского ТСП к клиенту распространяется на дочерние ТСП.

*Агрегатор*

Агрегатор – Компания, предоставляющая ТСП доступ к эквайерам платежных методов и операторам платежных методов, а также иные сервисы, востребованные ТСП. Объект «Агрегатор» описывает платежного агрегатора, обслуживаемого TranzAxis Payment Gateway. Функции агрегатора:

1. Обеспечение доступа ТСП к платежным методам эквайеров. Один агрегатор может иметь доступ к нескольким эквайерам посредством нескольких операторов платежных методов.
2. Обеспечение доступа ТСП к сервисам, предоставляемым сервис-провайдерами.
3. Предоставление ТСП информации и отчетов, связанных с ордерами и оплатой.
4. Предоставление ТСП инструментов для анализа данных ордеров, платежей и измерения конверсии.
5. Хранение данных клиентов, включая платежные реквизиты.

Агрегатор может иметь собственную конфигурацию и использовать общую (определенную на общесистемном уровне и доступную для использования всеми агрегаторами, настроенными в системе). В рамках одной инсталляции приложения могут обслуживаться несколько независимых агрегаторов.

*Ордер*

Ордер – индивидуальная сделка на предоставление ТСП клиенту товара или услуги. Ордер отражает финансовые и прочие условия сделки, ход и состояние ее исполнения. Объект «Ордер» представляет в системе детали сделки и ее текущее состояние. Настройки ордеров определяются их типами. Типы ордеров образуют иерархию с наследованием свойств. Над ордером исполняются и регистрируются действия. В рамках ордера исполняются (как действия определенных классов) и регистрируются транзакции.

*Оператор платежного метода*

Оператор платежного метода – Компания, предоставляющая ТСП возможность использования платежного метода (карта, банковский счет) для оплаты товаров или услуг. Объект «Оператор платежного метода» описывает в системе организацию, предоставляющую ТСП сервис для авторизации и клиринга платежей по платежным методам [10]. Такой организацией может выступать карточный процессор и т. п. Оператор платежного метода обслуживает ТСП нескольких агрегаторов, обеспечивая выполнение транзакций с использованием платежных методов.

*Эквайер*

Эквайер – Организация-посредник, имеющая лицензию от международной платежной системы или уполномоченного ей участника осуществлять обслуживание предприятий торговли. Объект «Эквайер» описывает в системе организацию, от которой ТСП получают возмещение по платежам и соответствующую информацию для сверки. Такой организацией может выступать банк-эквайер или другой агрегатор. Функции эквайера:

1. Обслуживание ТСП нескольких агрегаторов.
2. Выполнение взаиморасчетов с ТСП по транзакциям. Данные о проведенных взаиморасчетах и выплаченном ТСП возмещении эквайер направляет агрегатору.
3. Ведение диспутов, связанных с транзакциями ТСП. Данные о диспутах ТСП эквайер направляет агрегатору.

*3DS Directory Server*

3DS Directory Server – аппаратно-программный комплекс, размещаемый в межоперационном домене 3-D Secure и предназначенный для маршрутизации запросов на аутентификацию держателей карт. Объект «3DS Directory Server» описывает в системе Directory Server ПС (Directory Server Mastercard, Directory Server Visa) [11]. Функции объекта 3DS Directory Server:

1. Хранение полученной от Directory Server ПС информации о диапазонах карт, участвующих в программе 3-D Secure по протоколу 3DS v2.х.
2. Маршрутизация запросов на 3-D Secure аутентификацию держателей карт.

*Терминал*

Терминал – объект, от имени которого создаются ордера. Терминал принадлежит ТСП. Объект «Терминал» может представлять:

1. Систему управления интернет-магазина.
2. Наземный POS терминал.

*Клиент*

Клиент – покупатель товаров и услуг торгово-сервисного предприятия. Объект «Клиент» описывает в системе клиента ТСП или сервиса системы покупателя товаров и услуг, отправителя переводов. Клиент принадлежит ТСП или системе в целом.

*Токен*

Объект «Токен» предоставляет возможность хранения платежных реквизитов, которые могут использоваться для оплаты товаров и услуг покупателем или ТСП по подписке. Объект Токен может использоваться в следующих качествах:

1. Платежное средство ордера: токен клиента (плательщика) или получателя ордера. Платежные реквизиты хранятся в токене только в течение жизненного цикла ордера.
2. Сохраненный токен (CoF), в котором хранятся платежные реквизиты для последующего использования в других транзакциях. Сохраненный токен может принадлежать ТСП, клиенту ТСП, глобальному клиенту или подписке.

*Логины*

Объект «Логин» предоставляет доступ внешним системам и пользователям для исполнения определенных действий и запросов над определенными объектами TranzAxis Payment Gateway. Логин принадлежит агрегатору и одному или нескольким из его объектов (ТСП, терминалу или клиенту) в зависимости от назначения логина.

*Интерфейсы*

Объект «Интерфейс» используется для определения правил и параметров взаимодействия BES с участниками сделки, FES, внешними системами. Для интерфейса, реализующего взаимодействие FES c BES, используется Action Execution Service.

К интерфейсам для внешних систем относятся:

1. Исходящий интерфейс FIMI, реализующий взаимодействие TranzAxis Payment Gateway c оператором платежного метода по протоколу FIMI (ПП TWO). Интерфейс FIMI может использоваться для исполнения транзакций, передачи эквайеру извещений по транзакциям, расчета величин ордера (комиссии и конвертации валюты).
2. Исходящий интерфейс RTP, реализующий взаимодействие TranzAxis Payment Gateway c оператором платежного метода по протоколу RTP (ПП TranzAxis). Интерфейс RTP может использоваться для исполнения транзакций, передачи эквайеру извещений по транзакциям, расчета величин ордера (комиссии и конвертации валюты).
3. Исходящие интерфейсы к другим операторам платежных методов.
4. Исходящий интерфейс 3DS Directory Server v1.x, реализующий взаимодействие TranzAxis Payment Gateway с DirectoryServer для аутентификации в соответствии с протоколом 3DS v1.x.
5. Исходящие / входящие интерфейсы 3DS Directory Server v2.х, реализующие взаимодействие TranzAxis Payment Gateway с Directory Server для аутентификации в соответствии с протоколом 3DS v2.х.
6. Исходящий интерфейс CTS, реализующий взаимодействие TranzAxis Payment Gatewayc сервисом ApplePay.

В течение жизненного цикла над ордером выполняются действия по инициативе различных участников сделки. Каждый участник может быть привлечен к выполнению действий над ордером посредством вызова через интерфейс или отправки извещения.

К интерфейсам для участников сделки относятся:

1. Интерфейс взаимодействия с ТСП. Варианты взаимодействия:

* извещение о результатах 3-D Secure аутентификации клиента;
* извещение об изменении состояния ордера, о выполненных транзакциях.

Интерфейс взаимодействия с ТСП. Варианты взаимодействия:

1. Интерфейс взаимодействия с эквайером. Извещение может потребоваться эквайеру, в частности, при использовании push-платежных методов. В этом случае оператор платежного метода извещает TranzAxis Payment Gateway о статусе транзакции, а TranzAxis Payment Gateway может передать извещение эквайеру для включения транзакции в расчеты.
2. Интерфейс взаимодействия с физическими лицами. TranzAxis Payment Gateway может формировать и передавать сообщение в адрес физического лица через электронную почту и SMS. Правила формирования сообщения определяется политикой нотификации клиентов.

## Обзор существующих решений

В настоящее время электронная коммерции очень интенсивно и динамично развивается, поэтому растет спрос на сервисы обработки финансовых транзакций. Ниже будут рассмотрены платежные шлюзы широко известных компаний.

Authorize.Net

Организация Authorize.Net предоставляет услуги по обработке платежей как марчантам, так и обычным пользователям, которые могут принимать платежи при помощи кард-ридера и виртуального терминала. Основные функции и возможности, которые предоставляет Authorize.Net:

1. Обработка электронных чеков с банковских счетов.
2. Обработка платежей, полученных с мобильного устройства.
3. Обработка платежей, при помощи виртуального терминала.
4. Формирование финансовых отчетов.

Преимуществом использования платежной системы Authorize.Net является генерация финансовых отчетов по прошедшим транзакциям, что в свою очередь экономит время мерчанта при анализе прибыли интернет-магазина. К плюсам использования данной системы также можно отнести возможность взаимодействия с виртуальным терминалом. Виртуальный терминал – программное решение, позволяющее мерчанту принимать оплату от клиентов по телефону или электронной почте. Данная технология позволяет быстро совершать платежи и отслеживать ход выполнения транзакций в режиме реального времени.

PaySpace

Международная организация, предоставляющая услуги электронной коммерции в сфере B2B. Основные функции и возможности PaySpace:

1. Автоматические платежи.
2. Оспаривание процедуры возврата денежных средств клиенту.
3. Осуществление выплат на счет в любом банке.

Главным преимуществом PaySpace является эффективное предотвращение мошеннических операций, для этого было разработано уникальное программное обеспечение. Также платежный сервис PaySpace предлагает свои услуги low-risk и high-risk индустриям.

## Цель и задачи выпускной квалификационной работы

Платежный шлюз TranzAxis Payment Gateway – продукт, являющийся собственной разработкой компании «Компас Плюс», для того чтобы платежные сервисы могли с ним взаимодействовать требуется разработка программного интерфейса. Помимо взаимодействия с платежным шлюзом, еще одним преимуществом использования REST API является формат обмена данными JSON, который в настоящее время очень широко используется, в отличие от XML, являющимся основным при использовании протокола SOAP.

Целью работы является совершенствование процесса взаимодействия платежных сервисов с системой TranzAxis Payment Gateway, за счет разработки программного интерфейса REST API. В разрабатываемом программном интерфейсе предусмотрен обмен данными в формате JSON.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать использование платежных систем в сервисах электронной коммерции, изучить систему TranzAxis Payment Gateway.
2. Cпроектировать и разработать программный интерфейс для взаимодействия интернет-магазинов с платежной системой TranzAxis Payment Gateway.
3. Произвести опытную эксплуатацию программного интерфейса для взаимодействия интернет-магазинов с платежной системой TranzAxis Payment Gateway.

## Выводы по первой главе

В первой главе выпускной квалификационной работы было рассмотрено такое понятие как электронная коммерция, описание структуры создания системы для ее реализации, а также виды взаимодействия между участниками сделки. Одной из таких систем является платежный шлюз, который позволяет обрабатывать финансовые транзакции, а для взаимодействия с ним необходим программный интерфейс.

В данной главе был рассмотрен платежный шлюз TranzAxis Payment Gateway, его функциональные возможности, архитектура и основные объекты, а также был приведен пример типового сценария ордера на выполнение коммунального платежа.

Также в данной главе рассмотрены аналоги систем электронной коммерции такие как Authorize.Net и PaySpace. Данные системы обладают рядом преимуществ и являются лидерами на рынке, но система TranzAxis Payment Gateway предлагает мерчантам функционал, не уступающий в своих возможностях, а также моментальную обработку транзакций.

По итогу главы была поставлена цель выпускной квалификационной работы и сформированы задачи, которые необходимо решить для достижения поставленной цели.

# ПРОЕКТНЫЕ РЕШЕНИЯ РАЗРАБОТКИ ПРОГРАММНОГО ИНТЕРФЕЙСА ДЛЯ ВЗАИМОДЕЙСТВИЯ ИНТЕРНЕТ-МАГАЗИНОВ С ПЛАТЕЖНОЙ СИСТЕМОЙ TRANZAXIS PAYMENT GATEWAY

## Описание функциональных требований, предъявляемых к разрабатываемому программному интерфейсу

Программный продукт представляет собой REST API, который обеспечивает взаимодействие между мерчантом и системой TranzAxis Payment Gateway.

Программный продукт должен быть реализован преимущественно на платформе Node.js. Обращение торгово-сервисного предприятия помимо TLS защищено HTTP Basic Auth. Логин и пароль торговой точки передаются в кодировке base64. Доступ к системе может быть только у авторизованных пользователей.

Чувствительные данные (PAN) могут быть зашифрованы публичным ключом, который хранится в Frontend Server. Взаимодействие с платежным шлюзом TranzAxis Payment Gateway происходит по протоколу SOAP.

Торговая точка создает ордер и перенаправляет клиента на страницу ввода данных необходимых для совершения платежа. Обязательными параметрами запроса после создания ордера являются:

* ID – идентификатор ордера;
* PASSWORD – пароль ордера.

После отправки запроса, HPP совершает следующие действия:

1. Проверяет подпись в пароле ордера (для защиты от DDOS).
2. Читает ордер, а также связанные с ним настройки (GET /order/{id}).
3. Запрашивает у клиента и устанавливает токен (POST /order/{id}/set-src-token, /order/{id}/set-dst-token).
4. Выполняет согласование параметров сделки (POST /order/{id}/set-vendor-pay-params, /order/{id}/confirm-vals).
5. Выполняет аутентификацию клиента (например, для платежного метода "карта": POST /order/{id}/tds-areq, /order/{id}/tds-pares). Если аутентификацию пройти не удалось можно сменить токен и повторить аутентификацию; либо можно отменить ордер (/order/{id}/cancel с Status = Canceled).
6. Выполняет транзакцию (/order/{id}/exec-tran). Если транзакция отклонена можно сменитьтокен, повторить аутентификацию и транзакцию, либо отменить ордер (/order/{id}/cancel с Status = Declined).

## Описание выбранной платформы для разработки программного интерфейса

Программный интерфейс необходимо реализовать на платформе Node.js. Node.js – программная платформа, позволяющая транслировать не типизированный язык программирования JavaScript в машинный код. Для того чтобы JavaScript взаимодействовал с устройствами ввода/вывода, а также подключать внешние библиотеки, написанные на разных языках программирования, был реализован программный интерфейс, при помощи языка C++.

Платформа Node.js используется при разработке веб-приложений, в основном для реализации серверной части. В основе Node.js лежит асинхронность в сочетании с событийным подходом программирования. Событийно-ориентированное программирование базируется на выполнении кода в тот момент, когда поступил запрос из внешней среды или пока пользователь не начал взаимодействие с веб-приложением. Данный подход позволяет экономить ресурсы вычислительной техники, на которой развернуто веб-приложение.

Платформа Node.js обладает рядом преимуществ:

1. Использование языка JavaScript, который прост в восприятии и удобен для реализации объемных проектов.
2. Стандартная библиотека, которую предлагает платформа имеет широкий набор возможностей, а с выходом новых версий становится только лучше.
3. С каждым днем количество открытых библиотек, для разработки программных продуктов, интенсивно увеличивается.
4. Экономия времени, затрачиваемого на реализацию веб-приложения, благодаря простоте JavaScript.
5. Асинхронное программирование, ускоряющее работу программного кода.

## Структура программного интерфейса

В таблицах 2.1 – 2.10 предсатвлена структура программного интерфейса. Адрес API: <http://127.0.0.1:3000/>

Таблица 2.1 – Параметры запроса “Создать пользовательскую сессию”

|  |  |
| --- | --- |
| **POST (*psp/create-user-session*)**  **Создать пользовательскую сессию** | |
| Request | Response |
| **Authorization:**string <token **>**  **Content-type:** application/json **Body:**  {  "browserInfo": {  "browserAcceptHeader": string<заголовки браузера>  "browserLanguage": string <язык>  "browserTZ": string <часовой пояс>  "browserUserAgent": string <агентбраузера>  }  } | **1. Success: Status:** 200/OK  **Content-type**: application/json  **Body:**  {  "sessionId": string<id сессии>  "login": {  "id": int<id мерчанта>  "login": string<логин мерчанта>  "status": string<статус>  }  }  **2.Status:** 401 (Unauthorized)  **Content-type**: application/json  **Body:**  { |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
|  | "errorCode": "InvalidLogin",  "errorDescription": "Login not provided"  } |

Таблица 2.2 – Параметры запроса “Создать заказ”

|  |  |
| --- | --- |
| **POST (*order?terminalId*)**  **Создать заказ** | |
| Request | Response |
| **Authorization:**string <token **>**  **Content-type:** application/json **Body:**  {  "order": {  "typeRid": string <Тип id>  "amount": int<сумма>  "currency": string<валюта>  "language": string<язык>  "title": string<названиезаказа>  "description": string<описание заказа>  }  } | **1. Success: Status:** 200/OK  **Content-type**: application/json  **Body:**  {  "order": {  "id": int<id заказа>  "password": string <парользаказа>  "status": string <статусзаказа>  }  }  **2.Status:** 401 (Unauthorized)  **Content-type**: application/json  **Body:**  {  "errorCode": "InvalidLogin",  "errorDescription": "Login not provided"  } |

Таблица 2.3 – Параметры запроса “Получить детали заказа”

|  |  |
| --- | --- |
| **GET (*order/:id?password*)**  **Id заказа, пароль заказа**  **Получить детали заказа** | |
| Request | Response |
|  | **1. Success: Status:** 200/OK  **Content-type**: application/json  **Body:**  {  "order": {  "id": int<id заказа>  "typeRid": string <типid>  "createTime": string<времясозданиязаказа>  "finishTime": string<время подтверждения заказа>  "status": string <статус>  "lastStatusLogin": string <логин>  "amount": int<сумма>  "currency": string <валюта>}  }  **2.Status:** 401  **Content-type**: application/json  **Body:**  {  "errorCode": "GeneralError",  "errorDescription": "can't process query response"  } |

Таблица 2.4 – Параметры запроса “Отменить заказ”

|  |  |
| --- | --- |
| **POST (*order/:id/cancel?password*)**  **Id заказа, пароль заказа**  **Отменить заказ** | |
| Request | Response |
| **Content-type:** application/json **Body:**  {  "order": {  "status": "Cancelled"  }  } | **1. Success: Status:** 200/OK  **Content-type**: application/json  **Body:**  {}  **2.Status:** 401  **Content-type**: application/json  **Body:**  {  "errorCode": "InvalidOrderState",  "errorDescription": " Cancel is not expected"  } |

Таблица 2.5 – Параметры запроса “Создать потребителя”

|  |  |
| --- | --- |
| **POST (*order/:id/create-consumer?password*)**  **Id заказа, пароль заказа**  **Создать потребителя** | |
| Request | Response |
| **Content-type:** application/json **Body:**  { | **1. Success: Status:** 200/OK  **Content-type**: application/json |

Продолжение таблицы 2.5

|  |  |
| --- | --- |
| "consumerLogin": {  "login": string <логинпользователя>  "password": string <парольпользователя>  } } | **Body:**  {  "consumer": {  "id": int<idпользователя>  "rid": int<уникальныйid>  "status": string <статус>  }  }  **2.Status:** 401  **Content-type**: application/json  **Body:**  {  "errorCode": "NotUnique",  "errorDescription": "Login is not unique"  } |

Таблица 2.6 – Параметры запроса “Получить информацию о потребителе”

|  |  |
| --- | --- |
| **GET (*consumer/:rid*)**  **Уникальный id пользователя**  **Получить информацию о потребителе** | |
| Request | Response |
|  | **1. Success: Status:** 200/OK  **Content-type**: application/json  **Body:**  { |

Продолжение таблицы 2.6

|  |  |
| --- | --- |
|  | "consumer": {  "id": int<idпользователя>  "rid": int<уникальныйid>  "status": string <статус>  }  }  **2.Status:** 401  **Content-type**: application/json  **Body:**  {  "errorCode": "GeneralError",  "errorDescription": "can't process query response"  } |

Таблица 2.7 – Параметры запроса “Создать сессию сервиса токенизации”

|  |  |
| --- | --- |
| **POST (*order/:id/create-cts-session?password*)**  **Id заказа, пароль заказа**  **Создать сессию сервиса токенизации** | |
| Request | Response |
| **Content-type:** application/json **Body:**  {  "cts": string <сервистокенизации>  "validationUrl": string <адрессервера> } | **1. Success: Status:** 200/OK  **Content-type**: application/json  **Body:**  {}  **2.Status:** 401  **Content-type**: application/json |

Продолжение таблицы 2.7

|  |  |
| --- | --- |
|  | **Body:**  {  "errorCode": "InvalidOrderState",  "errorDescription": "Order status must be Preparing or WaitPushTran"  } |

Таблица 2.8 – Параметры запроса “Установить исходный токен”

|  |  |
| --- | --- |
| **POST (*order/:id/set-src-token?password*)**  **Id заказа, пароль заказа**  **Установить исходный токен** | |
| Request | Response |
| **Content-type:** application/json **Body:**  {  "token": {  "storedId": int<сохраненныйid>  } } | **1. Success: Status:** 200/OK  **Content-type**: application/json  **Body:**  {  "order": {  "status": string <статус>  "tdsV2AuthStatus": string <статусаутентификацииV2>"tdsV1AuthStatus": string <статусаутентификацииV1>  "cvv2AuthStatus": string <статусаутентификацииcvv2>  "otpAutStatus": string<статусодноразовогокода> |

Продолжение таблицы 2.8

|  |  |
| --- | --- |
|  | "aReq": {}  }  }  **2.Status:** 401  **Content-type**: application/json  **Body:**  {  "errorCode": "InvalidOrderState",  "errorDescription": "Order status must be Preparing or WaitPushTran"  } |

Таблица 2.9 – Параметры запроса “Отправить запрос 3-D Secure”

|  |  |
| --- | --- |
| **POST (*order/:id/tds-areq?password*)**  **Id заказа, пароль заказа**  **Отправить запрос 3-D Secure** | |
| Request | Response |
| **Content-type:** application/json **Body:**  {  "aReq": {  } } | **1. Success: Status:** 200/OK  **Content-type**: application/json  **Body:**  {  "aRes": {  }  }  **2.Status:** 401  **Content-type**: application/json |

Продолжение таблицы 2.9

|  |  |
| --- | --- |
|  | **Body:**  {  "errorCode": "InvalidOrderState",  "errorDescription": "Order status must be Preparing or WaitPushTran"  } |

Таблица 2.10 – Параметры запроса “Получить ответ 3-D Secure”

|  |  |
| --- | --- |
| **POST (*order/:id/tds-pares?password*)**  **Idзаказа, пароль заказа**  **Получить ответ 3-D Secure** | |
| Request | Response |
| **Content-type:** application/json **Body:**  **{**  "paRes": string <токен>  **}** | **1. Success: Status:** 200/OK  **Content-type**: application/json  **Body:**  {}  **2.Status:** 401  **Content-type**: application/json  **Body:**  {  "errorCode": "InvalidOrderState",  "errorDescription": " TdsPARes is not expected"  } |

## Схема обработки запросов программного интерфейса

Программный интерфейс для взаимодействия интернет-магазинов с платежной системой TranzAxis Payment Gateway принимает два вида запросов: POST и GET.

При POST запросе обязательными параметрами являются ID заказа и пароль заказа, в случае их отсутствия в заголовке запроса должен содержаться параметр Basic Authorization. Помимо этих параметров в запросе должно содержаться тело запроса. В том случае, если хотя бы одно из условий не выполнено, система предупреждает о невозможности выполнения запроса, о несоответствии полученных от мерчанта данных ожидаемым, посредством возвращения ответа, содержащего данные: код ошибки, описание ошибки. Пример неуспешного запроса представлен на рисунке 2.1.

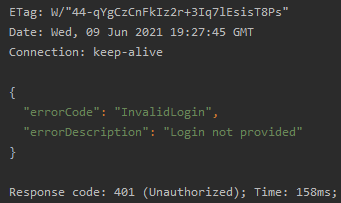


Рисунок 2.1 – Сообщение об ошибке

После проверки наличия параметров, программный интерфейс отправляет сформированный запрос к платежному шлюзу TranzAxis Payment Gateway по протоколу SOAP. Блок-схема обработки POST запроса изображена на рисунке 2.2.

При отправке GET запроса, необходимо указать параметры заказа ID и пароль, либо передать параметры авторизации, в заголовке запроса, после этого программный интерфейс из полученных данных формирует строку XML, которая будет отправлена платежному шлюзу TranzAxis Payment Gateway по протоколу SOAP. В ответе от бэкенд сервера будет получена строка XML, из полученных данных формируется JSON ответ и возвращается мерчанту. Блок-схема обработки GET запроса изображена на рисунке 2.3.

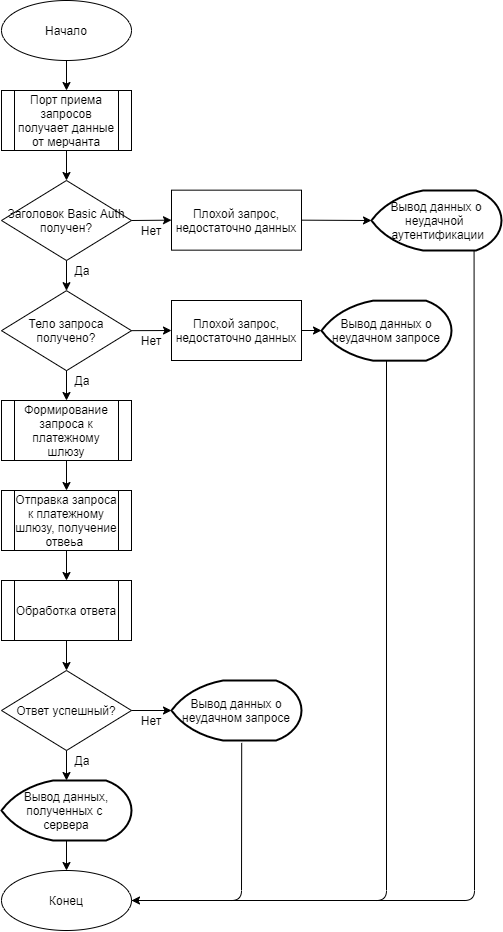


Рисунок 2.2 – Блок-схема обработки POST запроса

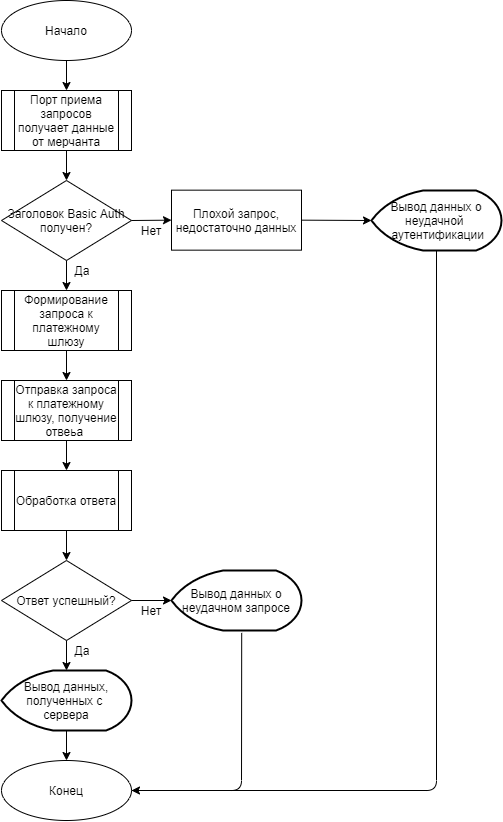


Рисунок 2.3 – Блок-схема обработки GET запроса

## Основные модули и функции работы программного интерфейса

Для приема запросов был реализован порт приема запросов (приложение А), который принимает POSTи GETзапросы. Функция обработки запроса принимает параметры:

* req – объект, содержащий параметры запроса;
* res – выходной поток данных.

Пример функции обработки запроса:

*router.post('', function(req, res, next) {  
 var body = req.body;  
 var terminalId = req.query.terminalId;  
 var userAuth = utils.basicAuth(req, res, next);  
  
 transaction.createOrder(body, terminalId, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});*

Данная функция принимает параметры запроса:

* body – тело запроса;
* terminalId – идентификатор терминала;
* userAuth – объект, содержащий логин и пароль мерчанта.

После получения всех параметров, они передаются в функцию обработки параметров запроса, которая расположена в модуле обработки транзакций (приложение Б). В данном примере это функция ceateOrder, пример функции обработки параметров запроса:

*function createOrder (body, terminalId, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN, tXPGUserSessionID, txpgFesUserSessionId) {  
 body.credentials = utils.getCredentials(userAuth);  
 body.terminal = {id: terminalId};  
  
 return new Promise(function(resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'createOrder')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
}*

Данная функция формирует объекты, в свойства которых помещает данные, полученные от мерчанта:

* credentials – объект, содержащий данные, необходимые для авторизации мерчаната;
* terminal – объект, содержащий данные о терминале.

Сформированный объект body передается в функцию getResponse, которая находится в модуле utils (приложение В). Помимо body, в функцию передаются параметры:

* method – метод SOAP запроса;
* reqType – тип SOAP запроса.

Фрагмент функции получения ответа:

*function getResponse (body, method, reqType) {  
 return new Promise(function (resolve, reject) {  
 var reqToTxpgObj = {};  
 var mapWrapperName = lowFirstSymb(method);  
 var wsdlUrl = 'http://schemas.radixware.org/aes.wsdl';  
 var bodyMapRq = require(`../request-map/${reqType}Rq.json`);  
 var wrapperMapRq = require(`../request-map/${mapWrapperName}Rq.json`);  
 var reqTxpgBodyObj = getDataFromRequest(body, bodyMapRq);  
 var wrapperData = {};  
  
 switch (method) {  
 case 'ExecuteAction':  
 wrapperData = {  
 actionType: 'R',  
 timeToEnd: '60000',  
 timeToStart: '30000',  
 actionRequest: createObjFromDataExact(reqTxpgBodyObj.data),  
 };  
 break;  
 case 'Query':  
 var type = bodyMapRq.rootPath.path;  
 var url = bodyMapRq.rootPath.url;  
  
 wrapperData = {  
 queryType: `{${url}}${type}`,  
 selection: createObjFromDataExact(reqTxpgBodyObj.data),  
 };  
 break;  
 default:  
 }  
  
 var nameSpace = reqTxpgBodyObj.nameSpace;  
 nameSpace.push({url: wsdlUrl, name: getWsdlName(wsdlUrl)});  
  
 var reqTxpgWrapperObj = getDataFromRequest(wrapperData, wrapperMapRq, nameSpace, {actionRequest: true, selection: true});  
 nameSpace = reqTxpgWrapperObj.nameSpace;  
 reqToTxpgObj = createObjFromDataExact(reqTxpgWrapperObj.data);*

Функция getResponse парсит объект body и сопоставляет его с объектом из map-файла, который соответствует типу SOAP запроса. По данным, полученным из этих двух объектов, формируется массив объектов, элементы которого содержат свойства:

1. path – массив, содержащий названия тегов XML запроса. Индекс массива соответствует уровню вложенности тега, последний элемент является контейнером для значения, полученного из тела запроса. Если первым символом последнего тега является “@”, в таком случае алгоритм определяет, что это атрибут.
2. val – значение полученное из объекта body.
3. item – число, определяющее индекс элемента списка в структуре XML дерева. Если индекс равен “-1”, данный элемент находится в обычном объекте.

Фрагмент массива объектов:

*[*

*{*

*"path": [*

*"api:CreateOrder",*

*"api:Request",*

*"api:Order",*

*"@TypeRid"*

*],*

*"val": "Ord11",*

*"item": -1*

*},*

*{*

*"path": [*

*"api:CreateOrder",*

*"api:Request",*

*"api:Order",*

*"@Amt"*

*],*

*"val": 100,*

*"item": -1*

*},*

*{*

*"path": [*

*"api:CreateOrder",*

*"api:Request",*

*"api:Order",*

*"@Ccy"*

*],*

*"val": "USD",*

*"item": -1*

*},*

Из сформированного массива объектов формируется объект JSON, который затем будет помещен в SOAP обертку и преобразован в XML:

*var reqToTxpgXml = '<?xml version=\'1.0\' encoding=\'UTF-8\'?>' +  
 '<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">' +  
 '<soapenv:Header/>' +  
 '<soapenv:Body>' +  
 `<aes:${method}` + getNameSpaces(nameSpace) + '>' +  
 res +  
 `</aes:${method}>` +  
 '</soapenv:Body>' +  
 '</soapenv:Envelope>';*

Сформированный запрос отправляется к TranzAxis Payment Gateway, после чего полученный ответ парситься и сопоставляется с соответствующим ему map-файлом ответа, в результате формируется JSON ответ и возвращается мерчанту. В том случае если запрос будет неуспешным, возникнет исключительная ситуация и API вернет ответ представленный на рисунке 2.3.

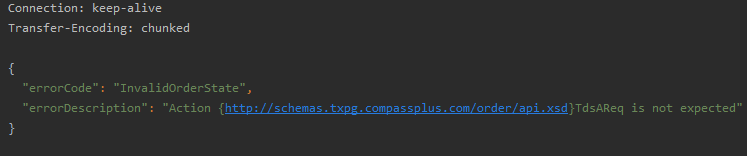


Рисунок 2.4 – Сообщение об ошибке, полученное от платежного шлюза

Основные функции, использованные в методе getResponse:

1. getDataFromRequest – функция, которая формирует массив объектов из параметров запроса. Она принимает следующие параметры:
   * + - * objFromBody – тело запроса;
         * objFromMap – map-файл запроса.

Данная функция рекурсивно проходит по всем элементам объекта и добавляет их в массив. Вместе с элементами алгоритм определяет пространство имен для каждого элемента и добавляет их к названиям тегов.

* + 1. createObjFromDataExact – функция, формирующая объект из массива, полученного методом getDataFromRequest. Она принимает следующие параметры:
       - * data – массив, сформированный из запроса;
         * objType – типобъекта (request или response).

Данная функция формирует из каждого элемента массива объект и формирует из них один глобальный объект, путем добавления новых свойств.

* + 1. getDataFromResponse – функция, формирующая массив из объекта, полученного в ответе от TranzAxis Payment Gateway.

## Выводы по второй главе

Во второй главе выпускной квалификационной работы были рассмотрены требования к разрабатываемому программному интерфейсу и выбрана платформа для разработки программного продукта. Исходя из требований к API в быстроте обработки запросов и простоте проектирования и реализации программного кода, была выбрана платформа Node.js.

В ходе работы была спроектирована структура программного интерфейса, в которой описаны все запросы, принимаемые API, для каждого были указаны тип запроса, его параметры и свойства тела запроса. Также были спроектированы схемы обработки POST и GET запросов и представлены в виде блок-схем, что позволило наглядно понять поведение конечного продукта. В данной главе также рассмотрены основные модули и функции программного интерфейса, это мощный инструмент, который позволяет обрабатывать огромные массивы данных древовидной структуры элементов любой вложенности и укладывается в минимальные сроки.

# РЕЗУЛЬТАТЫ ОПЫТНОЙ ЭКСПЛУАТАЦИИ ПРОГРАММНОГО ИНТЕРФЕЙСА ДЛЯ ВЗАИМОДЕЙСТВИЯ ИНТЕРНЕТ-МАГАЗИНОВ С ПЛАТЕЖНОЙ СИСТЕМОЙ TRANZAXIS PAYMENT GATEWAY

## Описание программного интерфейса

Работа пользователя с программным интерфейсом для взаимодействия интернет-магазинов с платежной системой TranzAxis Payment Gateway начинается с создания пользовательской сессии. Для того чтобы отправить запрос “Создать пользовательскую сессию”, необходимо указать адрес и тело запроса, представленные на рисунке 3.1.

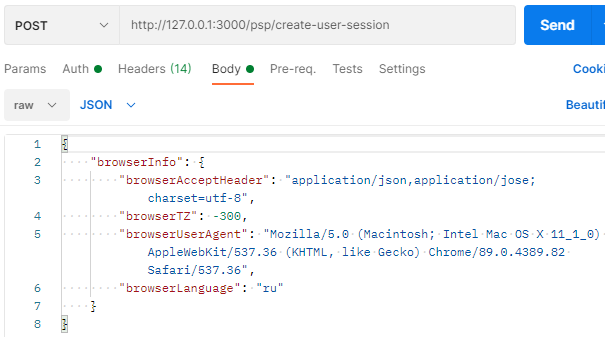


Рисунок 3.1 – Тело запроса “Создать пользовательскую сессию”

В верхней части рисунков, для всех примеров запросов, расположен адрес запроса, ниже указаны все необходимые параметры. В теле запрос “Создать пользовательскую сессию” пользователю необходимо указать информацию об используемом браузере. После отправки запроса, пользователь получает ответ, представленный на рисунке 3.2.



Рисунок 3.2 – Ответ на запрос “Создать пользовательскую сессию”

В ответе приходят данные о мерчанте и созданной сессии.

Следующий шагом, пользователю необходимо создать заказ, для этого нужно отправить запрос “Создать заказ”, с параметрами, телом запроса и параметрами авторизации, которые находятся в заголовке запроса, представленными на рисунках 3.3 – 3.5.

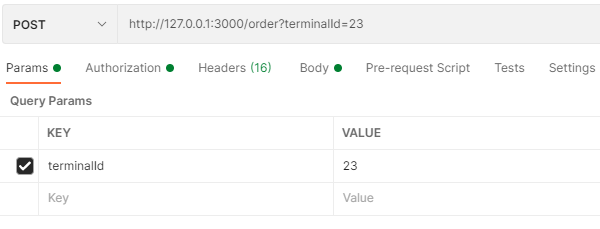


Рисунок 3.3 – Параметры запроса “Создать заказ”



Рисунок 3.4 – Тело запроса “Создать заказ”

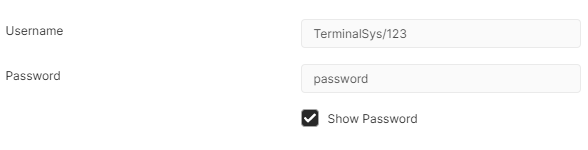


Рисунок 3.5 – Параметры аторизации запроса “Создать заказ”

В теле запрос “Создать заказ” пользователю необходимо указать данные о заказе, а для формирования заголовка авторизации логин и пароль мерчанта. После отправки запроса, пользователь получает ответ, представленный на рисунке 3.6.

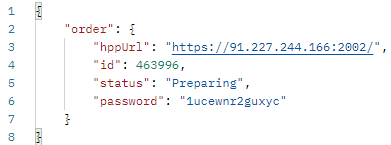


Рисунок 3.6 – Ответ на запрос “Создать заказ”

В ответе приходят данные о созданном заказе. Параметры id и password необходимо использовать, при отправке следующих запросов.

После того как заказ создан, пользователь может получить детали заказа, для этого нужно отправить запрос “Получить детали заказа” с параметрами, представленными на рисунке 3.7.

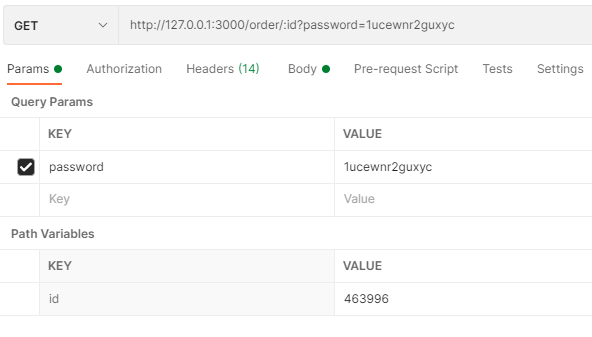


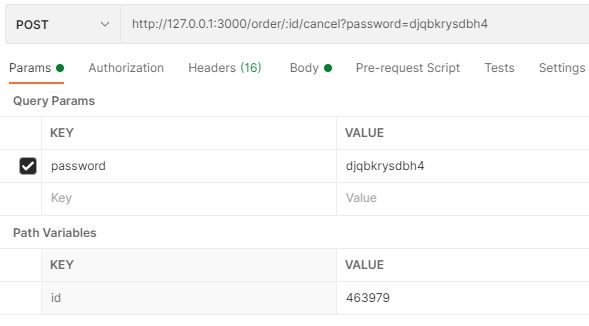
Рисунок 3.7 – Параметры запроса “Получить детали заказа”

После отправки запроса “Получить детали заказа”, пользователь получает ответ, представленный на рисунке 3.8.



Рисунок 3.8 – Ответ на запрос “Получить детали заказа”

В том случае если пользователю необходимо отменить заказ, нужно отправить запрос “Отменить заказ” с параметрами, представленными на рисунках 3.9 – 3.10.

Рисунок 3.9 – Параметры запроса “Отменить заказ”

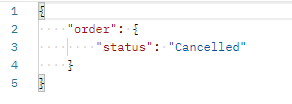


Рисунок 3.10 – Тело запроса “Отменить заказ”

После отправки запроса “Получить детали заказа”, пользователь получает ответ, представленный на рисунке 3.11.

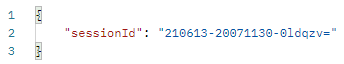


Рисунок 3.11 – Ответ на запрос “Отменить заказ”

Следующий шагом, пользователю необходимо создать потребителя (клиента интернет-магазина), для этого нужно отправить запрос “Создать потребителя”, с параметрами и телом запроса представленными на рисунках 3.12 – 3.13.

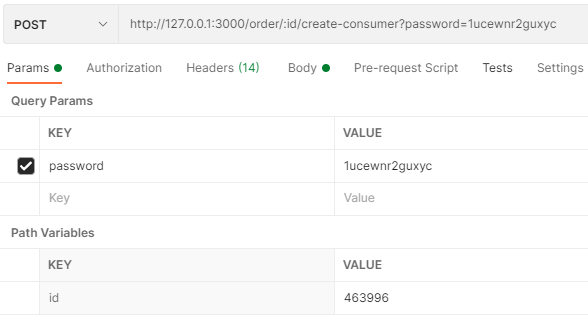


Рисунок 3.12 – Параметры запроса “Создать потребителя”

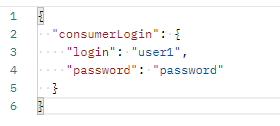


Рисунок 3.13 – Тело запроса “Создать потребителя”

После отправки запроса “Создать потребителя”, пользователь получает ответ, представленный на рисунке 3.14.



Рисунок 3.14 – Ответ на запрос “Создать потребителя”

Для того чтобы получить информацию о потребителе, нужно отправить запрос “Получить информацию о потребителе” с параметрами, представленными на рисунке 3.15.

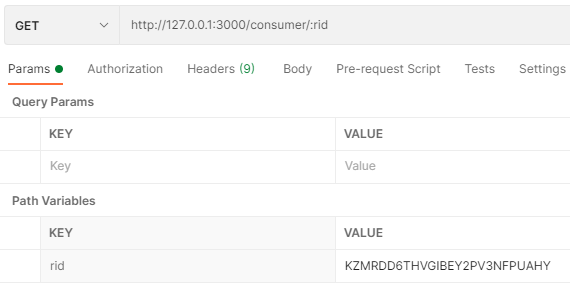


Рисунок 3.15 – Параметры запроса “Получить информацию о потребителе”

После отправки запроса “Получить информацию о потребителе”, пользователь получает ответ, представленный на рисунке 3.16.



Рисунок 3.16 – Ответ на запрос “Получить информацию о потребителе”

Для того чтобы пройти 3-D Secure аутентификацию, для начала нужно получить токен-источник, при помощи отправки запроса “Задать токен-источник” с параметрами, представленными на рисунке 3.17.

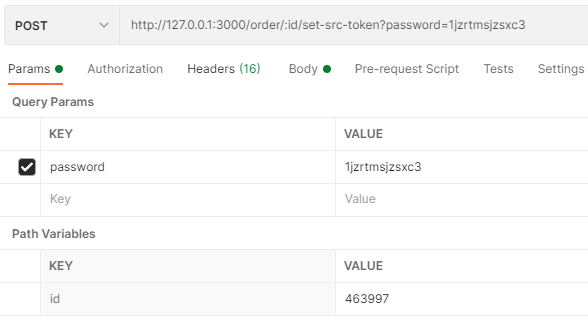


Рисунок 3.17 – Параметры запроса “Задать токен-источник”

После отправки запроса “Задать токен-источник”, пользователь получает ответ, представленный на рисунке 3.18.



Рисунок 3.18 – Ответ на запрос “Задать токен-источник”

После того как токен-источник сформирован, необходимо отправить запрос “Запрос на аутентификацию для 3-D Secure” с параметрами и телом запроса представленными на рисунках 3.19 – 3.20.

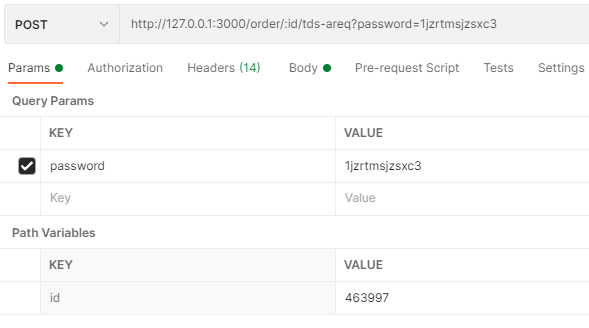


Рисунок 3.19 – Параметры запроса “Запрос на аутентификацию для 3-D Secure”



Рисунок 3.20 – Тело запроса “Запрос на аутентификацию для 3-D Secure”

После того как успешно выполнен запрос на аутентификацию, нужно отправить запрос “Обработка ответа аутентификации для 3-D Secure” с параметрами и телом запроса представленными на рисунках 3.21 – 3.22.

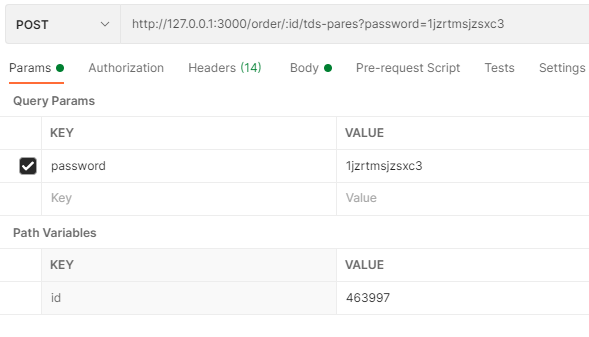


Рисунок 3.21 – Параметры запроса “Обработка ответа аутентификации для 3-D Secure”



Рисунок 3.22 – Тело запроса “Обработка ответа аутентификации для 3-D Secure”

После отправки запроса “Обработка ответа аутентификации для 3-D Secure”, пользователь получает ответ, представленный на рисунке 3.23.



Рисунок 3.23 – Ответ на запрос “Обработка ответа аутентификации для 3-D Secure”

## **Пример взаимодействия программного интерфейса с TranzAxis** Payment Gateway

Взаимодействие программного интерфейса с платежным шлюзом TranzAxis Payment Gateway будет рассмотрено на примере процедуры создания заказа. Сначала нужно отправить запрос программному интерфейсу. Чтобы отправить запрос “Создать заказ”, необходимо указать параметры запроса, параметры авторизации и тело запроса, представленные на рисунках 3.24 – 3.26.

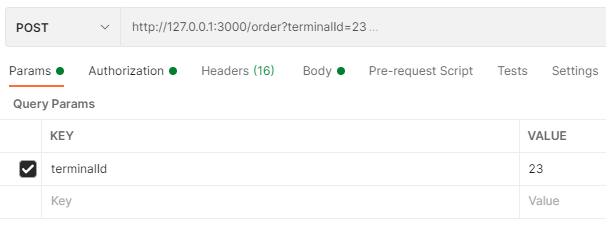


Рисунок 3.24 – Параметры запроса “Создать заказ”

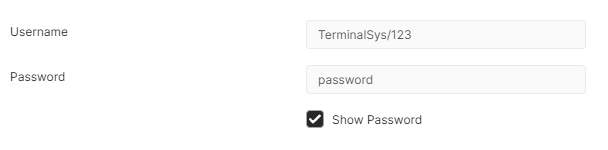


Рисунок 3.25 – Параметры авторизации запроса “Создать заказ”

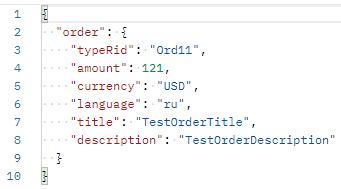


Рисунок 3.26 – Тело запроса “Создать заказ”

После того как программный интерфейс получил запрос “Создать заказ”, он формирует строку XML из полученных данных пользователя по XSD-схемам, в результате будет получена XML, представленная на рисунке 3.27.



Рисунок 3.27 – строка XML, сформированная программным интерфейсом из полученных данных пользователя

Следующим этапом процедуры создания заказа, программный интерфейс отправляет сформированную строку XML платежному шлюзу TranzAxis Payment Gateway и получает ответ в виде XML строки, представленной на рисунке 3.28.

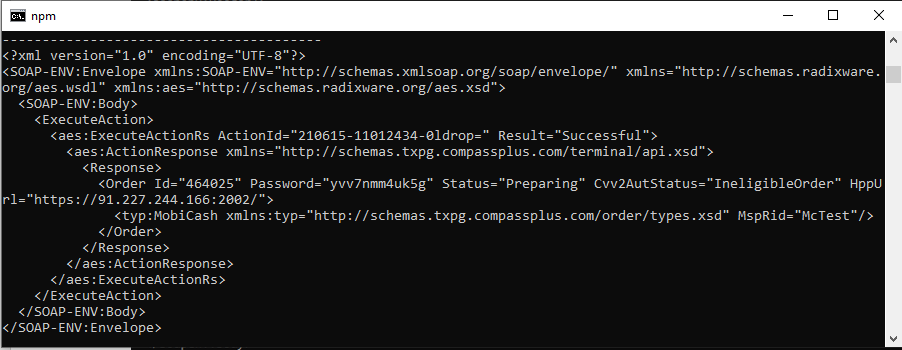


Рисунок 3.28 – строка XML, полученная от TranzAxis Payment Gateway

После того как ответ от платежного шлюза получен, программный интерфейс из полученных данных формирует ответ в формате JSON и возвращает его пользователю. Сформированный ответ представлен на рисунке 3.29.

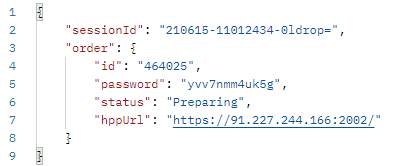


Рисунок 3.29 – ответ на запрос “Создать заказ”

## Выводы по третьей главе

В третьей главе выпускной квалификационной работы была рассмотрена опытная эксплуатация программного интерфейса для взаимодействия интернет-магазинов с платежной системой TranzAxis Payment Gateway, в частности были воспроизведены на примере такие процессы как: создание заказа, отмена заказа, создание пользовательской сессии, создание потребителя, 3-D Secure аутентификация.

Также в данной главе был рассмотрен пример взаимодействия программного интерфейса с платежным шлюзом TranzAxis Payment Gateway на примере транзакции создания заказа.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы был реализован программный интерфейс для взаимодействия интернет-магазинов с платежной системой TranzAxis Payment Gateway. В ходе выполнения работы была описана предметная область, выбрана платформа для разработки программного продукта, спроектирован и реализован функционал программного интерфейса и произведена опытная эксплуатация.

В первой главе выпускной квалификационной работы былы рассмотрены электронная коммерция и ее виды, аналоги систем электронной коммерции такие как Authorize.Net и PaySpace. Данные системы обладают рядом преимуществ и являются лидерами на рынке, но система TranzAxis Payment Gateway предлагает мерчантам функционал, не уступающий в своих возможностях, а также моментальную обработку транзакций, была поставлена цель выпускной квалификационной работы и сформированы задачи, которые необходимо решить для достижения поставленной цели.

Во второй главе выпускной квалификационной работы были рассмотрены требования к разрабатываемому программному интерфейсу и выбрана платформа для разработки программного продукта. Исходя из требований к API в быстроте обработки запросов, простоте проектирования и реализации программного кода, была выбрана платформа Node.js. В ходе работы была спроектирована структура программного интерфейса, в которой описаны все запросы, принимаемые API, для каждого были указаны тип запроса, его параметры и свойства тела запроса. Также были спроектированы схемы обработки POST и GET запросов и представлены в виде блок-схем, что позволило наглядно понять поведение конечного продукта. В данной главе также рассмотрены основные модули и функции программного интерфейса, это мощный инструмент, который позволяет обрабатывать огромные массивы данных древовидной структуры элементов любой вложенности и укладывается в минимальные сроки.

В третьей главе выпускной квалификационной работы была рассмотрена опытная эксплуатация программного интерфейса для взаимодействия интернет-магазинов с платежной системой TranzAxis Payment Gateway, в частности были воспроизведены на примере такие процессы как: создание заказа, отмена заказа, создание пользовательской сессии, создание потребителя, 3-D Secure аутентификация. Также было наглядно рассмотрено взаимодействие программного интерфейса с платежным шлюзом TranzAxis Payment Gateway на примере транзакции создания заказа. В результате опытной эксплуатации была продемонстрирована работа программного продукта, удовлетворяющая предъявленным требованиям, вследствии этого, цель выпускной квалификационной работы успешно достигнута.

СПИСОК ИСПОЛЬЗованных ИСТОЧНИКОВ

1. Electronic Commerce Reference. Справочное руководство [текст]: ООО «Компас Плюс»., 2020 – 33 с.
2. Платежный шлюз. Принцип работы платежного шлюза [Электронный ресурс]: habr. Режим доступа – https://habr.com/ru/post/496596.
3. TXPG-Overview. Общее описание системы [Текст]: ООО «Компас Плюс», 2021 – 21 с.
4. MPSP-Order. Описание ордеров и их компонентов [Текст]: ООО «Компас Плюс», 2019 – 33 с.
5. MPSP-Subject. Описание объектов платформы [Текст]: ООО «Компас Плюс», 2019 – 19 с.
6. MPSP-Authentication. Описание средств аутентификации токена [Текст]: ООО «Компас Плюс», 2019 – 6 с.
7. TXPG-Subject. Описание объектов платформы [Текст]: ООО «Компас Плюс», 2020 – 39 с.
8. TXPG-Order. Описание ордеров и их компонентов [Текст]: ООО «Компас Плюс», 2021 – 65 с.
9. TXPG-Frontend. Описание компонентов TranzAxis Payment Gateway [Текст]: ООО «Компас Плюс», 2021 – 11 с.
10. TXPG-Common. Описание объектов общего назначения [Текст]: ООО «Компас Плюс», 2021 – 11 с.
11. TXPG-API and Access. Описание архитектуры TXPG [Текст]: ООО «Компас Плюс», 2021 – 17.

ПРИЛОЖЕНИЕ А

Программный код модуля приема запросов “order”:

const express = require('express');  
const transaction = require('../scripts/transaction');  
const router = express.Router();  
const writer = require('../utils/writer.js');  
const utils = require('../utils/utils.js');  
  
/\*\*  
 \* Create order  
 \*\*/  
router.post('', function(req, res, next) {  
 var body = req.body;  
 var terminalId = req.query.terminalId;  
 var userAuth = utils.basicAuth(req, res, next);  
  
 transaction.createOrder(body, terminalId, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
  
/\*\*  
 \* Create consumer  
 \*\*/  
router.post('/:id/create-consumer', function(req, res, next) {  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
 var userAuth = '';  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.createConsumer(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
/\*\*  
 \* Create cts session  
 \*\*/  
router.post('/:id/create-cts-session', function(req, res, next) {  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
 var userAuth = '';  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.createCtsSession(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
/\*\*  
 \* Set src token  
 \*\*/  
router.post('/:id/set-src-token', function(req, res, next) {  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
 var userAuth = '';  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.setSrcToken(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
/\*\*  
 \* Tds aReq  
 \*\*/  
router.post('/:id/tds-areq', function(req, res, next) {  
 var userAuth = '';  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.tdsAReq(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
/\*\*  
 \* Tds paRes  
 \*\*/  
router.post('/:id/tds-pares', function(req, res, next) {  
 var userAuth = '';  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.tdsPaRes(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
/\*\*  
 \* Exec tran  
 \*\*/  
router.post('/:id/exec-tran', function(req, res, next) {  
 var userAuth = '';  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.execTran(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
/\*\*  
 \* Get order  
 \*\*/  
router.get('/:id', function(req, res, next) {  
 var userAuth = '';  
 var password = req.query.password;  
 var id = req.params.id;  
 var merchantId = req.query.merchantId;  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.getOrder(password, id, userAuth, merchantId)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
  
});  
  
/\*\*  
 \* Confirm vals  
 \*\*/  
router.post('/:id/confirm-vals', function(req, res, next) {  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
 var userAuth = '';  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.confirmVals(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
/\*\*  
 \* Upop secure otp req  
 \*\*/  
router.post('/:id/upop-secure-plus-otp-request', function(req, res, next) {  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
 var userAuth = '';  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.otpRequest(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
/\*\*  
 \* Upop secure otp verify  
 \*\*/  
router.post('/:id/upop-secure-plus-otp-verify', function(req, res, next) {  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
 var userAuth = '';  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.otpVerify(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
router.put('/:id', function(req, res, next) {  
 var id = req.params.id;  
  
 res.send(`id = ${id}`);  
});  
  
/\*\*  
 \* Cancel order  
 \*\*/  
router.post('/:id/cancel', function(req, res, next) {  
 var body = req.body;  
 var password = req.query.password;  
 var id = req.params.id;  
 var userAuth = '';  
  
 if (!id && !password) {  
 userAuth = utils.basicAuth(req, res, next);  
 }  
  
 transaction.cancelOrder(body, password, id, userAuth)  
 .then(function (response) {  
 writer.writeJson(res, response);  
 })  
 .catch(function (response) {  
 writer.writeJson(res, response);  
 });  
});  
  
router.post('/generate-map', function(req, res, next) {  
 transaction.generateMap();  
 res.send('successfully');  
});  
  
module.exports = router;

ПРИЛОЖЕНИЕ Б

Программный код модуля обработки запросов “transaction”:

const utils = require('../utils/utils.js');  
const generate = require('../utils/generateMap.js');  
  
module.exports = (function () {  
  
 function createOrder (body, terminalId, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN, tXPGUserSessionID, txpgFesUserSessionId) {  
 body.credentials = utils.getCredentials(userAuth);  
 body.terminal = {id: terminalId};  
  
 return new Promise(function(resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'createOrder')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function cancelOrder(body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN, tXPGUserSessionID, txpgFesUserSessionId) {  
 body.order = {id: id, password: password};  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'cancelOrder')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function getOrder(password, id, userAuth, merchantId, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN, tXPGUserSessionID, txpgFesUserSessionId) {  
 var body = {id: id, password: password, orderDetailLevel: 2, tokenDetailLevel: 2, ctsDetailLevel: 1, merchantDetailLevel: 2};  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'Query', 'getOrder')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function createConsumer (body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (body.order) {  
 body.order = Object.assign(body.order, {id: id, password: password});  
 } else {  
 body.order = {id: id, password: password};  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 if (body.consumerLogin) {  
 body.consumerLogin.password = utils.getPwdHash(body.consumerLogin.login, body.consumerLogin.password);  
 }  
  
 body.consumer = '';  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'createConsumer')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function createCtsSession (body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (body.order) {  
 body.order = Object.assign(body.order, {id: id, password: password});  
 } else {  
 body.order = {id: id, password: password};  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'createCtsSession')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function getConsumer (rid, userAuth, ownerKind, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 //var body = {rid: rid, consumerDetailLevel: 2, tokenDetailLevel: 2, devDetailLevel: 1, loginDetailLevel: 2};  
 var body = {rid: rid, consumerDetailLevel: 2, tokenDetailLevel: 2, devDetailLevel: 1, loginDetailLevel: 2};  
  
 if (ownerKind) {  
 body.kind = ownerKind;  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'Query', 'getConsumer')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function setSrcToken (body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (body.order) {  
 body.order = Object.assign(body.order, {id: id, password: password});  
 } else {  
 body.order = {id: id, password: password};  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'setSrcToken')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function confirmVals (body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (body.order) {  
 body.order = Object.assign(body.order, {id: id, password: password});  
 } else {  
 body.order = {id: id, password: password};  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'confirmVals')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function otpRequest (body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (body.order) {  
 body.order = Object.assign(body.order, {id: id, password: password});  
 } else {  
 body.order = {id: id, password: password};  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'otpRequest')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function otpVerify (body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (body.order) {  
 body.order = Object.assign(body.order, {id: id, password: password});  
 } else {  
 body.order = {id: id, password: password};  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'otpVerify')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function execTran (body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (body.order) {  
 body.order = Object.assign(body.order, {id: id, password: password});  
 } else {  
 body.order = {id: id, password: password};  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'execTran')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function tdsPaRes (body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (body.order) {  
 body.order = Object.assign(body.order, {id: id, password: password});  
 } else {  
 body.order = {id: id, password: password};  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'tdsPaRes')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function tdsAReq (body, password, id, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (body.order) {  
 body.order = Object.assign(body.order, {id: id, password: password});  
 } else {  
 body.order = {id: id, password: password};  
 }  
  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'tdsAReq')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function createUserSession (body, userAuth, tXPGTLSClientSerial, tXPGTLSClientCN, tXPGTLSIssuerCN) {  
 if (userAuth) {  
 body.credentials = utils.getCredentials(userAuth);  
 }  
  
 return new Promise(function (resolve, reject) {  
 utils.getResponse(body, 'ExecuteAction', 'createUserSession')  
 .then(function (response) {  
 resolve(response);  
 }).catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function generateMap() {  
   
 }  
  
 return {  
 createOrder: createOrder,  
 cancelOrder: cancelOrder,  
 generateMap: generateMap,  
 getOrder: getOrder,  
 setSrcToken: setSrcToken,  
 confirmVals: confirmVals,  
 tdsAReq: tdsAReq,  
 createUserSession: createUserSession,  
 createConsumer: createConsumer,  
 getConsumer: getConsumer,  
 tdsPaRes: tdsPaRes,  
 execTran: execTran,  
 createCtsSession: createCtsSession,  
 otpRequest: otpRequest,  
 otpVerify: otpVerify,  
 };  
})();

ПРИЛОЖЕНИЕ В

Программный код модуля “utils”:

const xmlBuilder = require('xmlbuilder');  
const xml2js = require('xml2js');  
const yaml = require('yaml');  
const jsYaml = require('js-yaml');  
const sha256 = require('js-sha256').sha256;  
const axios = require('axios');  
  
module.exports = (function () {  
 function getResponse (body, method, reqType) {  
 return new Promise(function (resolve, reject) {  
 var reqToTxpgObj = {};  
 var mapWrapperName = lowFirstSymb(method);  
 var wsdlUrl = 'http://schemas.radixware.org/aes.wsdl';  
 var bodyMapRq = require(`../request-map/${reqType}Rq.json`);  
 var wrapperMapRq = require(`../request-map/${mapWrapperName}Rq.json`);  
 var reqTxpgBodyObj = getDataFromRequest(body, bodyMapRq);  
 var wrapperData = {};  
 resolve(reqTxpgBodyObj.data);  
  
 switch (method) {  
 case 'ExecuteAction':  
 wrapperData = {  
 actionType: 'R',  
 timeToEnd: '60000',  
 timeToStart: '30000',  
 actionRequest: createObjFromDataExact(reqTxpgBodyObj.data),  
 };  
 break;  
 case 'Query':  
 var type = bodyMapRq.rootPath.path;  
 var url = bodyMapRq.rootPath.url;  
  
 wrapperData = {  
 queryType: `{${url}}${type}`,  
 selection: createObjFromDataExact(reqTxpgBodyObj.data),  
 };  
 break;  
 default:  
 }  
  
 var nameSpace = reqTxpgBodyObj.nameSpace;  
 nameSpace.push({url: wsdlUrl, name: getWsdlName(wsdlUrl)});  
  
 var reqTxpgWrapperObj = getDataFromRequest(wrapperData, wrapperMapRq, nameSpace, {actionRequest: true, selection: true});  
 nameSpace = reqTxpgWrapperObj.nameSpace;  
 reqToTxpgObj = createObjFromDataExact(reqTxpgWrapperObj.data);  
  
 jsonToXml(reqToTxpgObj)  
 .then(function (res) {  
 var reqToTxpgXml = '<?xml version=\'1.0\' encoding=\'UTF-8\'?>' +  
 '<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">' +  
 '<soapenv:Header/>' +  
 '<soapenv:Body>' +  
 `<aes:${method}` + getNameSpaces(nameSpace) + '>' +  
 res +  
 `</aes:${method}>` +  
 '</soapenv:Body>' +  
 '</soapenv:Envelope>';  
  
 console.log(reqToTxpgXml);  
 console.log('----------------------------------------');  
  
 sendReqToTxpg('http://10.7.1.55:20103', reqToTxpgXml)  
 .then(function (res) {  
 console.log(res.data);  
 var dataXml = res.data;  
 var responseFromTxpg = xmlToJson(dataXml)['Envelope']['Body'][method];  
 var bodyMapRs = require(`../request-map/${reqType}Rs.json`);  
 var wrapperMapRs = require(`../request-map/${mapWrapperName}Rs.json`);  
 var dataFromResponse = getDataFromResponse(responseFromTxpg, bodyMapRs, wrapperMapRs);  
 var response = createObjFromDataExact(dataFromResponse, 'response').root;  
  
 if (response) {  
 if (response.err) {  
 if (!response.err.errorCode) {  
 response.err.errorCode = 'ServiceError';  
 }  
 resolve(response.err);  
 }  
  
 if (response.result === 'Failed') {  
 resolve({  
 errorCode: "ServiceError"  
 });  
 }  
  
 delete response.result;  
  
 resolve(response);  
 } else {  
 resolve({  
 errorCode: "GeneralError",  
 errorDescription: "can't process query response"  
 });  
 }  
 }).catch(function (err) {  
 console.log(err);  
 resolve({  
 errorCode: "ServiceError"  
 });  
 });  
 })  
 .catch(function (err) {  
 reject(err);  
 });  
 });  
 }  
  
 function getDataFromResponse(response, bodyMap, wrapperMap) {  
 var data = [];  
 var numberOfElementsInArr = -1;  
 var elemNameReplacement = '';  
 var substrReplacement = '@#';  
 var arrStartIndices = [];  
 var arrItemNumber = [];  
  
 parseResponse(response, wrapperMap);  
 return data;  
  
 function parseResponse(objFromResponse, objFromMap, itemNumber) {  
 for(var elemName in objFromResponse) {  
 if (Array.isArray(objFromResponse[elemName])) {  
 if (objFromMap[elemName]) {  
 if (numberOfElementsInArr !== -1 && !objFromMap['dontCreateArr']) {  
 arrStartIndices.push(data.length);  
 arrItemNumber.push(itemNumber);  
 } else if (!objFromMap['dontCreateArr']) {  
 numberOfElementsInArr = 0;  
 }  
 var dontCreateArr = false;  
  
 objFromResponse[elemName].forEach(function (obj, index) {  
 if (objFromMap['dontCreateArr']) {  
 dontCreateArr = true;  
 parseResponse(obj, objFromMap[elemName],-1);  
 } else {  
 parseResponse(obj, objFromMap[elemName], index);  
 }  
 });  
  
 if (!dontCreateArr) {  
 addArrLengthInData();  
 }  
 }  
 } else if (typeof objFromResponse[elemName] === 'object') {  
 if (elemName === '$') {  
 parseResponse(objFromResponse[elemName], objFromMap, itemNumber);  
 } else {  
 if (objFromMap[elemName]) {  
 if (objFromMap[elemName] === 'bodyMap') {  
 parseResponse(objFromResponse[elemName], bodyMap);  
 } else if (objFromMap[elemName]['createArr']) {  
 if (numberOfElementsInArr !== -1) {  
 arrStartIndices.push(data.length);  
 arrItemNumber.push(itemNumber);  
 } else {  
 numberOfElementsInArr = 0;  
 }  
  
 parseResponse(objFromResponse[elemName], objFromMap[elemName], 0);  
  
 addArrLengthInData();  
 } else {  
 parseResponse(objFromResponse[elemName], objFromMap[elemName], itemNumber);  
 }  
 }  
 }  
 } else {  
 if (elemName.indexOf('xmlns') === -1 && objFromMap[elemName] && objFromResponse[elemName] || elemName === '\_') {  
 if (objFromMap[elemName] === substrReplacement) {  
 elemNameReplacement = lowFirstSymb(objFromResponse[elemName]);  
 } else {  
 var strPath = '';  
 var val = objFromResponse[elemName];  
  
 if (elemName === '\_') {  
 strPath = objFromMap;  
 } else {  
 strPath = objFromMap[elemName];  
 }  
  
 if (strPath.indexOf(substrReplacement) !== -1) {  
 strPath = substringReplacement(strPath, substrReplacement, elemNameReplacement);  
 }  
  
 if (strPath.indexOf('{}') !== -1) {  
 val = JSON.parse(val);  
 strPath = substringReplacement(strPath, '{}', '');  
 }  
  
 if (strPath.indexOf('[]') !== -1) {  
 val = val.split(' ');  
 strPath = substringReplacement(strPath, '[]', '');  
 }  
  
 if (itemNumber === undefined) {  
 itemNumber = -1;  
 } else if (itemNumber !== -1) {  
 numberOfElementsInArr++;  
 }  
  
 var path = ('root.' + strPath).split('.');  
  
 if (arrItemNumber.length > 0) {  
 var buf = [];  
 buf = arrItemNumber.slice(0);  
 buf.push(itemNumber);  
  
 data.push({path: path, val: val, item: buf});  
 } else {  
 data.push({path: path, val: val, item: itemNumber});  
 }  
 }  
 }  
 }  
 }  
 }  
  
 function addArrLengthInData () {  
 if (arrStartIndices.length === 0) {  
 if (data[data.length - numberOfElementsInArr].arrLength) {  
 var arrLength = data[data.length - numberOfElementsInArr].arrLength;  
  
 if (Array.isArray(arrLength)) {  
 data[data.length - numberOfElementsInArr].arrLength.unshift(numberOfElementsInArr);  
 } else {  
 data[data.length - numberOfElementsInArr].arrLength = [numberOfElementsInArr, arrLength];  
 }  
 } else {  
 data[data.length - numberOfElementsInArr].arrLength = numberOfElementsInArr;  
 }  
  
 numberOfElementsInArr = -1;  
 arrStartIndices = [];  
 arrItemNumber = [];  
 } else {  
 if (data[arrStartIndices[arrStartIndices.length-1]].arrLength) {  
 var arrLength = data[arrStartIndices[arrStartIndices.length-1]].arrLength;  
  
 if (Array.isArray(arrLength)) {  
 data[arrStartIndices[arrStartIndices.length-1]].arrLength.unshift(data.length - arrStartIndices[arrStartIndices.length-1]);  
 } else {  
 data[arrStartIndices[arrStartIndices.length-1]].arrLength = [data.length - arrStartIndices[arrStartIndices.length-1], arrLength];  
 }  
 } else {  
 data[arrStartIndices[arrStartIndices.length-1]].arrLength = data.length - arrStartIndices[arrStartIndices.length-1];  
 }  
  
 arrStartIndices.pop();  
 arrItemNumber.pop();  
 }  
 }  
 }  
  
 function substringReplacement(str, substr, strReplace) {  
 return str.replace(substr, strReplace);  
 }  
  
 function xmlToJson(data) {  
 var res = '';  
 var stripNS = xml2js.processors.stripPrefix;  
  
 xml2js.parseString(data, {tagNameProcessors: [stripNS], explicitArray: false}, function (err, result) {  
 if (result) {  
 res = result;  
 } else {  
 console.log(err);  
 res = err;  
 }  
 });  
  
 return res;  
 }  
  
 function sendReqToTxpg(url, data) {  
 try {  
 if (data) {  
 return axios.post(url, data);  
 } else {  
 return axios.get(url);  
 }  
 } catch (error) {  
 console.error(error);  
 }  
 }  
  
 function getWsdlName(url) {  
 var urlArr = url.split('/');  
 return urlArr[urlArr.length - 1].substring(0, 3);  
 }  
  
 function getNameSpaces(nameSpace) {  
 var nameSpaces = '';  
  
 nameSpace.forEach(function (item) {  
 nameSpaces += ' xmlns:'+item.name+'="'+item.url+'"';  
 });  
  
 return nameSpaces;  
 }  
  
 function getDataFromRequest(objFromBody, objFromMap, currentNameSpace, options) {  
 var data = [];  
 var nameSpace = [];  
 var numberOfElementsInArr = 0;  
  
 if (currentNameSpace) {  
 nameSpace = currentNameSpace;  
 }  
  
 var rootPath = addNameSpacesForPath(objFromMap.rootPath);  
 parseBodyObj(objFromBody, objFromMap);  
 return {data: data, nameSpace: nameSpace};  
  
 function parseBodyObj(objFromBody, objFromMap, itemNumber) {  
 for (var elemName in objFromBody) {  
 if (Array.isArray(objFromBody[elemName])) {  
 var list = '';  
 numberOfElementsInArr = 0;  
  
 objFromBody[elemName].forEach(function (obj, index) {  
 if (typeof obj === 'string') {  
 list += obj + ' ';  
 } else {  
 parseBodyObj(obj, objFromMap[elemName], index);  
 }  
 });  
  
 if (list) {  
 var pathObj = objFromMap[elemName];  
 var pathWithNameSpace = addNameSpacesForPath(pathObj);  
 var path = rootPath + '.' + pathWithNameSpace;  
 list = substringReplacement(list, ' ', '');  
  
 data.push({path: path.split('.'), val: list, item: itemNumber});  
 } else {  
 data[data.length - numberOfElementsInArr].arrLength = numberOfElementsInArr;  
 }  
 } else if (objFromMap[elemName].path) {  
 if (itemNumber === undefined) {  
 itemNumber = -1;  
 } else if (itemNumber !== -1) {  
 numberOfElementsInArr++;  
 }  
  
 var val = objFromBody[elemName];  
 var pathObj = objFromMap[elemName];  
 var pathWithNameSpace = addNameSpacesForPath(pathObj);  
  
 var path = rootPath + '.' + pathWithNameSpace;  
  
 if (typeof val === 'object') {  
 if (options) {  
 if (!options[elemName]) {  
 val = JSON.stringify(val);  
 }  
 } else {  
 val = JSON.stringify(val);  
 }  
 }  
  
 data.push({path: path.split('.'), val: val, item: itemNumber});  
 } else {  
 parseBodyObj(objFromBody[elemName], objFromMap[elemName], itemNumber);  
 }  
 }  
 }  
  
 function addNameSpacesForPath(pathObj) {  
 var pathArr = pathObj.path.split('.');  
 var pathWithNameSpace = '';  
  
 pathObj.url.forEach(function (url, index) {  
 var urlIs = false;  
  
 nameSpace.every(function (nameSpaceElem) {  
 if (nameSpaceElem.url === url) {  
 pathWithNameSpace += nameSpaceElem.name + ':' + pathArr[index] + '.';  
 urlIs = true;  
 return false;  
 } else {  
 return true;  
 }  
 });  
  
 if (!urlIs) {  
 var urlArr = url.split('/');  
 var name = urlArr[urlArr.length - 1].substring(0, 3);  
 var count = 1;  
  
 nameSpace.forEach(function (nameSpaceElem) {  
 if (nameSpaceElem.name === name) {  
 name = name.substring(0, 3) + count;  
 count++;  
 }  
 });  
  
 nameSpace.push({url: url, name: name});  
 pathWithNameSpace += name + ':' + pathArr[index] + '.';  
 }  
 });  
  
 if (pathArr[pathArr.length - 1].substring(0, 1) === '@') {  
 pathWithNameSpace += pathArr[pathArr.length - 1];  
 } else {  
 pathWithNameSpace = pathWithNameSpace.substring(0, pathWithNameSpace.length - 1);  
 }  
  
 return pathWithNameSpace;  
 }  
 }  
  
 function createObjFromDataExact(data, objType) {  
 var resObj = {};  
 var countForPath = -1;  
 var countForData = 0;  
 var countForItemNumber = -1;  
 var currentItemIndex = [];  
  
 data.forEach(function (dataItem, index) {  
 countForPath = -1;  
 countForData = index;  
 var currentObj = createObjFromPath(dataItem);  
 resObj = parseObj(currentObj, resObj, dataItem.item);  
 });  
  
 return resObj;  
  
 function parseObj(objFromCurrentObj, objFromRes, itemNumber) {  
 if (objFromRes === undefined) {  
 objFromRes = {};  
 }  
  
 for(var elemName in objFromCurrentObj) {  
 if (Array.isArray(objFromCurrentObj[elemName])) {  
 if (typeof objFromCurrentObj[elemName][0] === 'string') {  
 objFromRes[elemName] = objFromCurrentObj[elemName];  
 } else {  
 if (!objFromRes[elemName]) {  
 objFromRes[elemName] = [];  
 }  
  
 if (Array.isArray(itemNumber)) {  
 countForItemNumber++;  
  
 if (countForItemNumber === itemNumber.length) {  
 countForItemNumber = 0;  
 }  
  
 objFromRes[elemName][itemNumber[countForItemNumber]] = parseObj(objFromCurrentObj[elemName][0], objFromRes[elemName][itemNumber[countForItemNumber]], itemNumber);  
 } else {  
 objFromRes[elemName][itemNumber] = parseObj(objFromCurrentObj[elemName][0], objFromRes[elemName][itemNumber], itemNumber);  
 }  
 }  
 } else if (typeof objFromCurrentObj[elemName] === 'object') {  
 if (objFromRes[elemName]) {  
 objFromRes[elemName] = parseObj(objFromCurrentObj[elemName], objFromRes[elemName], itemNumber);  
 } else {  
 return Object.assign(objFromRes, objFromCurrentObj);  
 }  
 } else {  
 return Object.assign(objFromRes, objFromCurrentObj);  
 }  
 }  
  
 return objFromRes;  
 }  
  
 function createObjFromPath (dataItem) {  
 var obj = {}  
  
 if (countForPath < dataItem.path.length - 2) {  
 countForPath++;  
  
 if (dataItem.item !== -1) {  
 if (countForPath === 0 && dataItem.arrLength) {  
 if (Array.isArray(dataItem.arrLength)) {  
 var arrLength = dataItem.arrLength;  
  
 arrLength.reverse().forEach(function (arrLengthItem, index) {  
 if (index === 0) {  
 currentItemIndex.push(searchItemIndex(arrLengthItem));  
 } else {  
 currentItemIndex.push(searchItemIndex(arrLengthItem, currentItemIndex[currentItemIndex.length-1]-1));  
 }  
 });  
 } else {  
 if (Array.isArray(dataItem.item)) {  
 if (currentItemIndex.length === dataItem.item.length) {  
 currentItemIndex = [];  
 }  
 } else {  
 currentItemIndex = [];  
 }  
  
 currentItemIndex.push(searchItemIndex());  
 }  
 }  
  
 if (currentItemIndex.indexOf(countForPath) !== -1) {  
 obj[dataItem.path[countForPath]] = [];  
 obj[dataItem.path[countForPath]][0] = createObjFromPath(dataItem);  
 } else {  
 obj[dataItem.path[countForPath]] = createObjFromPath(dataItem);  
 }  
 } else {  
 obj[dataItem.path[countForPath]] = createObjFromPath(dataItem);  
 }  
 } else {  
 countForPath++;  
 obj[dataItem.path[countForPath]] = dataItem.val;  
 }  
  
 return obj;  
 }  
  
 function searchItemIndex(arrLength, itemIndex) {  
 var count = countForData;  
 var elem = data[count].path[itemIndex];  
 if (!itemIndex) {  
 itemIndex = data[count].path.length-1;  
 }  
 if (!arrLength) {  
 arrLength = data[countForData].arrLength;  
 }  
  
 while (count < countForData + arrLength) {  
 if (data[count].path.lastIndexOf(elem, itemIndex) !== -1) {  
 count++;  
 } else {  
 count = countForData;  
 itemIndex--;  
 elem = data[count].path[itemIndex];  
 }  
 }  
  
 if (objType !== 'response') {  
 itemIndex--;  
 }  
  
 return itemIndex;  
 }  
 }  
  
 async function jsonToXml(obj) {  
 return xmlBuilder.create(obj, {separateArrayItems: true, headless: true}).end();  
 }  
  
 function basicAuth(req, res, next) {  
 if (req.path === '/users/authenticate') {  
 return next();  
 }  
  
 if (!req.headers.authorization || req.headers.authorization.indexOf('Basic ') === -1) {  
 return res.status(401).json({errorCode: "InvalidLogin", errorDescription: "Login not provided"});  
 }  
  
 var base64Credentials = req.headers.authorization.split(' ')[1];  
 var credentials = Buffer.from(base64Credentials, 'base64').toString('ascii');  
 var arrCredentials = credentials.split(':');  
 var user = {name: arrCredentials[0], password: arrCredentials[1]};  
  
 if (!user) {  
 return res.status(401).json({errorCode: "InvalidLogin", errorDescription: "Login not provided"});  
 }  
  
 return user;  
 }  
  
 function getPwdToken(login, password, timeStamp) {  
 var pwdHash = sha256(login+'/'+password).toUpperCase();  
 return sha256(`timestamp=${timeStamp};pwd=${pwdHash}`).toUpperCase();  
 }  
  
 function getPwdHash (login, password) {  
 return sha256(login+'/'+password).toUpperCase();  
 }  
  
 function lowFirstSymb(str) {  
 if (!str) {  
 return str;  
 } else {  
 if (str[0] === '@') {  
 str = str.slice(1);  
 }  
 return str[0].toLowerCase() + str.slice(1);  
 }  
 }  
  
 function getCredentials(userAuth) {  
 var userNameArr = userAuth.name.split('/');  
 var timeStamp = Date.now();  
 var pwdToken = getPwdToken(userNameArr[1], userAuth.password, timeStamp);  
  
 return {  
 login: {  
 login: userNameArr[1],  
 ownerKind: userNameArr[0],  
 pwdToken: pwdToken,  
 pwdTimestamp: timeStamp,  
 },  
 };  
 }  
  
 return {  
 getResponse: getResponse,  
 basicAuth: basicAuth,  
 lowFirstSymb: lowFirstSymb,  
 getCredentials: getCredentials,  
 getPwdHash: getPwdHash,  
 createObjFromDataExact: createObjFromDataExact,  
 };  
  
})  
()

