2 Проектирование приложения

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *ДП 02.00.ПЗ* | | | | | |
|  |  |  |  |  |
|  |  | Ф.И.О | Подпись | Дата |
| Разраб. | |  |  |  | 2 Проектирование веб-приложения | Лит. | | | Лист | Листов |
| Пров. | |  |  |  |  | У |  | 1 | 9 |
| Консульт. | |  |  |  | БГТУ 74218026, 2020 | | | | |
| Н. контр. | |  |  |  |
| Утв. | |  |  |  |

* 1. **Обзор средств** разработки

## Платформа Mule ESB

Для реализации данного приложения была выбрана платформа *Mule ESB*.

Платформа *Mule ESB* представляет технологию от компании *Mulesoft,* предназначенную для объединения различных информационных систем на основе принципов обмена сообщениями, сопоставления данных, управления сообщениями, надежности, защиты и масштабирования между узлами.

Платформа ориентирована на языке программирования *Java*, но может быть брокером для других платформ, таких как *.NET* с помощью веб-служб или сокетов.

Архитектура представляет собой масштабируемый, распределённый объект-брокер, который может легко управлять взаимодействиями между приложениями различных производителей, включая облачные и с использованием почти всех современных протоколов.

Многие из конкурентных реализаций *ESB* предоставляют ограниченную функциональность или строятся поверх существующего сервера приложений или сервера сообщений, привязывая пользователя к конкретному поставщику. *Mule* независима от поставщика.

*Mule ESB* в качестве элементов обработки сообщений может использовать вставки кода на популярных языках программирования. Исходные коды *Mule ESB* написаны на Java и соответственно поддерживается взаимодействие с данным стеком технологий.

*Mule ESB* построена на принципе обмена сообщениями между коннекторами. Путём трансформирования и маршрутизации сообщений можно создать необходимый интеграционный процесс.

## Среда разработки

В качестве среды разработки было решено использовать *Anypoint Studio*, т.к. она является основным инструментом по разработке *Mule*-приложений, построена на основе *Eclipse* со своим набором для визуального моделирования и создания трансформаций, каталогом готовых компонент и доступом к *Anypoint Exchange*. Основой приложения является *Flow* – это описание того, как события, которые поступают в приложение, им обрабатываются. Для этого строится цепочка из обработчиков, каждый из которых отвечает за свою функцию: трансформацию, сериализацию, логирование, доступ к базе данных. Палитра содержит набор готовых обработчиков, которые можно встроить в поток и настроить соответственно задаче.

Очень частой задачей бывает трансформация данных. Одна система может передать в виде запроса массив в формате *XML*, в то время как другая ожидает *JSON* совсем в другой структуре. Для облегчения процесса написания этой трансформации в *Anypoint Studio* есть визуальные средства по разработке.

Для тестирования приложений имеется *MUnit* фреймворк, в котором теми же инструментами, что и при разработке, можно писать тесты и получать результаты по покрытию потоков тестами. А поддержка *Maven* дает возможность построить *CI* или *CD*-процессы как для обычных *Java*-приложений.

## **Обоснование** выбора языка программирования

*Java* – объектно-ориентированный язык программирования, разрабатываемый компанией *Sun Microsystems* с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался *Oak* и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения

Отличительной особенностью *Java* в сравнении с другими языками программирования общего назначения является обеспечение высокой продуктивности программирования, нежели производительность работы приложения или эффективность использования им памяти.

Язык *Java* поставляется с достаточно объемной библиотекой классов. Так же, как и библиотеки классов систем разработки приложений на языке *C++*, библиотеки классов Java значительно упрощают разработку приложений, представляя в распоряжение программиста мощные средства решения распространенных задач.

Три ключевых элемента объединились в технологии языка Java:

* Предоставляет для широкого использования свои.
* Высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки.
* Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов.

## **Платформа CloudHub**

*CloudHub* - это платформа интеграции как услуга *iPaaS*, где вы можете развертывать сложные приложения кросс-облачной интеграции в облаке, создавать новые *API*-интерфейсы поверх существующих источников данных, интегрировать локальные приложения с облачными службами и многое другое.

*CloudHub* представляется как эластичное облако, то есть оно масштабируется по запросу. Можно начать с малого и увеличивать масштаб по мере роста потребностей, не меняя приложения и не испытывая простоев. *CloudHub* предоставляет масштабируемую архитектуру, на которой можно создавать приложения интеграции, публиковать *REST* *API* или веб-сервисы и многое другое.

Используя *Anypoint Studio*, можно создавать приложения интеграции и развертывать их в *CloudHub* всего за несколько кликов. Затем можно получить к ним доступ, как к любому другому приложению, развернутому через платформу, войдя в *Anypoint Platform*.

*CloudHub* разработан, чтобы предоставить предприятиям мультиарендную, безопасную, эластичную и доступную интеграционную платформу.

Управление *CloudHub* осуществляется с помощью консоли *Anypoint Runtime Manager* в *Anypoint Platform*. Можно развернуть приложения напрямую из *Anypoint Studio*, через *CloudHub REST API* или через интерфейс командной строки *Anypoint Platform*.

Архитектура *CloudHub* включает в себя два основных компонента - сервисы платформы Anypoint и рабочее облако. Эти два компонента и консоль Runtime Manager, через которую вы получаете к ним доступ, работают вместе для запуска ваших приложений интеграции.

## Amazon SQS

*Amazon Simple Queue Service* – это полностью управляемый сервис очередей сообщений, с помощью которого можно изолировать и масштабировать микросервисы, распределенные системы и бессерверные приложения. *SQS* позволяет забыть о сложностях и лишних расходах, связанных с управлением промежуточным программным обеспечением, и дает разработчикам возможность сосредоточиться на важной работе. С помощью *SQS* можно отправлять, хранить и получать сообщения компонентов программного обеспечения в любом масштабе без потери сообщений, и необходимости обеспечивать доступность других сервисов.

*SQS* предлагает два типа очередей сообщений. Стандартные очереди обеспечивают максимальную пропускную способность, оптимальное упорядочение и доставку сообщений по принципу «хотя бы один раз». Очереди *FIFO SQS* с ограниченной пропускной способностью гарантируют, что сообщения будут обрабатываться строго однократно и исключительно в порядке отправления.

Amazon *SQS* можно использовать для обмена конфиденциальными данными между приложениями с применением шифрования на стороне сервера в целях шифрования текста каждого сообщения. Благодаря интеграции *Amazon SQS SSE* с *AWS Key Management Service* можно централизованно управлять ключами, защищающими сообщения *SQS*.

## Amazon MQ

*Amazon MQ* – управляемый сервис брокера сообщений для *Apache ActiveMQ* и *RabbitMQ*. С его помощью можно без труда настраивать на *AWS* брокеры сообщений и работать с ними. *Amazon MQ* позволяет уменьшить операционные обязанности благодаря эффективному управлению подготовкой, настройкой и обслуживанием брокера сообщений. Поскольку *Amazon MQ* подключается к приложениям с помощью API и протоколов, созданных в соответствии с отраслевыми стандартами, можно легко провести миграцию на AWS без необходимости переписывать код.

В *Amazon MQ* реализованы соответствующие отраслевым стандартам *API*-интерфейсы и протоколы для обмена сообщениями, включая *JMS*, *NMS*, *AMQP 1.0*, *STOMP*, *MQTT* и *WebSocket*. Благодаря этому перейти на *Amazon* *MQ* с любого другого брокера сообщений, использующего перечисленные стандарты, очень просто: потребуется только обновить конечные точки приложений для подключения к *Amazon* *MQ*.

*Amazon* *MQ* управляет администрированием и обслуживанием брокеров сообщений и автоматически выделяет инфраструктуру для высокой доступности. Нет необходимости в предоставлении оборудования или в установке и обслуживании программного обеспечения. *Amazon* *MQ* автоматически управляет такими задачами, как обновление программного обеспечения, обновление системы безопасности и обнаружение аварий и восстановление после них.

## Apache ActiveMQ

*Apache ActiveMQ* – мощный сервер обмена сообщениями и шаблонов интеграции с открытым исходным кодом.

*Apache ActiveMQ* обеспечивает быстродействие работы обмена сообщениями, поддерживает множество межъязыковых клиентов и протоколов, поставляется с простыми в использовании шаблонами интеграции предприятия и множеством дополнительных функций, при этом полностью поддерживает *JMS 1.1* и *J2EE* *1.4.*

Особенности *Apache ActiveMQ*:

* Разработан для высокопроизводительной кластеризации, взаимодействия клиент-сервер и однорангового взаимодействия.
* *REST API* для обеспечения независимого от технологий и языкового веб-*API* для обмена сообщениями.
* *Ajax* для поддержки веб-потоковой передачи для веб-браузеров с использованием чистого *DHTML*, что позволяет веб-браузерам быть частью структуры обмена сообщениями.
* Поддержка *CXF* и *Axis*, так что *ActiveMQ* может быть легко помещен в любой из этих стеков веб-сервисов для обеспечения надежного обмена сообщениями.
* Может использоваться как поставщик *JMS* в памяти, подходит для модульного тестирования *JMS*.
* Поддерживает очень быстрое сохранение с использованием *JDBC* вместе с высокопроизводительным журналом.
  1. Архитектура программного продукта

Графическое представление архитектуры проектируемого приложения изображено на рисунке 2.1.

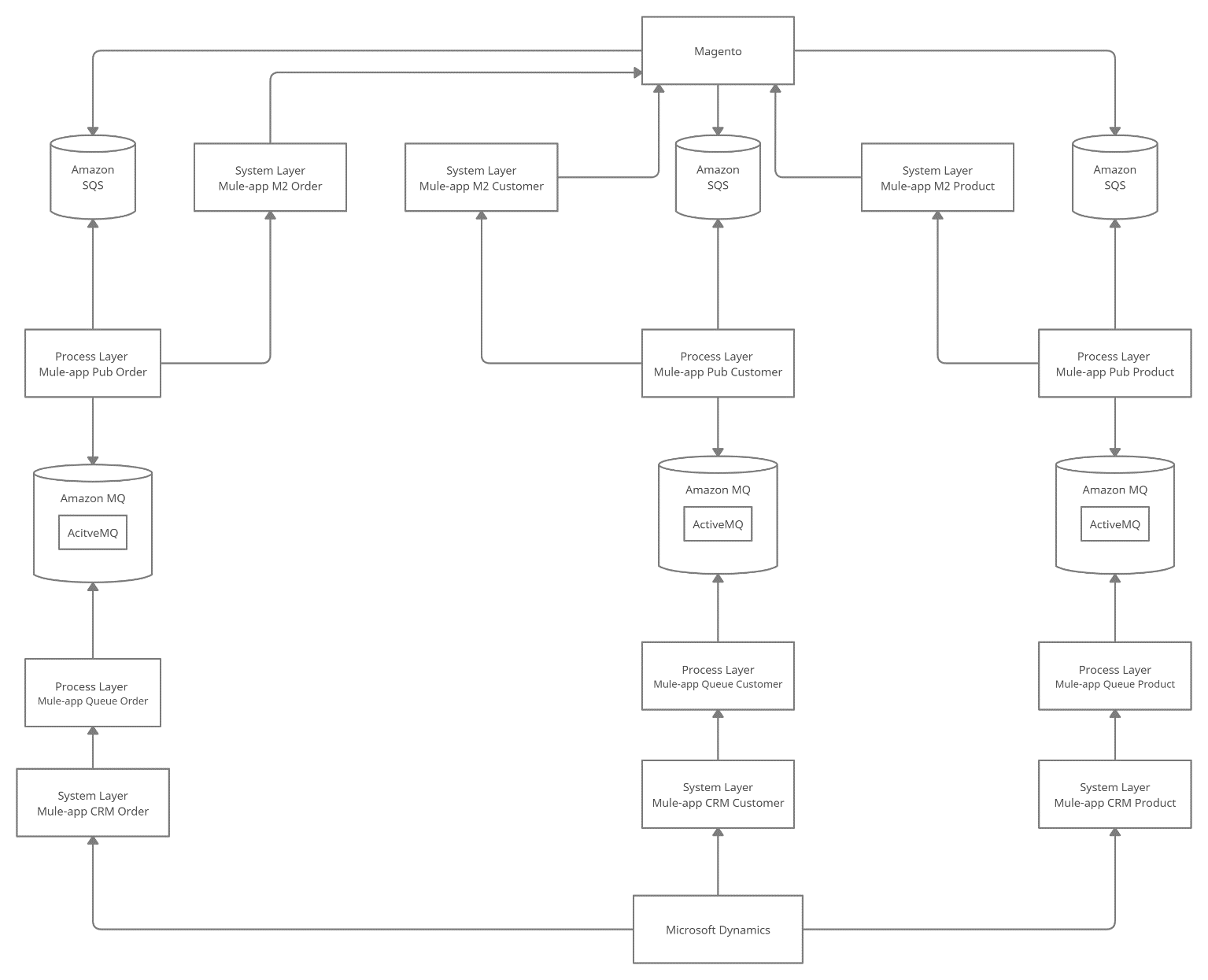


Рисунок 2.1 – Архитектура приложения

Интеграционная шина состоит из двенадцати *Mule*-приложений, трёх очередей *Amazon SQS* и трёх очередей *ActiveMQ*. Очереди *Amazon SQS* получают сообщения, которые содержат *Id product* или *Id customer*, или *Id order* в зависимости от выполняемого действия на *e-Commerce* платформе. Mule-прложения *Process Layer Pub* проcлушивают очереди *Amazon SQS*, когда в очереди приходят сообщения, *Mule*-приложения получают *Id* и выполняют запрос с *Uri* параметром в другие *Mule*-приложения *System Layer M2*, а они выполняют запрос в *e-Commerce* платформу на получение данных по *Id*. После выполнения запроса данные публикуются в очередь *Active MQ*.

*Mule*-приложения *System Layer CRM* представляют собой конечные точки. Когда *Microsoft Dynamics CRM* обращается к конечным точкам для получения данных, *Mule*-приложени *System Layer CRM* выполняет запрос в *Mule*-приложение *System Layer Queue*, которые обращаются в очередь *AcriveMQ* для получения данных.

* 1. **Выводы** по разделу

В результате обзора технических средств и анализа поставленной задачи для реализации дипломного проекта была выбрана интеграционная платформа *Mule ESB*, так как данная платформа предоставляет весь нужный функциональный набор для создания интеграционной шины. Исходя из выбранной платформы, языком, для написания интеграционной шины был выбран *Java*, так как данный язык программирования позволяет создавать различные безопасные и надежные приложения, работающие на платформе *Mule ESB*. В качестве очередей сообщений были выбраны очереди Amazon SQS и AciveMQ, которые подходят для различных задач и обеспечивают требуемое быстродействие.