**ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ**

**ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

**Факультет компьютерных наук**

**Кафедра «Информационные системы»**

*<Система интеграции Web-сайтов рекрутмента>*

071900 (230201) Информационные системы и технологии

*<Информационные системы и телекоммуникации>*

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_Тюкачев Н.А. к.ф.-м.н., доцент \_\_.\_\_.2013

Студент \_\_\_\_\_\_\_\_\_\_Петрушин И.А. \_\_.\_\_.2013

Руководитель \_\_\_\_\_\_\_\_\_\_Сапегин А.А. к.т.н., доцент \_\_.\_\_.2013

Воронеж2012

# ОГЛАВЛЕНИЕ

Оглавление

[ОГЛАВЛЕНИЕ 2](#_Toc353550545)

[ВВЕДЕНИЕ 4](#_Toc353550546)

[1. ПОСТАНОВКА ЗАДАЧИ 6](#_Toc353550547)

[2. АНАЛИЗ ЗАДАЧИ 7](#_Toc353550548)

[2.1 Анализ предметной области 7](#_Toc353550549)

[2.2 ESB как подход к интеграции на уровне данных 8](#_Toc353550550)

[2.3 Очереди Сообщений 9](#_Toc353550551)

[2.4 Основные термины 10](#_Toc353550552)

[Таблица 2.2.1 Основные термины 11](#_Toc353550553)

[2.5 Анализ требований 11](#_Toc353550554)

[Рисунок 2.3.1 Диаграмма Компонентов 11](#_Toc353550555)

[2.5.1 Компонент “ WEB Client ” 12](#_Toc353550556)

[2.5.1.1 Диаграммы вариантов использования 12](#_Toc353550557)

[Рисунок 2.3.1.1.1 Диаграмма вариантов использования 13](#_Toc353550558)

[2.5.1.2 Требования к созданию профиля резюме 14](#_Toc353550559)

[2.5.2 Компонент “ Web Service Business-Logic ” 15](#_Toc353550560)

[2.5.2.1 Диаграмма вариантов использования 15](#_Toc353550561)

[2.5.3 ESB Integrator 17](#_Toc353550562)

[2.5.3.1 Архитектура и общие термины JMS 18](#_Toc353550563)

[2.5.3.2 Модель взаимодействия точка-точка 19](#_Toc353550564)

[2.5.3.3 Модель взаимодействия издание-подписка 20](#_Toc353550565)

[2.5.3.4 Требования к JMS провайдеру 21](#_Toc353550566)

[3. АППАРАТНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА 23](#_Toc353550567)

[Таблица 3.1 Средства реализации 23](#_Toc353550568)

[3.1 Требования к составу и параметрам технических средств 23](#_Toc353550569)

[Таблица 3.1.1 Минимальные системные требования 24](#_Toc353550570)

[3.2 Требования к программным средствам, используемым системой 24](#_Toc353550571)

[Таблица 3.1.2 Минимальные программные требования 24](#_Toc353550572)

[4. РЕАЛИЗАЦИЯ 25](#_Toc353550573)

[4.1 Компонент “WEB Client” 25](#_Toc353550574)

[Таблица 4.1.1.1 Элементы конфигурации 27](#_Toc353550575)

[4.1.1.1 Диаграмма классов 27](#_Toc353550576)

[Рисунок 4.1.2.1 Диаграмма классов 27](#_Toc353550577)

[Таблица 4.1.2.1 Описание классов 28](#_Toc353550578)

[ЗАКЛЮЧЕНИЕ 28](#_Toc353550579)

[СПИСОК ЛИТЕРАТУРЫ 28](#_Toc353550580)

[ПРИЛОЖЕНИЕ 1. 29](#_Toc353550581)

ВВЕДЕНИЕ

Ни для кого не секрет, что наше общество и экономика перешагнули стадию постиндустриализма и стали первооткрывателями новой ступени развития – стадии Информационного общества. С каждым днём растёт роль информации, знаний и новых технологий в жизни каждого современного человека. Рамки глобального информационного пространства расширяются с каждым годом, что позволяет людям эффективно взаимодействовать друг с другом, получать доступ к мировым информационным ресурсам и удовлетворять свои потребности в услугах.

Бурное технологическое развитие открыло миру огромное множество различных видов аппаратных и программных решений, которые в синтезе позволяют не только увеличить скорость обработки информации и сделать человека сверх коммуникабельным, но и оптимизировать затраты, связанные с эксплуатацией данных технологий.

На этом фоне появляются проблемы интеграции уже существующих продуктов, реализованных с помощью концептуально разных технологических средств и не имеющих общего, понятного всем протокола взаимодействия.

Ответной реакцией на такие удручающие перспективы послужило создание новых методологий и принципов, которые в итоге легли в основу универсальных интеграционных механизмов и стандартов.

В реальной жизни цель интеграции сводится к оптимизации каких-либо бизнес-процессов и/или объединение нескольких информационных ресурсов в одно целое. Во втором случае, количество до сих пор нерешённых задач пугает своим масштабом.

На данный момент существует много различных интернет ресурсов, на которых соискатель работы может выложить своё резюме в целях получить предложение о трудоустройстве от какой-либо заинтересованной в нём компании. Как правило, эти же ресурсы предоставляют аналогичную возможность публикации вакансий HR) отделам компаний. Как правило, соискатели\работодатели публикуют идентичные резюме\вакансии , сразу на нескольких ресурсах, чтобы увеличить вероятность ответных предложений\соглашений.

Как было отмечено ранее, на сегодняшний день существует большое количество рекрут-сайтов. Самые популярные из них:

* Русские: [headhunter.ru](http://headhunter.ru/) , [job.ru](http://job.ru/), [superjob.ru](http://superjob.ru/" \t "_blank), [rabota.ru](http://rabota.ru/" \t "_blank);
* Зарубежные: [monster.com](http://monster.com/), [dice.com](http://dice.com/);

Таким образом, для управления своим резюме\вакансией пользователю необходимо заходить на каждый сайт, где был опубликован профиль и проделывать одну и ту же работу, в целях обновления , по каким-либо причинам не актуальной на текущий момент, информации. Самые простые примеры таких ситуаций:

* Соискатель работы хочет добавить в своё резюме несколько новых навыков и изменить желаемую заработную плату;
* Работодателю необходимо изменить требования, предъявляемые к какой-либо должности.

Количество времени, требуемое для обновления информации, прямо пропорционально количеству ресурсов. Очевидно, что это далеко не оптимальный с точки зрения производительности и не удобный, из-за монотонности действий, способ. Но другого выбора у пользователей нет. Данные интернет ресурсы не предоставляют никаких программных интерфейсов для управления профилями и далеко не все поддерживают их импорт/экспорт. ы

В рамках данной курсовой работы будет реализована система синхронизации информации между рекрут - сайтами, услугами которых, пользуется пользователь.

# ПОСТАНОВКА ЗАДАЧИ

**Требуется разработать систему, которая должна предоставлять клиенту:**

1. **Программный интерфейс для взаимодействия с системой.**
2. **Правила формирования запроса к системе, включая формат схемы экземпляра профиля-резюме.**
3. **Возможность управлять пользовательским профилем (публикация/обновление резюме на указанных веб-ресурсах).**
4. **Надежный механизм хранения и синхронизации информации между системой и веб-ресурсами, поддерживающий асинхронную модель передачи данных.**

# АНАЛИЗ ЗАДАЧИ

* 1. Анализ предметной области

Прежде чем строить интеграционное решение, необходимо четко понять, какой подход будет наиболее эффективен для решения стоящей перед лицом задачи.

Существует три подхода к интеграции информационных систем:

1. Интеграция на уровне данных. Суть данного подхода заключается в следующем: приложения работают независимо друг от друга, каждое использует свой набор данных. В случае необходимости осуществляется обмен данными между приложениями. При этом, если обмен данными осуществляется путем вызова сервисов или отправки/получения сообщений.
2. Интеграция на уровне бизнес-процессов. Суть данного подхода заключается в следующем: приложения выставляют сервисы, являющиеся интерфейсами к бизнес - логике данных приложений. Взаимодействие между приложениями реализовано в рамках бизнес-процесса, на отдельных шагах которого осуществляется вызов того или иного сервиса. Реализуется данный подход с помощью сервисной шины предприятия (*ESB*), которая занимается виртуализацией сервисов, предоставляемых приложениями, и решений класса *Business Process Management System (BPMS)*, как правило основанных на языках *BPEL* или*BPMN*, которые реализуют логику процесса.
3. Интеграция на уровне композитных приложений. Бизнес-логика отдельного приложения строится путем вызова сервисов, предоставляемых как данным приложением, так и другими системами. Таким образом, на одном шаге бизнес-процесса могут взаимодействовать несколько сервисов, в то время как при интеграции на уровне бизнес-процессов на одном шаге процесса вызывается один сервис. Реализация композитных приложений осуществляется с помощью использования технологий *Java Business Integration (JBI, JSR 208)* или Service Component Architecture (SCA).

Отсюда можно сделать вывод, что наиболее целесообразным медом интеграции, в рамках поставленной задачи, является подход, описанный в первом пункте.

* 1. ESB как подход к интеграции на уровне данных

Самым оптимальным вариантом интеграции на уровне дынных является использование сервисной шины предприятия - Enterprise Service Bus (ESB).

Основной принцип сервисной шины — концентрация обмена сообщениями между различными системами через единую точку, в которой, при необходимости, обеспечивается транзакционный контроль, преобразование данных, сохранность сообщений. Все настройки обработки и передачи сообщений предполагаются также сконцентрированными в единой точке, и формируются в терминах служб, таким образом, при замене какой-либо информационной системы, подключённой к шине, нет необходимости в перенастройке остальных систем.

Важной особенностью сервисной шины является поддержка двух режимов передачи информации: синхронного и асинхронного.

В рамках поставленной задачи важно наличие именно второй характеристики, так её принцип обеспечивает надёжную передачу данных между приложениями, за счёт возможности использования сервисной шиной в качестве транспортного механизма технологии очередей сообщений (Message Queue). Таким образом, одно приложение посредством ESB может передать данные другому приложению без необходимости вызова процедуры получателя и определенно без ожидания результата. Отправитель не обязан знать, как найти получателя. Он может просто направить данные в ESB и быть уверенным, что они будут переданы.

* 1. Очереди Сообщений

Помимо возможности асинхронной передачи данных технология MQ обладает следующими важными свойствами:

* Слабое связывание — очереди сообщений создают неявные интерфейсы обмена данными, которые позволяют процессам быть независимыми друг от друга т.е вы просто определяете формат сообщений отправляемых от одного процесса другому.
* Избыточность — Очереди позволяют избежать случаев неэкономного использования ресурсов процесса(например памяти) в результате хранения необработанной (лишней информации) информации.
* Масштабируемость — очереди сообщений позволяют распределить процессы обработки информации. Таким образом, они позволяют легко наращивать скорость, с которой сообщения добавляются в очередь и обрабатываются.
* Эластичность и возможность выдерживать пиковые нагрузки — очереди сообщений могут выполнять роль своего рода буфера для накопления данных в случае пиковой нагрузки, смягчая тем самым нагрузку на систему обработки информации и не допуская ее отказа.
* Отказоустойчивость — очереди сообщений позволяют отделить процессы друг от друга, так что если процесс, который обрабатывает сообщения из очереди падает, то сообщения могут быть добавлены в очередь на обработку позднее, когда система восстановится.
* Гарантированная доставка — использование очереди сообщений гарантирует, что сообщение будет доставлено и обработано в любом случае (пока есть хотя бы один обработчик).
* Гарантированный порядок доставки — большая часть систем очередей сообщений способны обеспечить гарантии того, что данные будут обрабатываться в определённом порядке (чаще всего в том порядке в котором они поступили).
* Буферизация — очереди сообщений позволяет отправлять и получать сообщения при этом работая с максимальной эффективностью, предлагая буферный слой — процесс записи в очередь может происходить настолько быстро, насколько быстро это в состоянии выполнить очередь сообщений, а не обработчик сообщения.
* Понимание потоков данных — очереди сообщений позволяют выявлять узкие места в потоках данных приложения, легко можно определить какая из очередей забивается, какая простаивает и определить что необходимо делать — добавлять новых обработчиков сообщений или оптимизировать текущую архитектуру.
  1. Основные термины

|  |  |
| --- | --- |
| **Термин** | **Описание** |
|  |  |

Таблица 2.2.1 Основные термины

* 1. Анализ требований

**Для реализации всей необходимой функциональности (пункт “Постановка задачи”) данная система должна состоять из четырёх основных компонентов (**Рисунок 2.3.1**):**

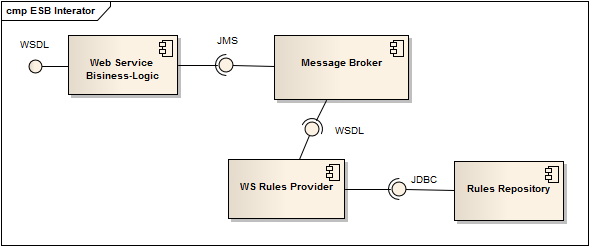


Рисунок 2.3.1 Диаграмма Компонентов

1. WS Business-Logic – компонент, предоставляющий клиенту универсальных программный интерфейс к бизнес-логике системы.
2. Message Broker – главный компонент системы, представляющий собой messaging-платформу.
3. WS Rules Provider – сервис, предоставляющий компоненту «Message Broker» набор бизнес-функций, связанных с управлением workflow-правилами системы.
4. Rules Repository – хранилище правил, по которым система производит трансформацию и оркестрацию потока запросов.

Для реализации данной системы целесообразно использовать стек Java - технологий, так как они позволяют строить кроссплатформенные и масштабируемые решения. Кроме того, на сегодняшний день существует довольно широкий спектр фрэймворков для быстрого и удобного построения программных решений, основанных на этой платформе.

* + 1. Компонент “ Web Service Business-Logic ”

В целях построения гибко расширяемой архитектуры системы, компоненты презентационного уровня и уровня бизнес-логики системы, должны быть максимально независимы друг от друга (Риунок диаграмма компонентов). Данное требование, в рамках сервис-ориентированной архитектуры (SOA – Service Oriented Architecture), может быть реализовано на основе технологии веб-сервисов.

На сегодняшний день наибольшее распространение получили следующие протоколы реализации веб-сервисов:

* SOAP (Simple Object Access Protocol) - WSDL, UDDI;
* REST (Representational State Transfer);
* XML-RPC (XML Remote Procedure Call).

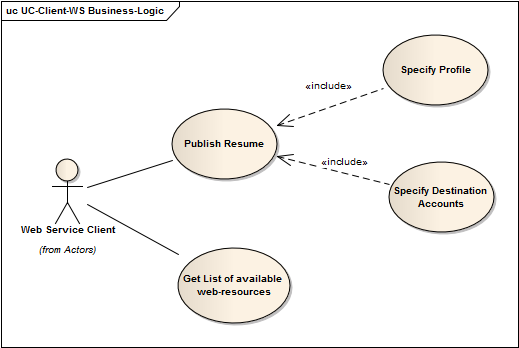
В рамках поставленной задачи, целесообразно использовать первый протокол, так как SOAP более применим в сложных архитектурах, где взаимодействие с объектами выходит за рамки теории CRUD  (Create Read Update Delete), и требуется обеспечение надёжности и безопасности передачи данных.

Таким образом, клиент должен иметь возможность обращаться в бизнес-функциям системы посредством WSDL файла, опубликованного сервисом.

* + - 1. Диаграмма вариантов использования

Как видно из рисунка 1, сервис должен предоставлять клиенту следующие возможности:

* Получить список всех доступных веб-ресурсов рекрутмента, для которых система реализует интеграцию.
* Выполнить запрос о публикации резюме, предварительно указав необходимую информацию о аккаунтах веб-ресурсов (логин/пароль) и данные о публикуемом профиле.

****

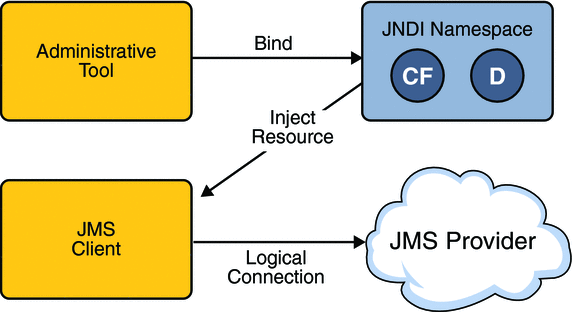
* + 1. Компонент “Message Broker ”

Взаимодействие сервиса, к которому имеет доступ клиент, с системой должно осуществляться посредством **Java Message Service**  (JMS).

JMS - это спецификация J2EE технологии, определяющая набор интерфейсов к системам, ориентированных на работу через сообщения (message-oriented application programming). Таким образом, программа, написанная с использованием JMS, будет корректно работать с любой системой сообщений, поддерживающей эту спецификацию (или имеющую соответствующие интерфейсы).

* + - 1. Архитектура и общие термины JMS

Архитектура JMS выглядит следующим образом (Рис. 1.):



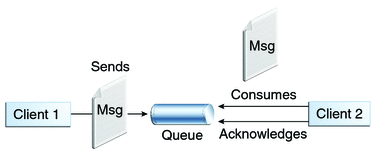
* JMS client - Прикладные программы Java, использующие JMS;
* JMS-провайдером (JMS provider) - Система обработки сообщений, управляющая маршрутизацией и доставкой сообщений,
* Приложение JMS (JMS application) – это прикладная система, состоящая из нескольких JMS клиентов, и, как правило, одного JMS-провайдера. JMS-клиент, посылающий сообщение, называется поставщиком (producer). JMS-клиент, принимающий сообщение, называется потребителем (consumer). Один и тот же JMS клиент может быть одновременно и поставщиком и потребителем в разных актах взаимодействия;
* Сообщения (Messages) – это объекты, передающиеся и принимающиеся компонентами (клиентами JMS);
* Средства администрирования (Administrative tools) – средства управления ресурсами, использующимися клиентами.

JMS предоставляет два подхода к передаче сообщений:

* «издание-подписка» (publish an subscribe);
* «точка-точка» (point to point) .

Спецификация JMS называет эти два подхода зонами сообщений (Messaging Domains).

* + - 1. Модель взаимодействия точка-точка

Модель передачи сообщений «точка-точка» предоставляет возможность клиентам JMS посылать и принимать сообщения (как синхронно, так и асинхронно) через виртуальные каналы, называемые очередями (queues). Модель передачи сообщений «точка-точка» основывается на методе опроса, при котором сообщения явно запрашиваются (считываются) клиентом из очереди. Несмотря на то, что чтение из очереди могут осуществлять несколько клиентов, каждое сообщение будет прочитано только единожды - провайдер JMS это гарантирует. 

* + - 1. Модель взаимодействия издание-подписка

При использовании модели взаимодействия «издание-подписка» один клиент (поставщик) может посылать сообщения многим клиентам (потребителям) через виртуальный канал, называемый темой (topic). Потребители могут выбрать подписку (subscribe) на любую тему. Все сообщения, направляемые в тему, передаются всем потребителям данной темы. Каждый потребитель принимает копию каждого сообщения. Модель передачи сообщений издание-подписка, по существу, представляет собой модель, сервера, инициирующего соединение и «проталкивающего» информацию на клиента. В JMS эта концепция реализуется с помощью специальных «слушателей» (Listener), регистрируемых в системе. При возникновении нового события Listener, закрепленный за данной темой, возбуждается. Следует отметить, что при использовании модели «издание-подписка» клиенты JMS могут устанавливать долговременные подписки, позволяющие потребителям отсоединиться и позже снова подключиться и получать сообщения, поступившие во время отключения связи.



* + - 1. Требования к JMS провайдеру

Поскольку JMS является лишь оболочкой или интерфейсом, описывающим доступные для приложения методы, для работы приложения понадобится определенная реализация JMS, называемая провайдером JMS API.

В рамках данной курсовой работы целесообразно использовать провайдер, который относится к списку открытых решений:

* ActiveMQ (Apache)
* Fuse MQ (Red Hat)
* OpenJMS ( The OpenJMS Group)
* JBoss Messaging ( JBoss)
* JORAM (OW2)

Что касается функциональных и технических требований – важно чтобы провайдер предоставлял следующие возможности:

* Конфигурирование, администрирование очередей сообщений посредством консоли управления (желательно с графической оболочкой).
* Предоставление программного интерфейса к бизнес-функциям провайдера сторонним системам. Наличие данной характеристики позволит расширить стандартную функциональность посредством сторонних открытых решений.
* Предоставление специализированных инструментов для мониторинга характеристик производительности JMS-системы.
* Распределение нагрузки и отказоустойчивость.
* Поддержка системы сообщений и уведомлений об ошибках.
* Независимость от протокола связи.
  + - 1. Требования к созданию профиля резюме

На начальном этапе внедрения системы, необходимо предоставить пользователю возможность публикации/синхронизации своего резюме по крайней мере на двух самых популярных рекрут-сайтах: hh.ru и master.com.

Что касается создания резюме, то тут пользовательский интерфейс должен предоставлять все необходимые возможности для внесения информации, которая впоследствии будет преобразована в соответствии с форматом представления её на других рекрут-ресурсах.

* + 1. ESB Integrator

1. ****АППАРАТНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА****

В качестве средств реализации были выбраны следующие программные продукты и технологии:

|  |  |
| --- | --- |
| **Технология** | **Описание** |
| **Fuse IDE** | Среда разработки ESB решения |
| **JUnit 4.0** | Технология тестирования |
| **Maven 2.0** | Инструмент для сборки |
| **Git** | Система контроля версий |
| **GitHub.com** | Хостинг – сервис для CVS Git |

Таблица 3.1 Средства реализации

* 1. Требования к составу и параметрам технических средств

|  |  |  |
| --- | --- | --- |
| **Требование** | **Минимальное значение** | **Рекомендуемое значение** |
| **Оперативная память** | 128 Мб | 1 Гб и более |
| **Свободное пространство на ЖД** | 300 Мб | 1 Гб и более |
| **Процессор** | 533 МГц | 1 Гб и более |

Таблица 3.1.1 Минимальные системные требования

* 1. Требования к программным средствам, используемым системой

Java Virtual Machine + Операционная система:

* Microsoft Windows;
* Linux;
* Mac OS.

Инструменты необходимые для разработки:

1. Eclipse IDE.
2. Plugin Development Kit.

|  |  |  |
| --- | --- | --- |
| **Требование** | **Минимальное значение** | **Рекомендуемое значение** |
| **Версия Java JDK , JRE** | 1.6. | 1.7.0 и выше |
| **Версия Java3D API 1.5** | 1.5 | 15.2 и выше |
| **Версия Eclipse** | 3.4 | 3.7 и выше |

Таблица 3.1.2 Минимальные программные требования

1. ****РЕАЛИЗАЦИЯ****
   1. Компонент “WEB Client”

В Java существует несколько технологий для разработки пользовательского web-интерфейса :

* JavaServer Page (JSP) -  это технология Java 2 Platform, Enterprise Edition (J2EE) для создания приложений, генерирующих динамическое web-содержимое - HTML, DHTML, XHTML и XML.
* JavaServer Faces (JSF) – технология построения интерфейса в концепции MVC ( Model-View-Controller).

Для решения поставленной задачи решено было использовать технологию JSF, так как она использует подход MVC к построению архитектуры приложения и предоставляет эффективный компонентный подход к разработке.

В приложениях такого рода контроллер отвечает за прием данных от пользователя и выдачу соответствующего представления, представление формирует для браузера HTML страницу, а модель содержит данные, полученные из веб-форм и ту информацию, которую необходимо вывести на экран. Этот подход позволяет оградить уровень представления данных от бизнес-логики. Фреймворков, поддерживающих MVC, имеется достаточно, но JSF — стандарт для приложений Java EE, и в этом его преимущество. Во-первых, технология активно развивается Oracle, а во-вторых, для нее на данный момент создано множество библиотек, позволяющих использовать нестандартные UI компоненты, основанные на jQuery.

Если рассуждать формализованными терминами языка, то:

***Представление*** — это файл**\*.JSF** или **\*.XHTML**, отвечающий за вывод данных в браузер и содержащий ссылки на конкретные данные в модели.

***Модель*** — JavaBean, хранящий ту или иную информацию в приватных полях и предоставляющий для них геттеры/сеттеры наряду с методами обработки такой информации.

***Контроллер*** — это и есть внутренний механизм JSF, позволяющий провести линковку первого со вторым.

Страницы JSF, формирующиеся из XML тэгов. Каждый тэг представляет конкретный UI-компонент. Веб-разработчику не нужно вдаваться в написание HTML разметки или вставок на JavaScript, так как они полностью генерируются компонентными тэгами JSF. Так как каждый компонент по-сути независим и содержит определенное поведение (то есть «знает», как получить свои данные и отрисовать себя в браузере), JSF предоставляет подход к программированию UI, очень похожий на принцип оперирования POJO.

Динамические данные на JSF страницах моделируются с помощью POJOs, называемых управляемыми компонентами (JSF Manged Beans). Его жизненный цикл управляется контейнером.

Для быстрой разработки пользовательского интерфейса было решено использовать библиотеку готовых графических компонентов, значительно расширяющие функциональность фреймворка, а именно: **PrimeFaces**.

**Код Пример шаблона**

**Код Пример формы.**

* + - 1. Диаграмма классов

При реализации архитектуры модуля , были использованы следующие паттерны проектирования:

* Singleton;
* Abstract Factory;
* Façade.

Рисунок 4.1.2.1 Диаграмма классов

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
|  |  |

Таблица 4.1.2.1 Описание классов

* 1. **Компонент “WS Business-logic”**

**3.2.1 Диаграмма классов**

* 1. **Компонент “ESB Integrator”**

# ****ЗАКЛЮЧЕНИЕ****

# ****СПИСОК ЛИТЕРАТУРЫ****

<http://khpi-iip.mipk.kharkiv.edu/library/extent/prog/jsf/jsf.html>

<http://sait-com.com/articles/javaee/3525-obzor-texnologii-jsf.html>

# ****ПРИЛОЖЕНИЕ 1.****