* 1. Разработка приложения

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 03.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | |  |  |  | 3 Разработка программного обеспечения веб-приложения | Лит. | | | Лист | Листов |
| Пров. | |  |  |  |  | У |  | 1 | 17 |
| Консульт. | |  |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | |  |  |  |
| Утв. | |  |  |  |

* 1. Проектирование интеграционной шины

Главной особенностью данного проекта является разработка интеграционной шины, которая обеспечивает требуемое быстродействие передачи данных между системами. Для реализации шины необходимо решить ряд задача:

* обеспечить конфиденциальность данных интегрируемых систем;
* выбрать наиболее подходящие очереди сообщений;
* спроектировать *Mule*-приложения;

Для решения первой задачи я воспользовалсямодулем *Mule Secure Configuration*, который позволяет зашифровать свойства конфигурацию *Mule*-приложения и используется как еще один уровень безопасности *Mule*-приложений. Для работы с модулем *Mule Secure Configuration* требуется создать файл конфигурации в формате *YAML*, в котором определяются свойства приложения. Данные можно зашифровать по различным алгоритмам:

* Симметричный алгоритм блочного шифрования *AES*.
* Криптографический алгоритм, реализующий блочное симметричное шифрование с переменной длиной ключа Blowfish.
* Алгоритм для симметричного шифрования *DES*.
* Блочный шифр с длиной блока 64 бита и переменной длиной ключа RC2.

Ключ, по которому данные были зашифрованы, указывается при запуске *Mule*-приложения в конфигурации запуска, либо при развертывании *Mule*-приложения в *CloudHub*.

Выбор очереди *ActiveMQ* был обусловлен тем, что ее можно использовать для того, чтобы избавиться от операционных издержек и прочих связанных с ними затрат на работу с приложением для обмена сообщениями с открытым исходным кодом. *ActiveMQ* подходит для интеграции приложений при разработке новых облачных приложений, использующих микросервисы, которые оперируют сложными шаблонами обмена сообщениями и требуют низкой задержки, высокой доступности и надежности хранения сообщений. Поэтому я использовал *ActiveMQ*, как очередь сообщений между *Mule*-приложениями.

Я использовал *Amazon SQS* как очередь сообщений между *eCommerce*-платформой и *Mule*-приложениями, так как *Amazon* *SQS* – это компактный и полностью управляемые сервис очередей сообщений со встроенным масштабированием и простыми удобными *API*. *Amazon* *SQS* можно использовать для того, чтобы разъединять и масштабировать микросервисы, распределенные системы и бессерверные приложения. Также размер сообщений в *Aamazon* *SQS* ограничивается размером до 256 Кб, а передаваемые данные между *eCommerce*-платформой и *Mule*-приложениями являются *Id* сущностей с которыми произошло событие.

* 1. Разработка Mule-приложений

## Process Layer Mule-app Pub

Перед началом разработки была продумана файловая структура *Mule*-приложения, которая позволяла бы отделить основной код *Mule*-приложения от конфигурации всего *Mule*-приложения, обработчиков ошибок, интерфейса. Схема файловой структуры представлена на рисунке 3.1.

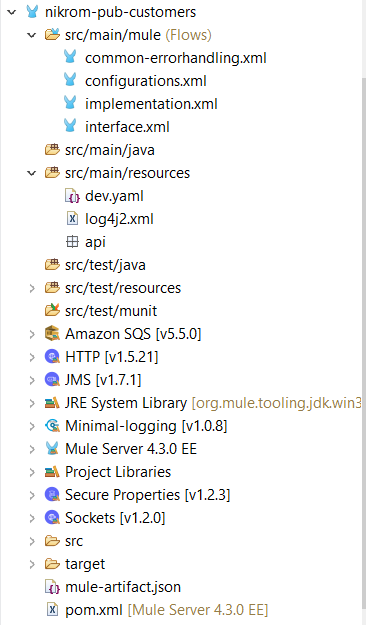


Рисунок 3.1 – Файловая структура *Mule*-приложения

Структура *Mule*-приложения содержит большое количество папок, каждая из которых имеет свое назначение. Можно выделить несколько файлов отличающихся от стандартной структуры таких *Mule*-приложений.

Файл *interface.xml* во вкладке *Message Flow* содержит поток *interfaceFlow*, который включает в себя блок *Receive Message*. Блок *Receive Message* осуществляет прослушивание очереди сообщений Amazon SQS, в итоге получает сообщения и регистрирует тело сообщения. Блок *Set Variable* устанавливает значения переменной из тела сообщения для дальнейшего ее использования в заголовке запроса. Блок *Info* является сокращенной версией обычного *Logger* и используется для более компактного и удобного для просмотра отображения логов *Mule*-приложения. Блок *Flow-Reference* используется для перенаправления потока сообщений на поток находящийся в файле implementation.xml, для продолжения работы *Mule*-приложения. Содержание вкладки *Message Flow* файла *interface* представлено на рисунке 3.2.

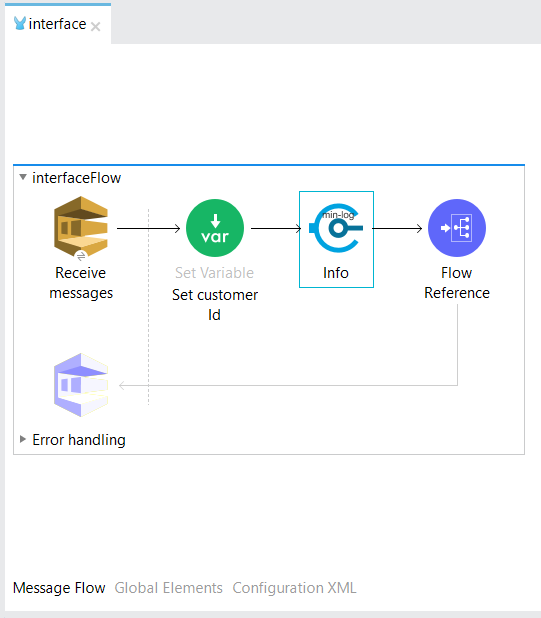


Рисунок 3.2 – Содержание *Message Flow* файла *Interface.xml*

Файл implementation.xml выполняет основной код Mule-приложения и содержит в себе код, который выполняет запрос в *Process Layer Mule-app M2* и отправляет полученные данные в очередь сообщений *ActiveMQ*. Поток *MessageLogger-SubFlow* содержит в себе блок *Info*, у которого в свойства указана переменная *logmessage*, она на различных этапах выполнения *Mule*-приложения содержит в себе соответствующие данные. Поток *call-m2-get-and-publisher-flow* в себе блоки *Set Variable* и MessageLogger-SubFlow, которых служат для установки значения переменной и перенаправления потока на поток для отображения логов. Блок *Request* используется для отправления *Get* запроса в *Process Layer Mule-app M2* и содержит в своих свойствах: *Method*, *Path*, *Host* и *Uri* *Parameter*, который содержит в себе переменную *Id*. Блок *Logger* в данном файле используется для более подробного описание результата выполнения запроса. Блок *Publish* используется для публикации данных в очередь *ActiveMQ* и содержит в своих свойствах: название очереди, *brokerUrl*, имя и пароль пользователя. На рисунке 3.3 представлено содержание *Message Flow* файла implementation.xml.

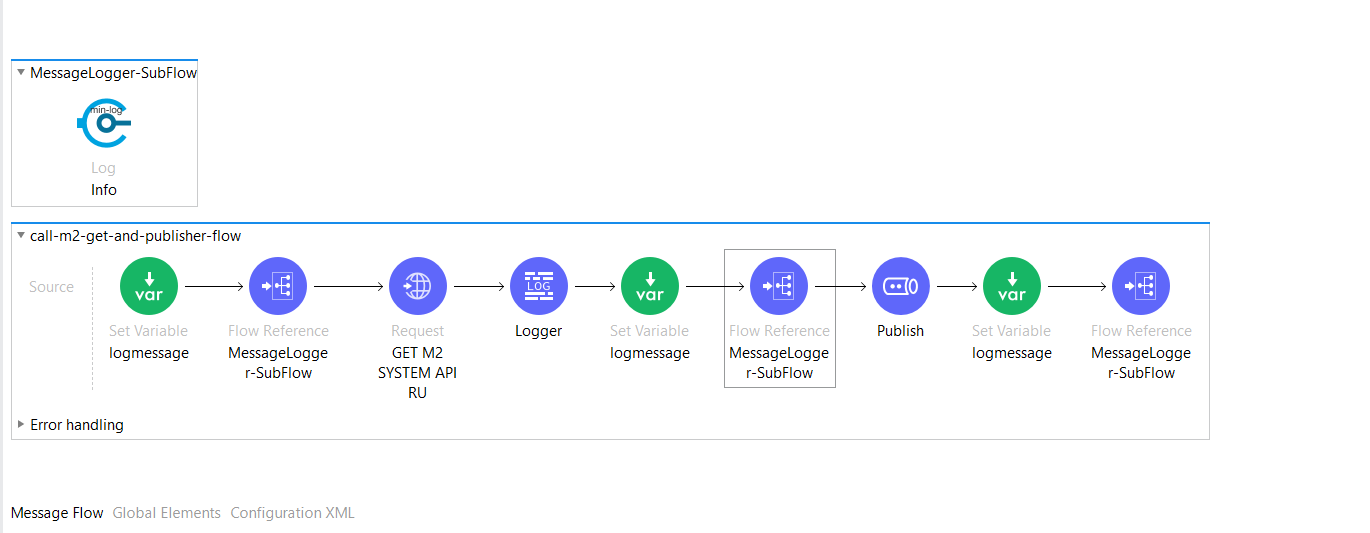


Рисунок 3.3 – Содержание *Message Flow* файла *implementation.xml*

Файл *configuration.xml* во вкладке Global Elements содержит в себе конфигурационные элементы для блоков, которые используются во всех файлах *Mule*-приложения. Элемент *Secure Properties Config* используется для шифрования свойств для всех конфигурационных элементов. Элемент *JMS* *Config* содержит в себе данные для доступа к очереди *ActiveMQ*. Элемент *HTTP Request configuration* содержит в себе свойства для отправки запроса в *Process Layer Mule-app M2*. Элемент *Amazon SQS Configuration* содержит в себе свой для доступа в очередь *Amazon SQS*. На рисунке 3.4 представлено содержание вкладки Global Elements.

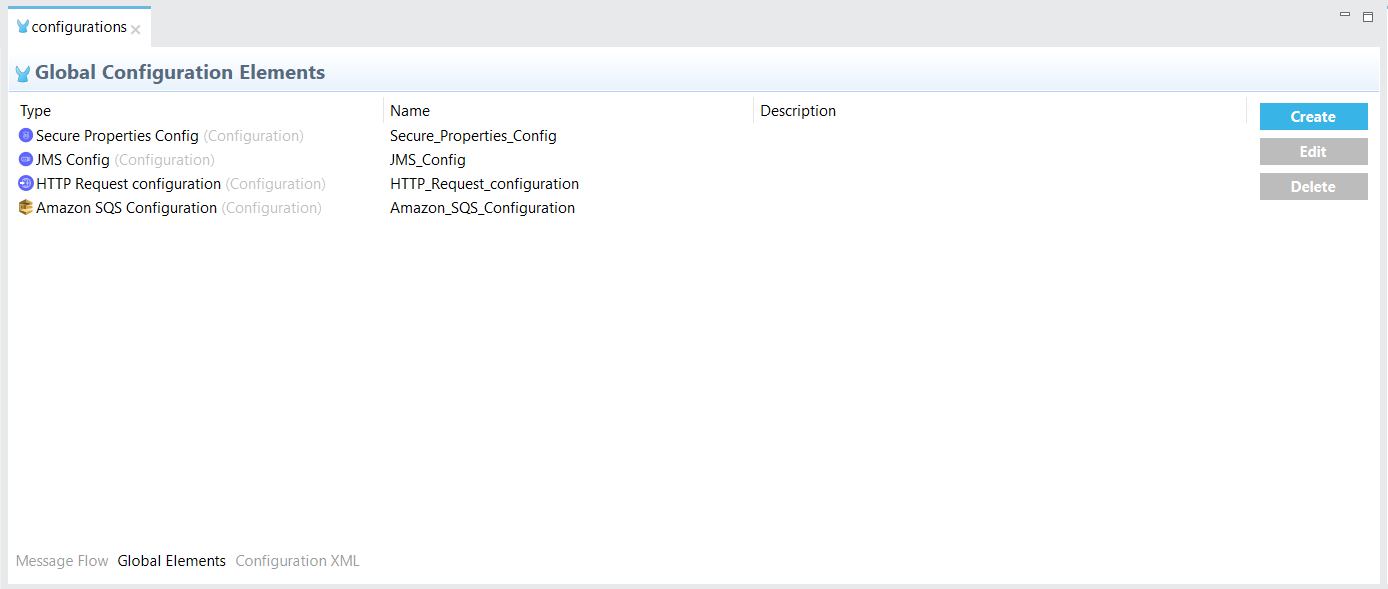


Рисунок 3.4 – Содержание вкладки Global Elements файла *configuration.xml*

Файл *common-errorhandling.xml* во вкладке Message Flow содержит поток *globalError\_Handler*, который используется для обработки ошибок, полученных при отправке запроса в *Mule*-приложение *Process Layer Mule-app M2* с *Uri Parameter*, оно отправляет запрос на получение данных в *eCommerce*-платформу. Поток *global-prepare-error-response-sub-flow* используется для публикации ошибок в очередь ошибок *ActiveMQ*. Содержание вкладки Message Flow файла *common-errorhandling.xml* представлено на рисунке 3.5.

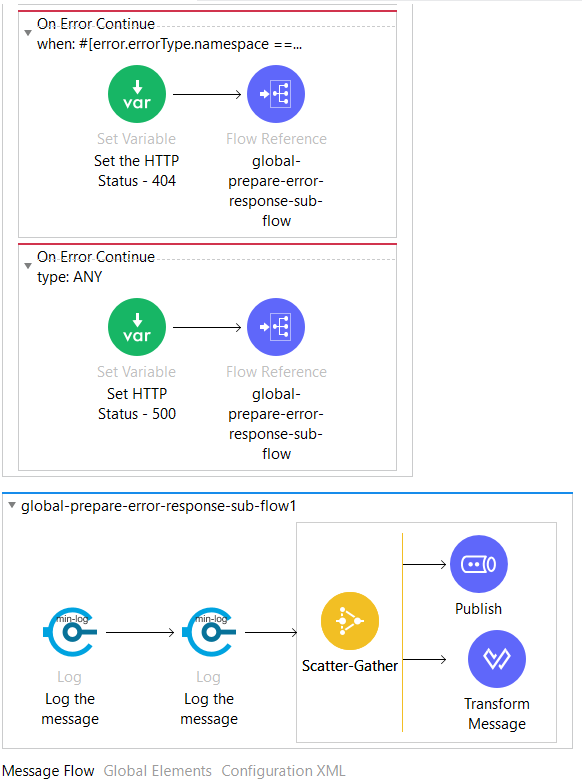


Рисунок 3.5 – Содержание вкладки Message Flow файла *common-errorhandling.xml*

Файл *dev.yaml*  содержит в себе свойства, которые используются в элементах конфигурации. На рисунке 3.6 представлено содержание файла *dev.yaml*.



Рисунок 3.6 – Содержание файла *dev.yaml*

Некоторые данные в файле *dev.yaml* зашифрованы по алгоритму *Blowfish*, чтобы *Mule*-приложение смогло расшифровать эти данные в конфигурации запуска приложения указывается ключ шифрования, а в модуле *Secure Properties* указывается алгоритм шифрования.

## System Layer Mule-app M2

Структура *Mule*-приложения похожа *System Layer Mule-app Pub* за исключением отсутствия файла *common-errorhandling.xml*, который в данном *Mule*-приложением не требуется, так как ошибки будут обрабатываться в *Mule*-приложении *System Layer Mule-app Pub*. Также структура *Mule*-приложения отличается тем что в данном *Mule*-приложении присутствует файл, например, api\_customers.raml, в котором описывается используемое *API eCommerce-*платформы. Структура *Mule*-приложения представлена на рисунке 3.7.

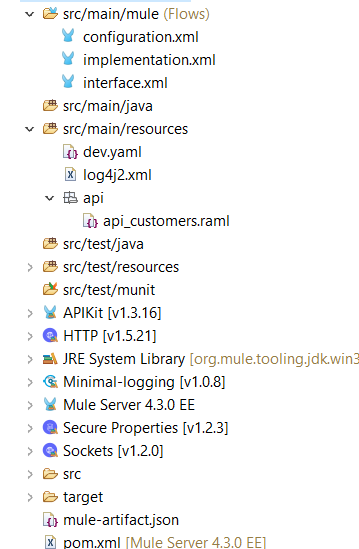


Рисунок 3.7 – Структура *Mule*-приложения

В данном *Mule*-приложении файл *interface.xml* содержит поток *interfaceFlow*. Блок Listener осуществляет возможность обращения к *Mule*-приложению, свойства блока включает в себя *path*, по которому другие приложения могут обращаться к *System Layer Mule-app M2*. Содержание *Message Flow* файла *interface.xml* представлено на рисунке 3.8.

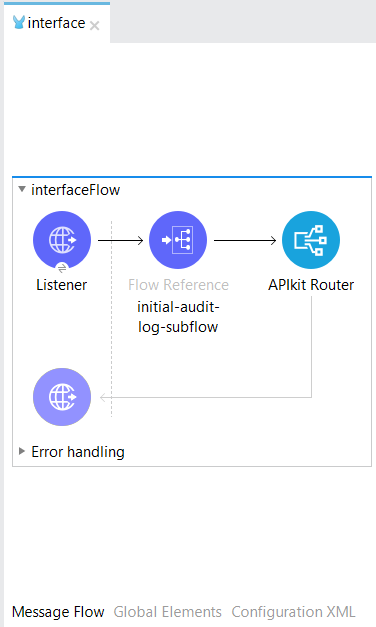


Рисунок 3.8 – Содержание *Message Flow* файла *interface.xml*

В файле *implementation.xml* поток *get:\customers\(Id)-implementation* осуществляет преобразование *path*, для обращения к *eCommerce-*платформе и получению требуемых данных по *Id*. Блок *Transform Message* служит для преобразования данных, в данном *Mule-*приложение, он соединяет *path* c *Id*, которое может быть получено при запросе в это *Mule*-приложение с *Uri Parameter*. Блок *Request* выполняет запрос в *eCommerce*-платформу, в свойствах этого блока указана переменная *Id*, с помощью которого можно получить требуемые данные. На рисунке 3.9 представлено содержание *Message Flow* файла *implementation.xml*.

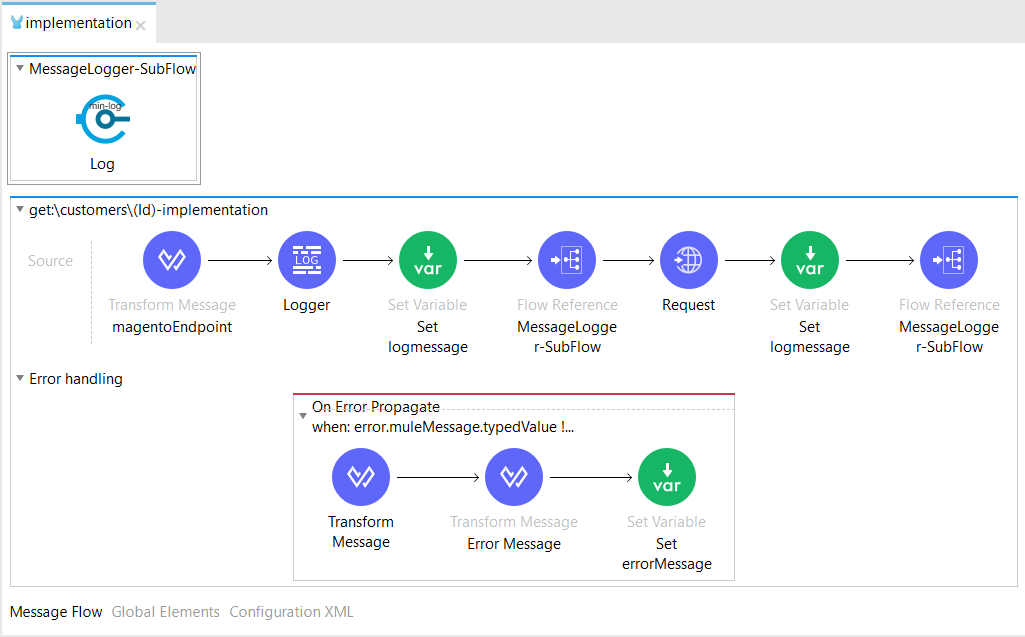


Рисунок 3.9 – Содержание вкладки *Message Flow* файла *implementation.xml*

Файл *configuration.xml* содержит блоки *Minimal-logging*, которые используются для установки дополнительных свойств, отображаемых в логах. Также в этом файле определены глобальные элементы конфигурации *System Layer Mule-app M2 Mule*-приложения. Содержание вкладки *Message Flow* файла *configuration.xml* представлено на рисунке 3.10.

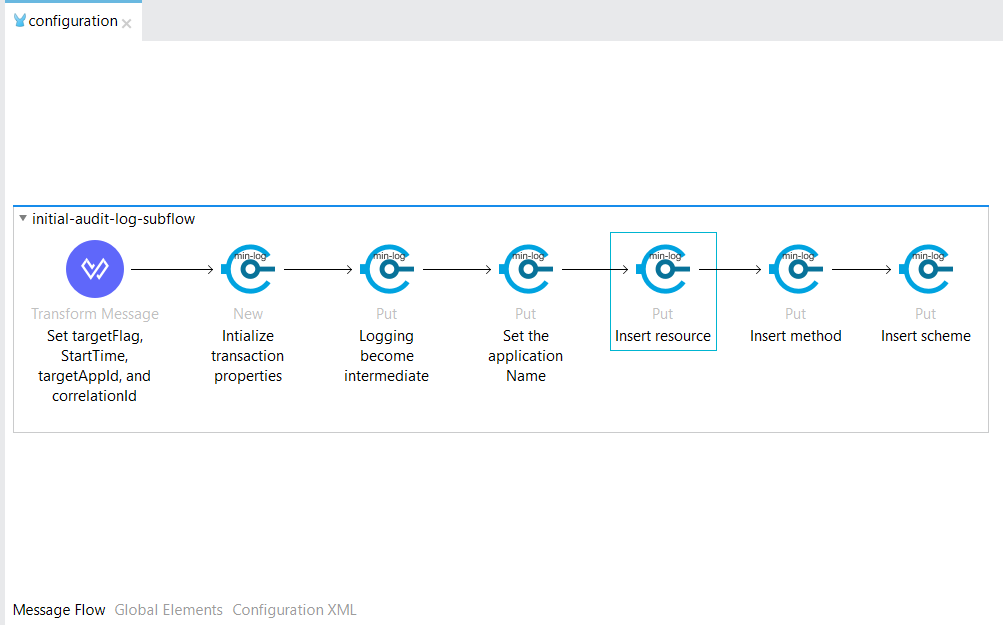


Рисунок 3.10 – Содержание вкладки *Message Flow* файла *configuration.xml*

## Process Layer Mule-app Queue

Структура файлов данного *Mule*-приложения аналогичная *Mule-*приложению *System Layer Mule-app M2*, за исключением отсутствия файла *api.raml*. Данное *Mule-*приложение используется для получения данных из очереди *ActiveMQ*, к нему обращается *Mule-*приложению *System Layer Mule-app CRM*, которое служит конечной точкой.

Файл *implementation.xml* содержит поток *nikrom-q-Flow*, в котором блок *Consume* извлекает данные из очереди *ActiveMQ*. Блок *Transform Message* в данном *Mule-*приложении преобразует полученные данные, путем избавления от пустых массивов. Содержание *Message Flow* файла *implementation.xml* представлено на рисунке 3.11.

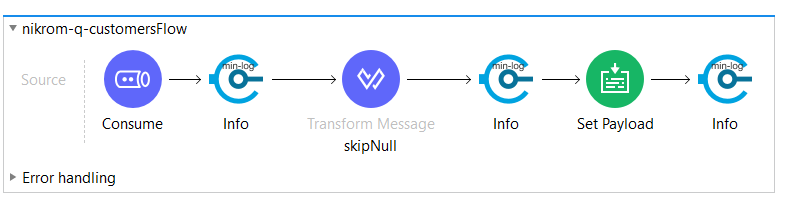


Рисунок 3.11 – Содержание *Message Flow* файла *implementation.xml*

В данном *Mule*-приложении файл *interface.xml* содержит поток *interfaceFlow*. Блок Listener осуществляет возможность обращения к *Mule*-приложению, свойства блока включает в себя *path*, по которому другие приложения могут обращаться к *Process Layer Mule-app Queue*. На рисунке 3.12 представлено содержание *Message Flow* файла *interface.xml*.

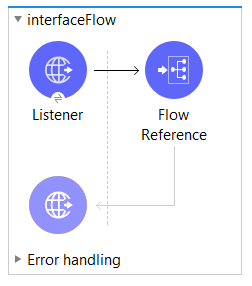


Рисунок 3.12 – Содержание *Message Flow* файла *interface.xml*

## System Layer Mule-app CRM

Структура файлов данного *Mule-*приложения аналогична *Process Layer Mule-app Queue*. *Mule*-приложение *System Layer Mule-app CRM* используется в качестве конечных точек, которые используются пакетом программного обеспечения для управления взаимоотношениями с клиентами *Microsoft Dynamics CRM*.

Файл *interface.xml* содержит поток *interface*, блок *Listener* осуществляет возможность обращение *Microsoft Dynamics CRM* к данному *Mule-*приложению и перенаправляет поток к потоку в файле *implementation.xml*. Содержание *Message Flow* файла *interface.xml* представлено на рисунке 3.13.

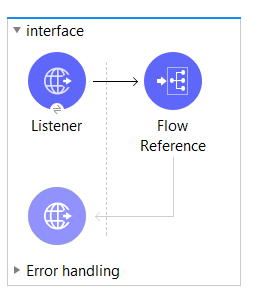


Рисунок 3.13 – Содержание *Message Flow* файла *implementation.xml*

Файл *implementation.xml* содержит поток *nikrom-crm-Flow*, который обращается в *Mule-*приложение *Process Layer Mule-app Queue* на получение данных из очереди. Блок *Request* осуществляет запрос по *path* и *host*, опубликованного в *CloudHub Mule-*приложения *Process Layer Mule-app Queue*. Блок *Logger* используется для отображения логов выполнения блока *Request*. Содержание *Message Flow* файла *implementation.xml* представлено на рисунке 3.14.

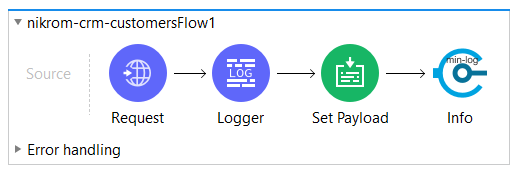


Рисунок 3.14 – Содержание *Message Flow* файла *implementation.xml*

## Amazon SQS

В сервисе очередей сообщений *Amazon SQS* были созданы три очереди: *customers, orders, products*. Данные очереди используются для того, чтобы хранить *Id*. Когда на eCommerce-платформе происходят различные события, например, создание продукта, *Id* этого продукта попадает в очередь. Далее данные из очереди извлекает *Mule-*приложение. Очереди *Amazon SQS* используемые в интеграционной шине представлены на рисунке 3.15.

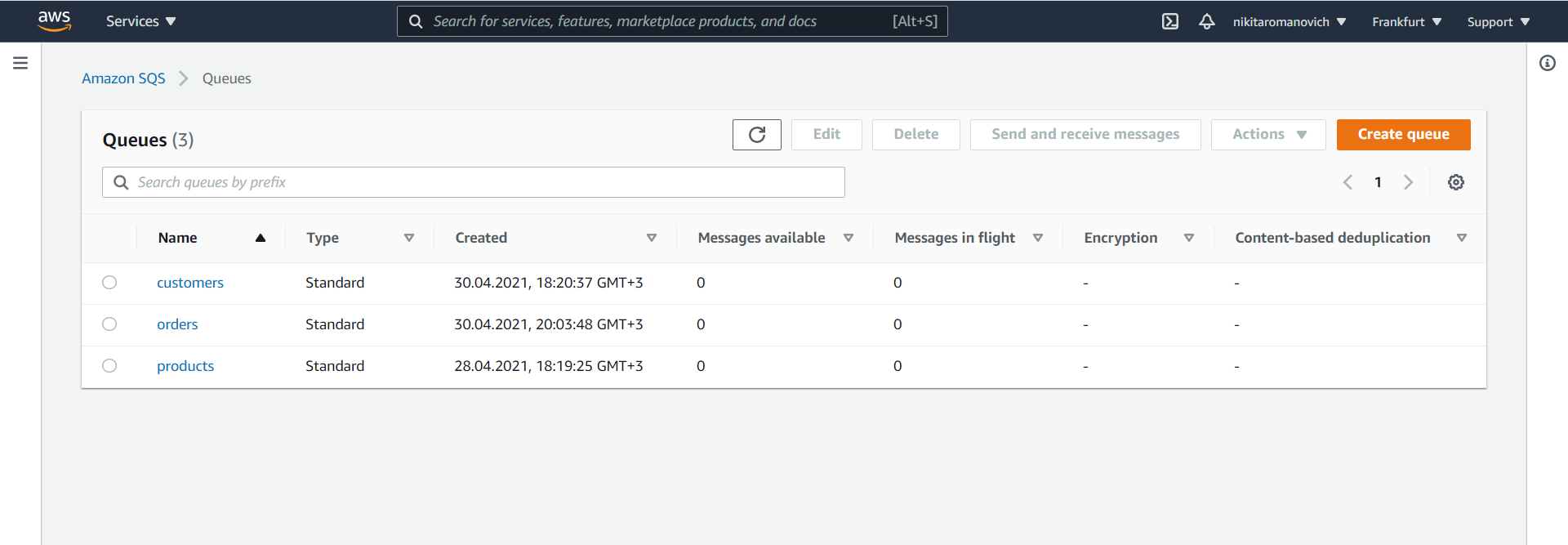


Рисунок 3.15 – Очереди *Amazon SQS* используемые в интеграционной шине

## ActiveMQ

На сервере обмена сообщений были созданы очереди три очереди для хранения данных: *customer, orders, products*. Также были созданы три очереди для хранения ошибок полученные при запросе на получение данных из *eCommerce*-платформы: *customers\_dlq, orders\_dlq, products\_dlq*. На рисунке 3.16 представлены очереди *ActiveMQ* используемые в интеграционной шине.

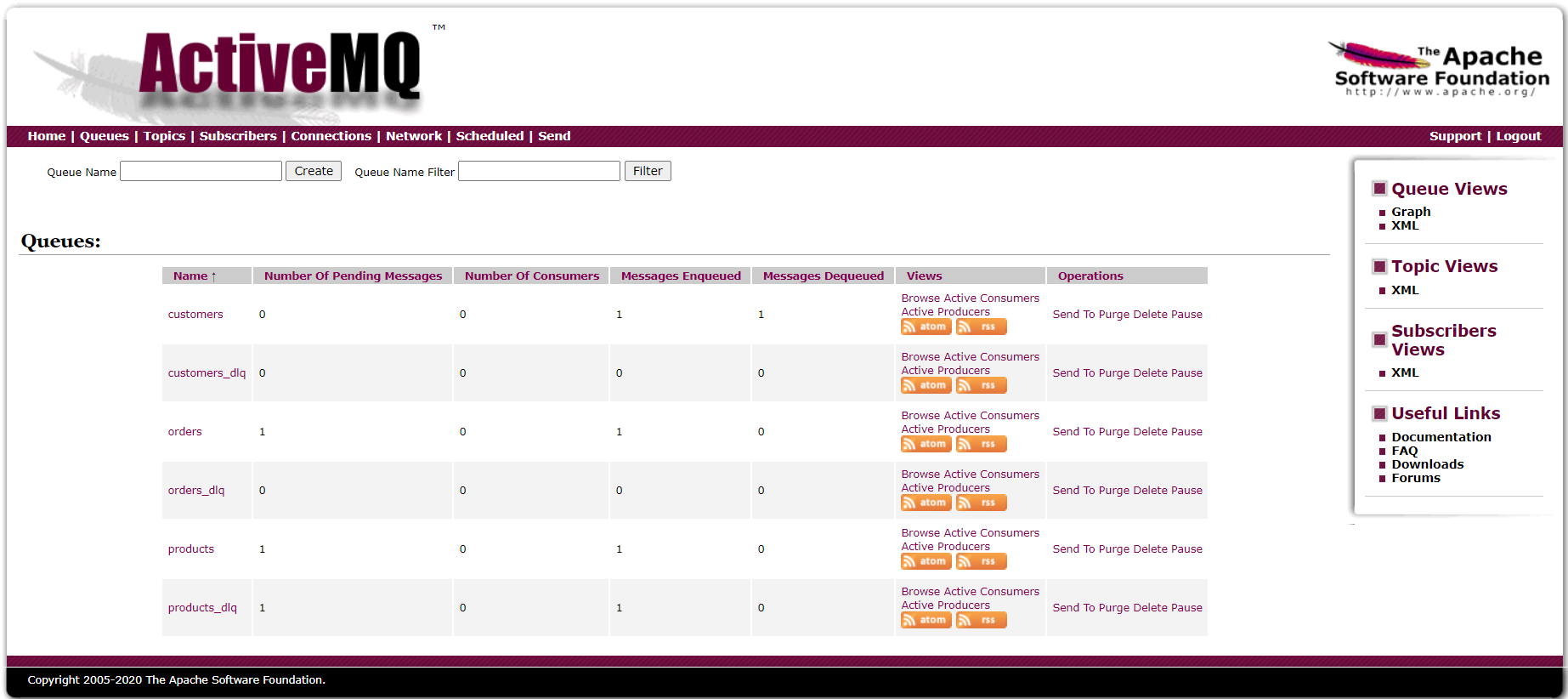


Рисунок 3.16 – Очереди *Active MQ* используемые в интеграционной шине

* 1. Вывод по разделу

В результате выполнения дипломного проекта было разработано приложение «Интеграционная шина между eCommerce-платформой и Microsoft Dynamics СRM». Все поставленные задачи перед проектом были выполнены. Основные сложности связанные с обеспечением конфиденциальности данных интегрируемых систем решились стандартными способами с минимальной потерей производительности приложения. Кроме этого были выбраны сервисы очередей сообщений и созданы в них очереди, которые обеспечивают требуемый функционал для реализации дипломного проекта.

Полученное приложение не требует специфического программного обеспечения, для его запуска необходимо наличие установленного интернет-браузера на компьютере пользователя, доступ в сеть *Internet* любым доступным и удобным для пользователя способом, а также доступ к имеющимся данным. В интеграционной шине использованы устойчивые формы записи кода (использование обработчиков ошибок) благодаря чему повышается отказоустойчивость приложения.

Приложение справляется со всеми поставленными задачами и целями, а именно:

* разработаны *Mule*-приложения;
* созданы очереди *Amazon SQS*;
* созданы очереди *ActiveMQ*;
* реализован модуль обеспечения конфиденциальности;
* разработан поток для отслеживания очереди *Amazon SQS*;
* реализовано преобразование данных;
* разработан поток для добавления данных в очереди *ActiveMQ*;
* реализован поток для извлечения данных из очереди *ActiveMQ*;
* разработан поток для передачи данных между *Mule­­*-приложениями;
* разработан поток для обработки ошибок;

Спроектированная архитектура проекта, позволит добавлять и усовершенствовать функциональные возможности и поддерживать работу системы в будущем.