ВВЕДЕНИЕ

В наш век высоких технологий, в нашей стране сложно приставить жизнь людей без информационных технологий. Каждый хочет прийти в больницу, школу, райисполком, горисполком, словом, в любое госучреждение и не стоять в очереди, а просто получить ту бумажку (справку, ну, или документ, если угодно), которая ему нужна. В Республике Беларусь используется государственная информационная система социальной защиты, сокращенно ГИССЗ. Она представляет собой огромных размеров реляционную базу данных, с множеством встроенных функций, которые позволяют вычислять пособия по уходу за ребенком, по инвалидности, пенсии, словом различного рода выплаты, установленные действующим законодательством Республики Беларусь. На самом деле я еще не разобрался во всех тонкостях работы с ГИССЗ, с тем, как связаны между собой таблицы, потому как их там туча, и даже если просто взять какой-нибудь data modeler, и смоделировать структуру связей ­– ничего не получится. Вернее получится, получена каша из таблиц и стрелочек между ними. Соответственно, что бы вытянуть что то полезное их ГИССЗ нужно заручиться поддержкой человека, который знает базу, или воспользоваться какой либо специальной программой, или, как понятно из названия дипломного проекта, сервисом.

В настоящее время, становятся популярны различные веб технологии, веб-сайты, веб-интерфейсы, веб-сервисы. И это действительно очень удобно. Если об удобстве сайтов все итак знают, так как огромное число людей пользуются различными социальными сетями, наподобие facebook’а, вконтакте; поисковиками, как то google, yandex, то об удобстве веб-сервисов знают немногие. Скажем, у нас есть какая-то специальная программа, в которую зашит какой-то принцип обращения к базе данных. Если вдруг в целях какой-либо модернизации или оптимизации база данных будет изменена, то во-первых, очевидно, разработчику нужно модифицировать данное программное обеспечение, а во-вторых всем пользователям этой программы следует обновить ее. А как обновить? Скачать свежую версию? Возможно. Программа может так же содержать модуль обновления внутри себя. Да, это неплохой вариант, если программа сильно использует аппаратные средства на клиентских компьютерах или хранит какие то данные на ней же. Примеров такого масса, да что там, целая туча: игры наподобие world of warcraft, world of tanks; операционные системы, сервера баз данных и др. Но, что если мы имеет дело с небольшим интерфейсом, который прогнозирует погоду, следит за расписанием поездов или самолетов, составляет граф пользователей какой то социальной сети? В таком случае, есть несколько вариантов реализации такого данного интерфейса. Если, скажем, он будет следить за расписанием самолетов в национальном аэропорте, то можно написать парсер html-страничек сайта аэропорта. Однако в таком случае, если разработчики сайта изменят немного структуру, то интерфейс автоматически станет не рабочим, или по крайней мере работающим неверно. Но есть другой путь. Можно использовать предоставленный интерфейс программирования приложений(сокращенно ИПП), того ресурса, который хотим использовать(веб-ИПП, если ресурс расположен в интернете). К примеру, такой интерфейс предоставляет социальная сеть вконтакте. В таком случае, если даже разработчики поменяют структуру сети, то так же они изменят и реализацию функций данного интерфейса так, что бы они оставались актуальными. Таким образом клиентские приложения обновлять не придется.

Такой вот веб-ИПП является практически синонимом веб-сервиса. Только ИПП, по сути, является библиотекой для какого языка программирования, а веб-сервис работает за счет обмена сообщениями какого либо формата. То есть нет зависимости от того на каких языках программирования написаны веб-сервис и клиентские приложения: клиент на c# может спокойно использовать веб-сервис на java.

Целью данного дипломного проекта является разработка и реализация системы веб-сервиса взаимодействия ГИССЗ с АСУ «Занятость». В первую очередь данная система предназначена для представления определенной информации данной АСУ.

В соответствии с поставленной целью были определены следующие задачи:

* выбор протокола обмена сообщениями;
* изучение структуры базы данных ГИССЗ;
* описание класса для работы с базой данных ГИССЗ;
* разработка класса для проверки входных параметров запроса на предмет наличия в них вредоносных элементов, таких как sql-инъекции;
* разработка класса, разворачиваемого на сервере, который и будет представлять ядро веб-сервиса.
* разработка функции, вычисляющей электронную цифровую подпись (ЭЦП).

Сервис будет состоять из серверного веб-приложения, предоставляющего следующие функции:

* предоставление информации о получении социальной выплаты;
* предоставление информации о размере получаемой пенсии (пособия), в том числе за определенный месяц, год;
* предоставление информации о назначении пенсии;
* защита пересылаемых данных с помощью ЭЦП.

**1** ОБЗОР ЛИТЕРАТУРЫ

1.1 Аналитический обзор

В общем понимании, технология web-сервисов – этой технология создания распределенных систем, собранных из взаимодействующих между собой программ, созданных и работающих на основе различных платформ. Web-сервисы могут согласовывать работу больших, состоящих из множества частей приложений, предоставляя для приложения функции обмена данными. Однако помимо этого, web-сервисы могут быть повторно-используемыми приложениями, представляющими различные услуги, такие как прогноз погоды, расписание поездов, получения талончика в поликлинике, или как в нашем случае выдача справок из государственной информационной системы.

Web-сервисы можно разделить на три группы:

* web-сервисы, взаимодействие с которыми производится с помощью XML-сообщений по SOAP-протоколу (Simple Object Access Protocol) и имеющие интерфейсы, описанные в формате WSDL (Web Services Description Language). Такое описание интерфейса сервиса позволяет клиентам генерировать в автоматическом режиме код, необходимый для работы с сервисом. Данное описание может быть доступно клиенту с реестра UDDI (Universal Description, Discovery and Integraition), в которым сервис заранее зарегистрирован.
* XML web-сервисы, ориентированные на сообщения. Такие веб-сервисы имеют низкоуровневую обработку XML сообщений, обрабатывая полученные XML данные как они есть, и полностью формирует ответное XML сообщение. Такие web-сервисы могут передавать и получать сообщения как в формате SOAP так и в чистом XML-формате.
* RESTful web-сервисы, представляющие удаленные ресурсы, которые доступны с помощью HTTP-запросов. Данные web-сервисы обеспечивают взаимодействие с удаленными ресурсами, передавая клиенту их представления. RESTful web-сервисы идентифицируются URL-адресом и обрабатывают HTTP-методы GET, PUT, POST и DELETE в ответ на запрос клиента. Технология REST web-сервисом может обходиться и без WSDL-описания[1].

1.2 SOAP-сервис

SOAP – протокол обмена сообщениями, структурированными определенным образом. Первоначально, он нужен был по большей части для удаленного вызова процедур. Сейчас же, протокол также используется для обмена сообщениями в формате XML. Данный протокол может использоваться с протоколами прикладного уровня, но чаще всего используется поверх HTTP. SOAP является одним из стандартов, на которых основывается разработка веб-сервисов. Сообщение SOAP – это простой XML-документ, который состоит из нескольких элементов. Структура SOAP-сообщения показана на рисунке 1.1.

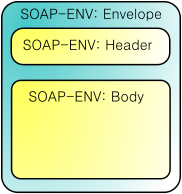


Рисунок 1.1 – Структура SOAP-сообщения

Как видно, сообщение состоит из следующих элементов:

* Envelope – Обязательный элемент, является самым «верхним» элементов SOAP сообщения. Является корневым элементом XML документа. У данного элемента могут быть атрибуты xmlns, определяющие пространства имен, и другие атрибуты, снабженные префиксамми.
* Header – Опциональный элемент, который кроме атрибутов xmlns может содержать 0 или более стандартных атрибутов, таких как encodingStyle, actor (или role для версии 1.2), mustUnderstand, relay.
* Body – Обязательный элемент, который включает в себя данные XML, которые определяют сообщение, которое должно быть отправлено. В данный элемент можно вложить произвольные элементы, спецификация никак не определяет их структуру. Определен лишь один стандартный элемент, который может быть в теле сообщения – Fault, содержащий сообщение об ошибке.

В SOAP-сообщениях могут передаваться данные различных типов (строки, даты, массивы, числа и т.п.). Определение типов данных выполняется в схемах XML.

Типы, определенные в схеме, заносятся в пространство имен, идентификатор которого служит значением атрибута encodingStyle. Данный атрибут может появиться в любом SOAP-сообщении, однако версия протокола 1.2 запрещается его для элемента envelope. Указанное пространство имен будет известно в том элементе, в котором записан атрибут, и во всех вложенных элементах.

Атрибут actor задает целевой SOAP-узел – тот, который расположен в конце цепочки и будет обрабатывать заголовок полностью. Данный атрибут может встречаться в отдельных блоках заголовка, указывая узел-обработчик этого блока. После обработки блок удаляется из SOAP-сообщения.

Атрибут mustUnderstand указывает, должен ли SOAP-узел при обработке элемента учитывать его синтаксис, определенный в схеме документа, или вообще не обрабатывать сообщение. Тип значений этого атрибута boolean, то есть принимает значения 0 или 1 (в версии 1.2 вместо цифр нужно употреблять true или false). Правильное использование данного атрибута повышает точность обработки сообщения.

Атрибут relay показывает, что заголовочный блок, адресованный SOAP-посреднику, должен быть передан дальше, если он не был обработан. Тип его значения boolean.

Если сервис, обрабатывая SOAP-сообщение, обнаруживает какую-либо ошибку, то он прекращает формирование ответа и отправляет клиенту сообщение, содержащие в теле один элемент Fault с сообщением об ошибке. В версии 1.1 элемент Fault имел 4 дочерних элемента:

* код ошибки faultcode – используется программами, обрабатывающими ошибки;
* описание ошибки faultstring – словесное описание, предназначенное для человека;
* место обнаружения ошибки faultactor – адрес сервера, который зафиксировал ошибку;
* детали ошибки detail – описывает ошибки, зафиксированные в теле элемента body. Если в body не было найдено ошибок, то данный элемент отсутствует.

В версии 1.2 содержание элемента fault изменилось:

* код ошибки code – содержит обязательный вложенный элемент value с кодом ошибки и не обязательный subcode, с вложенным в него value с дополняющим кодом ошибки и элемент subcode, и далее все повторяется рекурсивно;
* причина ошибки reason. Обязательный элемент, содержащий необязательный атрибут xml:lang, указывающий язык сообщения, и любое число вложенных элементов, содержащих словесное описание ошибки;
* node – не обязательный элемент, содержащий адрес промежуточного узла, заметившего ошибку;
* role – так же не обязательный элемент. Указывает роль узла, заметившего ошибку[2, 3];

## 1.3 WSDL описание сервиса

Web Services Description Language, или сокращенно WSDL – это XML-ориентированный язык, который предназначен для определения (описания) web-сервиса и доступа к нему. Он публикуется в открытом доступе (регистрируется в каталоге служб). Клиенты, которым нужна определенная web-служба, выполняют поиск по каталогу. Найдя нужный сервис, они импортируют его WSDL описание, на основе которого генерируют интерфейс, с помощью которого и строится клиентское приложение. Каждый документ WSDL состоит из четырех основных логических частей:

* portType – методы, предоставляемые web-сервисом. Данный элемент является наиболее важным, так как он определяет сам web-сервис, предоставляемые им операции и используемые сообщения;
* message – сообщения, используемые web-сервисом;
* types – типы данных, используемые web-сервисом. Для максимальной независимости от платформы WSDL использует синтаксис XML Schema для определения типов данных;
* binding – определяет формат сообщения и детали протокола для каждого порта.

Структура и различия версий WSDL 1.1 и 2.0 представлены на рисунке 1.2.



Рисунок 1.2 – Структура WSLD-описания

Пример простейшего фрагмента WSDL-описания представлен на рисунке 1.3.

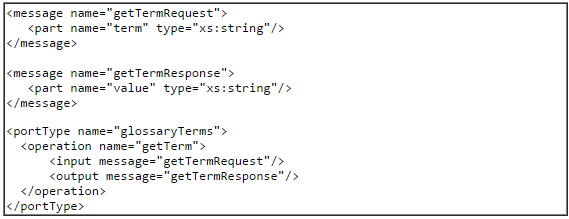


Рисунок 1.3 – Пример фрагмента WSDL-описания

Как видно, в этом фрагменте элемент portType определяет glossaryTerms как имя порта, а getTerm как имя операции. Определены два сообщения, с которыми может работать web-сервис: getTermRequest и getTermResponse. Но именно в части portType операция, которая в ней определена, сообщает, что первое сообщение посылается web-сервису, а второе возвращает web-сервис как ответ на первое.

На данном примере была представлена операция запрос-ответ, однако в WSDL определено четыре типа операций:

* однонаправленные – операция принимает сообщения, но ответа не последует;
* запрос-ответ – операция принимает запрос и возвращает ответ;
* вопрос-ответ – операция может послать запрос и будет ждать ответ;
* извещение – операция может послать сообщение, но не будет ждать ответ.

Связь с SOAP реализуется в части binding с помощью элемента soap:binding, который имеет два атрибута:

* style – rpc или document
* transport – определяет используемый протокол SOAP. К примеру «http://schemas.xmlsoap.org/soap/http» определяет протокол HTTP[4, 5].

1.4 Алгоритмы хеш-функций в ЭЦП.

Электронная цифровая подпись – реквизит электронной информации (документа), полученный после некоторого криптографического преобразовании информации с помощью закрытого ключа подписи и позволяющий проверить отсутствие искажения (подделки) информации и авторства этой информации. Существует множество различных алгоритмов по вычислению электронной цифровой подписи. Прочитав несколько статей, было решено использовать подписи, основанные на двух алгоритмах:

* message digest 5 (md5);
* secure hash algorithm 1 (sha 1).

Алгоритм md5 состоит из пяти шагов:

1. Append padding bits. Исходное сообщение дополняют так, что бы его длина (в битах) была сравнима с 448 по модулю 512. В сообщение добавляют единичный байт (0x80), а затем добавляют нулевые биты, до тех пор, пока не будет выполняться поставленное выше условие.
2. Append length. К результату предыдущего шага добавляется 64-разрядное представление длины исходного сообщения до выполнения предыдущего шага. В маловероятном случае, когда длина сообщения больше чем 264 используются лишь младшие 64 бита длины.
3. Initialize MD buffer. На этом шаге инициализируется буфер, состоящий из четырех слов:

* *A*: 01 23 45 67;
* B: 89 ab cd ef;
* C: fe dc ba 98;
* D: 76 54 32 10.

1. Process message in 16-word block. На этом шаге определяются четыре вспомогательные логические функции, работающие над 32-битными словами:
   * ;
   * ;
   * ;
   * .

Так же на этом этапе используется 64-элементная таблица, построенная по синусной функции. Пусть T[i] обозначает i-й элемент таблицы, значения элементов данной таблицы будут равны . Далее копируем каждый 16-битный блок в массив X[16] и производим манипуляции:

* ;
* ;
* ;
* .

Далее происходят преобразования-раунды, которых будет всего четыре. Каждый раунд состоит из 16 преобразований, которые можно представить как , где:

* + *A*, *B*, *C*, *D –* регистры;
  +  *–* одна из логических функций;
  +  *–* k-тый элемент 16-битного блока
  + *–* операция циклического сдвига на *s* позиций влево.

Затем суммируем результаты вычислений:

* + ;
  + ;
  + ;
  + .

1. Output. Искомая подпись будет состоять из *A*, *B*, *C*, *D*, где *A* – младший байт, а *D* – старший соответственно[6, 7].

Схема работы алгоритма md5 представлена на рисунке 1.4.

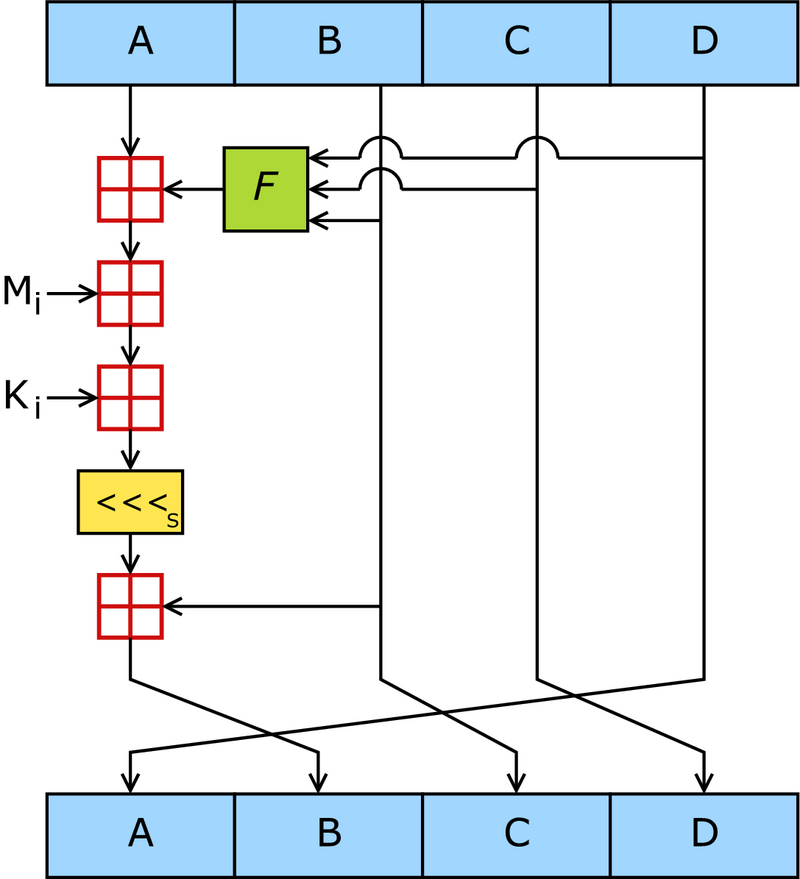


Рисунок 1.4 – Схема работы алгоритма md5

Алгоритм sha 1. Для сообщения произвольной длины (до двух эксабайт) генерирует 160-битной значение. Вычисление подписи производится следующим образом:

1. Message padding. Исходное сообщение разбивается на блоки по 512 бит в каждом. Последний блок дополняется до длины 512 бит. Дополняется аналогично алгоритму md5: сначала добавляется 1, а затем нули.
2. Functions and constants. Определяются четыре нелинейные операции и четыре константы. Так же на данном этапе инициализируются пять 32-х битных переменных:
   * ;
   * ;
   * ;
   * .
   * ;
   * ;
   * ;
   * ;
   * .
3. Computing the message digest. Вычисление дайджеста происходит в несколько шагов:

* разделяем каждый блок в 512 бит исходного сообщения на 16 блоков ;
* далее вычисляем для *t* от 16 до 79 , где  – циклический сдвиг влево;
* для *t* от 0 до 79 вычисляем временное значение *temp* , а так же присваиваем переменной *e* значение переменной *d,* переменной *d* значение *с*, переменной *c* значение *b* со сдвигом влево на 30 позиций, переменнойb значение a, а переменной *a* значение *temp*;
* после этого значения переменных *a*, *b*, *c*, *d*, *e* прибавляются к *A*, *B*, *C*, *D*, *E.*

Итоговым значением дайджеста будем считать 160-битную строку, образованную значениями *A*, *B*, *C*, *D*, *E*[8, 9]. Не так давно было несколько публикаций, в том числе и от известной всем google, о том, что алгоритм sha 1 не столь безопасен, как казалось лет пять назад. Возможно, что на черном рынке может появиться услуга по поиску коллизий sha 1, однако услуги такого рода не стоят дешево и не выполняются быстро даже на многопроцессорных кластерах. Полагается, что тому, кому понадобится взлом данного хеша, не хватит денег для этого. Стоит отметить, что существует алгоритм sha 2, которая используется в различных областях, таких как Digital Signature Algorithm (DSA) и bitcoin. Исследования данной хеш-функции на криптостойкость не выявили каких-либо уязвимостей, что делает ее более привлекательной для использования, однако алгоритмы sha 2 работают в 2-3 раза медленней чем популярные md5 и sha 1, а для разрабатываемого web-сервиса скорость работы стоит не на последнем месте[10, 11].

Схема работы алгоритма SHA 1 представлена на рисунке 1.5.

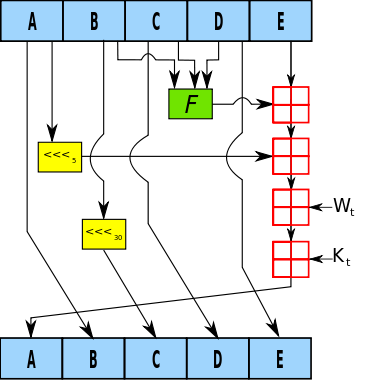


Рисунок 1.5 – Схема работы алгоритма SHA 1

**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив теоретические сведения из подобранной специальной литературы и составив список функций, которые предоставляет разрабатываемый веб-сервис, разобьем сервис на отдельные функциональные блоки (модули). Такой подход позволяет изменять или заменять модули без изменения всей системы в целом.

В проектируемом веб-сервисе можно выделить следующие блоки:

* блок начальной конфигурации;
* блок формирования ответа;
* блок работы с базой данных;
* блок базы данных ГИССЗ;
* блок авторизации;
* блок формирования ЭЦП;
* блок получения запроса и отправки ответа;
* блок проверки запроса.
* блок формирования пакета об ошибке.

Структурная схема, показывающие перечисленные выше блоки и связи между ними приведена на чертеже ГУИР.400201.010 C1.

Каждый отдельный блок выполняет свою функцию. Рассмотрим функциональные блоки веб-сервиса.

*Блок начальной конфигурации* представляет собой функцию, в которой создаются все необходимые экземпляры классов, заполняются структуры данных, которые нужны для работы остальных блоков веб-сервиса, к примеру методы хеширования и путь к файлу с хешами, у которых есть доступ к функциям сервиса, версия и путь к драйверу jdbc, выполняется развертывание веб-сервиса на сервере. Словом, данный блок отвечает за подготовку веб-сервиса к работе.

*Блок формирования ответа* выполняет заполнение структуры данных, которая будет отправлена клиенту. Вначале управление передается в блок авторизации, так как для того что бы получить какую-либо информацию из базы данных, нужно установить соединение с ней, то есть авторизоваться. После успешной авторизации запускается одна из нескольких функций для формирования ответа, так как типов запросов несколько, соответственно и блок состоит из нескольких независимых функций. Вызывая определенные функции блока работы с базой данных, а так же, выполняя расчеты по нужным статьям (скажем, размеры полученных средств человеком за несколько месяцев, за год) одна из данных функций заполняет поля ответа. После того как все нужные поля ответного сообщения были заполнены, управление передается в блок формирования ЭЦП, что бы подписать ответ.

На рисунке 2.1 показан фрагмент структурной схемы, на котором отображена взаимосвязь блока формирования ответа с другими блоками системы.

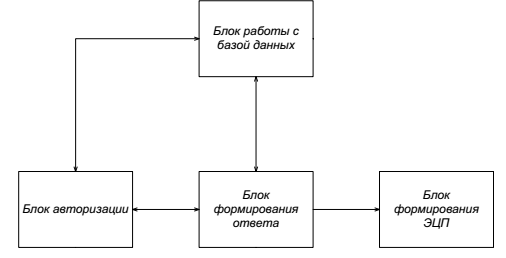


Рисунок 2.1 – Взаимосвязь блока формирования ответа с другими блоками системы

*Блок работы с базой данных* представляет собой класс, в функции которого входит установление соединения с базой данных ГИССЗ, а так же составление правильного запроса на языке sql, с помощью которого можно получить нужную для формирования ответа веб-сервиса информацию.

*Блок базы данных ГИССЗ* представляет собой объектно-реляционную систему управления базами данных (сокращенно СУБД) компании Oracle, содержащую данные о людях Республики Беларусь. Она включает в себя множество таблиц, порядка пятисот сорока, и процедур и функций. Данная СУБД использует технологию «клиент-сервер», что позволяет клиенту и серверу обмениваться минимальной нужной информацией, в то время как основная нагрузка ложится на сервер. Такой подход в большинстве случаев менее требователен к пропускной способности компьютерной сети, в отличие от файл-серверных СУБД, таких как «Microsoft access», особенно при выполнении операций по поиску данных по пользовательским параметрам, так как для этого нет нужды в получении клиентом всего массива данных сервера. Клиент лишь передает нужные для поиска параметры сервера, а сервер по полученному запросу производит поиск в своей локальной базе данных. Достоинствами такой организации базы данных являются:

* более низкая нагрузка локальной сети;
* удобство централизованного управления;
* высокая надежность, доступность и безопасность.

К недостаткам можно отнести лишь повышенные требования к серверу, так как очевидно, клиентов может быть подключено масса, а сервер один. Как было указано выше, в качестве системы управления базой данных для ГИССЗ была выбрана СУБД oracle. Это произошло из-за того, что данная система управления базами данных является лучшей в мире, ей присущи различные особенности, такие как:

* MVCC – многоверсионность данных для управления параллельными транзакциями, что позволяет предоставлять каждому пользователю «слепка» БД, и вносимые пользователем изменения в БД не затрагивают других пользователей до момента фиксации транзации;
* секционирование, что повышает управляемость, производительность и доступность для больших баз данных;
* поддержка последовательностей
* пакеты – это дополнительное пространство имен, в которое можно объединять процедуры и функции;
* сильный оптимизатор;
* откат базы данных – позволяет откатывать все базу данных или таблицу как бы назад во времени, если внесенные изменения привели к неверной или незапланированной работе базы.

*Блок авторизации данных* включает в себя проверку логина и пароля, из полученного запроса. Осуществляется это следующим образом:

* выполняется конкатенация значений логина пароля;
* для полученной строки с помощью выбранного в блоке начальной конфигурации типа хеш-функции: md5 или sha-1;
* проверяется наличие полученного значения в файле, хранящем значения хеш-функций, которые получаются из данных пользователей, которым доступно соединение с базой данных;
* с использованием блока для работы с базой данных выполнятся попытка соединения с базой данных;
* если соединение не было установлено, формируется и отправляется клиенту сервиса сообщение об ошибке, в противном случае начинает формироваться ответ на запрос.

На рисунке 2.2 показан фрагмент структурной схемы, на котором отображена взаимосвязь блока авторизации с другими блоками системы.

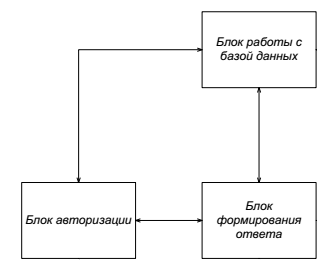


Рисунок 2.2 – Взаимосвязь блока авторизации с другими блоками системы

*Блок формирования ЭЦП* отвечает за формирование электронной цифровой подписи сформированного ответа. Этот блок позволяет веб-сервису защитить ответ от подделки, осуществить контроль целостности передаваемого клиенту ответа, доказать, что ответ был сформирован и получен клиентом именно от данного веб-сервиса. Блок будет представлять собой класс, который содержит несколько методов, реализующих вычисление ЭЦП на основе двух алгоритмов: sha-1 и md5. Открытым методом будет лишь один, позволяющий подписать ответ. Выбор алгоритма осуществляется в блоке начальной конфигурации.

На рисунке 2.3 показан фрагмент структурной схемы, на котором отображена взаимосвязь блока формирования ЭЦП с другими блоками системы.

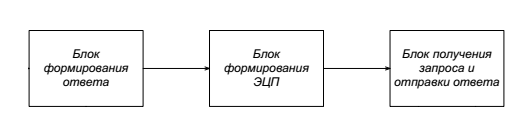


Рисунок 2.3 – Взаимосвязь блока формирования ЭЦП с другими блоками системы

*Блок получения запроса и отправки ответа* представляет собой класс, разворачиваемый на сервере. Собственно этот класс и будет являться конечной точкой веб-сервиса. Он будет содержать минимум три открытых метода, доступных для удаленного вызова. Параметрами для этих методов будут параметры входных запросов от клиентов, а возвращаемым значением ответ веб-сервиса.

*Блок проверки запроса.* В виду того, что веб-сервис предоставляет доступ к государственной базе данных, в которой сохранены конфиденциальные данные граждан Республики Беларусь, следует предусмотреть возможность предотвращения несанкционированного доступа к базе данных, а так же изменение информации, сохраненной в ней и доступ к информации, которую веб-сервис не должен предоставлять. Вычисление хеш-функции от конкатенации логина и пароля позволяет проверить их до соединения с базой данных. Данный блок предотвращает внедрение вредоносного sql-кода в запрос к базе данных. Защита предусматривает следующие принципы внедрения sql-кода:

* внедрение в строковые параметры;
* использование ключевого слова UNION;
* экранирование хвоста запроса;
* расщепление sql-запроса.

На рисунке 2.4 показан фрагмент структурной схемы, на котором отображена взаимосвязь блока проверки запроса с другими блоками системы.

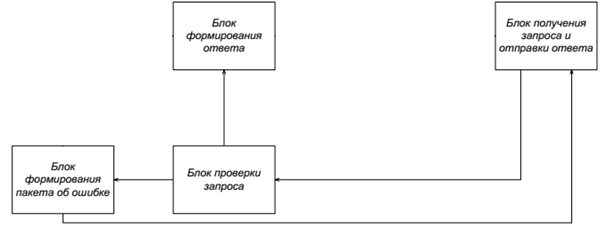


Рисунок 2.4 – Взаимосвязь блока проверки запроса с другими блоками системы

*Блок формирования пакета об ошибке*. Во время проверки параметров запроса, авторизации и формировании ответа веб-сервиса может возникнуть ряд проблем. К примеру:

* неверные логин или пароль;
* неверно заданы параметры запроса;
* найдено более одной записи в базе данных;
* обнаружен вредоносный код в параметрах запроса;
* недоступна база данных;
* и другие.

О каждой ошибке веб-сервис будет собирать пакет, сообщающий клиенту код ошибки. Расшифровка кодов будет в открытом доступе.

**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

На данном этапе рассмотрим подробней функционирование программного обеспечения. Для этого проведем анализ основных блоков, частей программы и рассмотрим, как они взаимосвязаны. Также рассмотрим реализацию всех классов, назначение их методов и полей.

Как известно, сервисы, расположенные в вебе, не имеют клиентской части. Они работают по большей части по системе запрос-ответ и представляют собой программу, разворачиваемую на сервере. Для этого можно использовать какой либо сервер приложений, к примеру glassfish, но для быстрого тестирования более удобным развернуть веб-сервис на каком то определенном пользовательском порте, чем и занимается класс, содержащий метод main, WebServicePublisher. Данный класс не относится прямо к проекту, он лишь запускает сервис.

Для разработки веб-сервиса была выбрана концепция объектно-ориентированного программирования (ООП), хоть по большей части реализация похожа на функциональную. Тем не менее все данные, функции, переменные будут упакованы в классы, поэтому далее рассмотрим подробно описания классов, содержащихся в проекте:

* 1. Класс auth

Класс представляет собой реализацию львиной доли блока авторизации, представленного на структурной схеме. Сервис предоставляет доступ к своим услугам лишь тем, кому это было разрешено, то бишь выданы логин и пароль. Соответственно данный класс занимается тем, что решает, обрабатывать запрос или нет, валидные ли логин с паролем, указанные в запросе, или нет.

В данном классе присутствуют несколько полей:

* ModeHash – принимает целочисленные значения больше нуля и указывает какой алгоритм подсчета хеша будет использоваться для проверки логина и пароля;
* ModeAllowed – принимает целочисленные значения больше нуля и указывает какой тип хранения хешей будет использоваться, в оперативной памяти, то есть в массиве данных, или постоянно в файле;
* File – строковая переменная, обозначающая имя файла, в котором хранятся валидные хеши данных авторизации;
* AllowedHashes – структура данных, простыми словами массив, в котором могут храниться хеши, при соответствующем ModeAllowed.

А так же несколько методов:

* аuth() – конструктор по умолчанию, используется в случаях, когда в конфигурационный файл не найден или указаны не все параметры, в данном методе поля инициализируются значениями по умолчанию;
* auth() – конструктор, параметрами которого выступают два целочисленных значения и одно строковое, очевидно обозначающие значения полей данного класса;
* GetAllowedHashes() – данный метод подтягивает из файла, путь к которому хранится в соответствующем поле класса, значения хешей, хранящихся там, в поле класса AllowedHashes;
* CheckFile() – метод позволяющий проверить наличие хеша, переданного в него как параметр, в файле с валидными хешами.
* CheckList() – позволяет проверить наличие хеша, переданного в данный метод как параметр, в поле класса AllowedHashes;
* permission() – основной и единственный открытый (не считая конструкторов) метод в классе, возвращает true или false в зависимости от того, есть ли разрешение у запроса на формирования на него ответа.

На рисунке 3.1 показана связь данного класса с другими классами системы.



Рисунок 3.1 – Взаимосвязь класса auth с другими классами веб сервиса

* 1. Класс hashes

Данный класс является чем то вроде точкой доступа к нескольким алгоритмам подсчета хеш суммы заданной строки. Используется для хранения алгоритмов подсчет хешей.

В нем реализованы методы:

* GetMD5() – вычисляет хеш сумму заданной строки по алгоритму MD5;
* toSHA1() – вычисляет хеш сумму заданной строки в соответствии с алгоритмом SHA 1.

Все методы данного класса являются статическими, с той целью, что бы их можно было использовать не создавая не нужный экземпляр класса caches.

На рисунке 3.2 отображена связь класса hashes с остальными классами системы.



Рисунок 3.2 – Взаимосвязь класса hashes с другими классами системы.

* 1. Класс checker

Класс, предназначенный для того, что бы проверять параметры запроса на содержание недопустимых символов, а так же различного рода вредоносных вставок, таких как sql-инъекции. Первоначально предполагалось провести анализ различного рода атак на базы данных, затем по полученным результатам выстроить какого-то рода защиту, однако, проанализировав суть атак типа sql-инъекция, мы пришли к выводу, что достаточным будет проверки параметров на содержание лишь русских букв (исключая пробелы), а в случае даты рождения на соответствие определенному формату.

Таким образом класс содержит лишь пару статических методов, к которым имеется доступ без создания экземпляра класса:

* check() – проверяет не содержит ли строка, переданная параметром методу, символов отличных от букв русского алфавита;
* checkBirthDate() – проверяет соответствует ли строка переданная как параметр формату «xx.xx.xxxx», где «x» обозначает произвольную цифру.

Взаимосвязь данного класса с другими классами системы показана на рисунке 3.3.



Рисунок 3.3 – Взаимосвязь класса checker с другими классами веб сервиса

* 1. Класс WebServiceInt

Класс представляет собой интерфейс главного класса веб сервиса. В описании его методов задаются soap параметры, такие как style и name. Первый используется для того что бы выбрать схему wsdl описания, второй для того, что бы сделать wsdl описание более понятным для человека, и приятным для формирования на его основе интерфейса на стороне клиента. К сожалению, возможности выделить полное имя человека в отдельный класс не было, так как под каждое поле пишется определенный веб параметр.

Класс объявляет методы:

* getFirst() – веб метод, возвращающий экземпляр класса BriefInformation, который содержит в себе краткую информацию о человеке;
* getSecond() – веб метод, возвращающий экземпляр класса FullInformaion, который содержит в себе развернутую информацию о человеке, а именно о получении им определенной социальной выплаты, о ее размере по месяцам;
* getThird() – веб метод, возвращающий экземпляр класса PensionInformation, который содержит информацию о получаемой человеком пенсии и ее размере.

То как связан данный интерфейс с другими классами, а именно какие классы реализуют данный интерфейс, можно увидеть на рисунке 3.4.



Рисунок 3.4 –Взаимосвязь интерфейса WebServiceInt со всеми его реализациями

* 1. Класс WebServiceImpl

Основной класс дипломного проекта. Наследуется от WebServiceInt. Является именно тем классом, который разворачивается на сервере и является конечной точкой сервиса.

Имеет несколько полей:

* db – экземпляр класса DB, описание которого представлено ниже, используется для доступа к базе данных;
* Authorization – экземпляр класса auth, описание которого представлено выше, используется для проведения авторизации, а именно проверки того разрешен ли доступ к сервису по указанным в запросе логину и паролю;

А так же реализованы методы, которые являются, или вернее, предоставляют услуги данного веб сервиса:

* getFirst() – метод, который является реализацией одноименного метода, заявленного в интерфейсе, его основной задачей является собирание ответа на запрос о краткой информации о человеке;
* getSecond() – так же реализация метода объявленного в интерфейсе, его основной задачей является собирание ответа на запрос о полной информации о человеке;
* getThird() – третий основной метод класса, возвращает ответ на запрос о получаемой пенсии определенного человека;
* WebServiceImpl() – конструктор данного класса, по сути являющийся блоков начальной конфигурации, представленного на структурной схеме, в данном методе читается файл конфигурации, определяются поля класса.

Таким образом, из тела входящего soap сообщения достаются значения, которые являются параметрами к одному из трех веб методов данного класса. Данный класс является отправной точкой веб сервиса. Так как разрабатываемый сервис является частью большого комплекса сервисов от нашего предприятия, он должен быть легко расширяем и внедряем в другие модули других сервисов. Поэтому хоть и запускается данный сервис из стороннего класса, имеющего метод main, он должен быть способным развернуться на каком либо сервере по типу glassfish, tomcat, а это в свою очередь означает, что первоначальную настройку не имеет смысла проводить в внешнем классе, который занимается запуском сервиса. Было принято решение, что первоначальную настройку (блок начальной конфигурации на структурной схеме) лучше всего провести в конструкторе главного класса WebServiceImpl, так как когда сервис запускается первым, что выполняется является конструктор данного класса, а после завершения его работы, сервис будет настроен и ожидает soap сообщение от клиентов.

Для такой первоначальной настройки используется стандартный пакет java.util.Properties, позволяющий легко обрабатывать файлы, содержащий в себе наборы формата «ключ = значения». Обычно для таких настроек используют файлы с расширением «.ini». Соответственно было решено не создавать велосипед, а просто использовать в данном проекте файл с именем «config.ini», который и будет содержать в себе пары выше представленного формата. В нем задаются следующие параметры:

* host – строковое поле, имя базы данных, с которой будет производиться соединение;
* HashesFile – строка, которая означает имя файла, в котором будут хранится валидные хеши;
* ModeHash – целочисленное поле, указывает какой алгоритм хеширования будет применяться;
* ModeAllowed – целочисленное поле, указывает какой тип хранения валидный хешей будет использоваться;
* logfile – означает имя файла, который называют лог файлом, в который пишется информация о состоянии сервиса (сервера) наподобие черного ящика;

Так же в нем задаются следующие параметры, но они используются в классе HandlerValidator:

* KeyAlg – означает имя алгорится, для которого будут или были сгенерированы ключи;
* provName – означает наименование крипто провайдера, не обязательно к указанию, так как опционально;
* KeyStorageFile – важное поле, отвечает за имя файла, являющегося хранилищем ключей и расширение которого должно быть .jks;
* storageKey – так как всякое хранилище находится под защитой пароля, нужно поле отвечающее за это пароль, собственно это и есть такое поле;
* aliasKey – все ключи содержатся в хранилище под псевдонимами, доступ к которым осуществляется по паролю, данное поле задает пароль к плевдониму;
* KeyLength – означает длину генерируемых для теста, или просто так, ключей;
* signAlg – означает алгоритм, по которому генерируется цифровая подпись;
* aliasName – значение наименования псевдонима, под которым содержатся в хранилище ключей нужные нам ключи.

По сути, все методы выполняют однотипную работу, различаются лишь тем, какую структуру данных заполняют и какие методы вытягивают из класса getmesomething. Результатом их работы является заполненный экземпляр соответствующего класса.

Однако, так как перед тем как начать собирать ответ, нужно выполнить различного рода проверки входящего soap сообщения, а так же обработать все ошибки, которые возникают по мере составления ответа. Если возникают какие то проблемы, будь то неверные параметры от клиента, ошибки во время работы с базой данных, не определенные указатели, какие то другие ошибки на сервисе, в соответствии со стандартом soap, сервис должен послать клиенту сообщение об ошибке с тегом Fault, в котором будут дочерние элементы faultcode, содержащий код ошибки, и faultstring, содержащий словесное описание ошибки для человека. В параметрах данного класса указан @HandlerChain(), являющийся еще одним обработчиком soap сообщений, которые будут приходить на обработку, и уходить после нее. В скобках указывается xml файл, стандартного для такого обработчика формата, содержащий в теле <handler-class>тут указывается класс обработчик </handler-class> , обработчиком будет являться класс HandlerValidator, описание которого будет ниже.

А пока что, предлагаю посмотреть как связан данный центральный класс сервиса с остальными классами, что представлено на рисунке 3.5.



Рисунок 3.5 – Взаимосвязь класса WebServiceImpl с остальными классами системы

* 1. Класс HandlerValidator

Данный класс, являясь еще одним обработчиком soap сообщения, наследуется от SOAPHandler<SOAPMessageContext>, что позволяет ему обрабатывать как входящие, так и исходящие, после обработки классом WebServiceImpl, сообщения. Он нужен для того, что бы произвести проверку сообщения, собрать пакет об ошибке, а так же подписать сообщение с помощью электронной цифровой подписи.

В данном классе присутствуют следующие поля:

* MS – экземпляр класса MesageSignature, описание которого будет ниже, и нужный для создания электронной цифровой подписи на основе ключей, лежащих в хранилище ключей, которое представляет собой зашифрованный файл с расширением .jks;
* filein – поле типа File, используется для создания потока чтения, который в котором будет тело soap сообщения, которое нужно подписать;
* fileout - поле типа File, используется для создания потока вывода, в который пойдет созданная электронная цифровая подпись;
* KeyAlg – поле тип String, хранящее название алгоритма, для которого генерировались ключи для электронной цифровой подписи;
* KeyLength – поле типа int, которое означает длину закрытого и открытого ключей;
* Logfile – поле строкового типа, хранит имя файла с логами;
* provName – поле строкового типа, хранящее название крипто провайдера, к примеру SUN и другие, если нет необходимости в явном указании, то данное поле может быть пустым;
* signalg – поле типа String, хранящее имя алгоритма, который используется для создания электронной цифровой подписи;
* KeyStorge – поле типа int, являющееся подобием флага, показывает есть ли в наличии хранилище ключей или создавать пару на ходу, что подходит для целей тестирования сервиса;
* KeyStorageFile – поле строкового типа, содержит путь к хранилищу ключей типа .jks (Java Key Store);
* storageKey – строковое поле, хранящее пароль доступа к хранилищу ключей;
* aliasKey – строковое поле, хранящее пароль доступа к паре ключей, хранящихся под определенным псевдонимом;
* aliasName – строковое поле, хранящее псевдоним, под которым в хранилище ключей хранится нужная нам пара ключей;
* log – экземпляр класса Log, который занимается логированием состояния сервиса.

В нем реализованы следующие методы:

* HandlerValidator – конструктор данного класса, предназначен для задания первоначальных параметров, так же использует config.ini для считывания нужных ему параметров, таких как длина ключей электронной цифровой подписи, пути к открытому и закрытому ключам электронной цифровой подписи, имя алгоритма подписи, имя лог-файла;
* handleMessage – основной метод данного класса, в нем происходит запись информации о состоянии сервиса в лог файл, проверка количества параметров входного запроса, подписание исходящего сообщения с помощью электронной цифровой подписи;
* getHeaders() – метод, возвращающий ничего, нужный лишь потому, что нужно определить, хотя по идее возвращающий набор ссылок на теги из заголовка сообщения;
* handleFault() – метод, обрабатывающий ошибки сервера, и он тоже не используется;
* close() – метод выполняющийся в конце обработки soap сообщения, так же нужный лишь потому что объявлен в базовом классе;
* getFaultString() – метод, возвращающий словесное описание ошибки по ее коду;
* generateSOAPErrMessage() – метод, который изменяет soap сообщение, путем добавления в него тегов означающих ошибку, для того что бы клиент узнал какого типа ошибка возникла, и что он сделал не так;
* DefaultFields() – метод, который занимается установлением полей данного класса в начальный значения, то бишь в значения по дефолту;
* ConfigFields() – метод, нужный для того, что бы прочитать файл конфигурации (config.ini), и установить поля данного класса в соответствии с тем, что указано в конфигурационном файле;
* ConfigEDS() – метод, который выполняет создает экземпляр классаMessageSignature, определяет поле класса MS, в зависимости от того, что было указано в конфигурационном файле, заполняет поля экземпляра класса MessageSignature MS;
* StringToChar() – метод, который принимает параметром строку типа String, конвертирует ее в массив типа char[] и возвращает полученный массив в качестве результата;
* EDS() – метод, принимающий параметром soap сообщение, полученное в результате обработки клиентского запроса и являющееся ответом на него, и на основе контекста тела данного soap сообщения, рассчитывает электронную цифровую подпись, а затем изменяет структуру сообщения, добавляя в заголовок несколько дочерних элементов, в контекстах которых будет передана цифровая подпись;
* CheckTroubles() – метод, параметром которого является исходящее soap сообщение, проверяет не возникло ли ошибки в составлении ответа на запрос клиента, и если возникла, то вызывает метод по составлению и отправки пакета об ошибке;
* CheckGetFirstFields() – метод, который выполняет проверку входного soap сообщения от клиента на соответствие полей заданному формату, а так же на количество передаваемых параметров;
* CheckGetSecondFields() – метод, который выполняет проверку входного soap сообщения от клиента на соответствие полей заданному формату, а так же на количество передаваемых параметров, только для запроса о полной информации.

Таким образом, данный класс можно назвать вторым центральным классом системы, так как он так же занимается управлением обработкой входящего soap сообщения, только на другом уровне. Если WebServiceImpl отвечает за собирание составление ответа, то данный класс отвечает за проверку входящего soap сообщения на всякого рода угрозы и неточности со стороны клиента, за составления пакета об ошибке, если что то во время обработки запроса пошло не так, а так же, что не мало важно, за подписание исходящего soap сообщения с помощью электронной цифровой подписи, которая создается на основе тех ключей, которые уже имеются у нас в хранилище ключей.

На рисунке 3.6 изображена часть диаграммы классов, которая показывает, как связан данный класс и другие классы сервиса.



Рисунок 3.6 – Взаимосвязь класса HandlerValidator с остальными классами системы

* 1. Класс getmesomething

Данный класс является частью блока работы с базой данных. Он содержит статические методы, в которых захардкодены sql-запросы к базе данных, и которые возвращают нужные для работы сервиса данные из ГИССЗ.

В данном классе реализованы следующие методы:

* PassportID() – метод, возвращающий номер паспорта, по имени, фамилии, отчеству (ФИО) и дате рождения человека;
* Address() – метод, возвращающий адрес (город, улица, дом, квартира) по ФИО и дате рождения лица;
* SocialPayment() – возвращает наименование социальной выплаты, по ФИО и дате рождения человека;
* FullPayment() – возвращает строковую таблицу, которая содержит наименования «месяц, год» и численные значения выплат, полученных человеком за данный период в определенные месяц, год;
* getAidByCid() – функция, которая возвращает параметр Aid по номеру дела Cid, нужная для работы метода FullPayment();
* getCidByPid() – метод, который возвращает номер заведенного дела Cid по указанному персональному идентификатору человека Pid;
* getMonthByNum() – возвращает название месяца по его указанному порядковому номеру от 1 до 12 соответственно;
* getPidByFullName() – метод, возвращающий уникальный идентификатор человека pid, по заданным ФИО и дню рождения;
* PensionPayment() – метод, параметрами которого являются имя, фамилия, отчество и дата рождения человека, и который предоставляет данные о получаемой им пенсии;
* FullName() – метод, который по ФИО и дате рождения полученным из параметров входящего soap сообщения, ищет в ГИССЗ наличие данного человека.

А связь данного класса с остальными компонентами сервиса можно увидеть на рисунке 3.7.



Рисунок 3.7 – Взаимосвязь класса getmesomething с остальными компонентами сервиса

* 1. Класс DB

Данный класс представляет собой реализация блока работы с базой данных, выделенного в результате системного. Класс предназначен, как бы это странно не звучало, для работы с базой данных, но не со всякой, а с oracle database, так как для ГИССЗ используется именно эта база данных. Функционал класса представлен его методами:

* DB() – конструктор, параметром которого является строковая величина, хранящая Oracle System Identifier (SID) базы данных, к которой будет выполняться подключение;
* disconnect() – метод, завершающий использование соединения с базой данных, обеспечивается разъединение;
* connectdatabase() – метод, параметрами которого являются две строковые величины логин и пароль соответственно, выполняющий соединение с базой данных ГИССЗ и возвращающий ID соединения;
* SimpleSelect() – метод, параметром которого является строка, хранящая sql запрос, позволяющий выполнить произвольный sql запрос к бд и возвращающий полученную в результате выполнения запроса таблицу;
* ifconnect() – метод, возвращающий true или false в зависимости от того установлено ли соединение с базой данных ГИССЗ.

То, отношения данного класса с другими показано на рисунке 3.8.



Рисунок 3.8 – Связь класса DB с другими классами системы

* 1. Класс BriefInformation

Класс, являющийся структурой данных, поля которой заполняются методом getFirst() главного класса, и экземпляр которого отправляется клиенту в качестве ответа на запрос о краткой информации о человеке.

Содержит следующие поля:

* FirstName – строковая величина, хранящая имя найденного в ГИССЗ человека или код ошибки, возникшей в результате процесса обработки данных пришедших от клиента;
* MiddleName – строковая величина, хранящая отчество того же человека или словесное описание ошибки, возникшей в результате процесса обработки данных пришедших от клиента;
* LastName – соответственно фамилия его же;
* BirthDate – поле типа String, означающее день рождения человека;
* PassportID – поле типа String, обозначающее номер паспорта человека;
* Address – строковая величина, обозначающая адрес;
* Date – поле типа String, хранящее в себе дату предоставления информации;
* SocialPayment – наименование социальной выплаты которую получает искомое лицо;
* Now – текущее время сервера.

Имеет три конструктора:

* по умолчанию – нужен лишь для инициализации экземпляра;
* с тремя строковыми параметрами – для быстрого заполнения ФИО;
* с двумя строковыми параметрами – для быстрого заполнения структуры с информацией об ошибке.

Хотя в классе и реализовано несколько конструкторов, на деле используется лишь по умолчанию, так как все поля должны быть открытыми, чтобы корректно возвращался результат работы сервиса, то есть корректно собиралось soap сообщение и строилось wsdl описание сервиса, а так же с двумя строковыми параметрами. Так как хотелось бы скрыть от посторонних глаз поля в которых указано что конкретно пошло не так на сервере, а так же названия этих полей, было решено использовать первые два поля в каждом из классов для ответов.

В FirstName будет содержаться код ошибки, а в MiddleName ее дамп. Можно было бы конечно создать отдельные поля и пометить их как private, но это дополнительная морока и лишняя потребленная память. Хоть и копейки, но копеечка рубль бережет.

Класс BriefInformation связан лишь с одни классом, тем не менее эта связь показана на рисунке 3.9.



Рисунок 3.9 – Связь данного класса с остальными классами сервиса

* 1. Класс PensionInformation

Класс, являющийся структурой данных, поля которой заполняются методом getThird() главного класса, и экземпляр которого отправляется клиенту в качестве ответа на запрос о пенсионной информации о человеке.

Содержит следующие поля:

* FirstName – строковая величина, хранящая имя найденного в ГИССЗ человека или код ошибки, возникшей в результате процесса обработки данных пришедших от клиента;
* MiddleName – строковая величина, хранящая отчество того же человека или словесное описание ошибки, возникшей в результате процесса обработки данных пришедших от клиента;
* LastName – соответственно фамилия его же;
* BirthDate – поле типа String, хранящее дату рождения человека;
* PassportID – поле типа String, означающее номер паспорта человека, найденного в ГИССЗ;
* Address – строковая величина, обозначающая адрес;
* Date – поле типа String, хранящее в себе дату предоставления информации;
* PensionPayment – наименование пенсии, которую получает искомое лицо;
* Now – текущее время сервера.

Имеет три конструктора:

* по умолчанию – нужен лишь для инициализации экземпляра;
* с тремя строковыми параметрами – для быстрого заполнения ФИО;
* с двумя строковыми параметрами – для быстрого заполнения структуры с информацией об ошибке.

Этот класс, как и предыдущий, играет роль структуры данных, отправляемой клиенту как ответ на запрос, в данном случае, об информации о предоставлении пенсии указанному в запросе лицу. Тут так же имеется не очевидное решение использовать первые два поля как хранилище описания возникающих в процессе собирания ответа ошибок.

Связь данного класса и других классов системы приведена на рисунке 3.10.



Рисунок 3.10 – Связь данного класса и других классов системы

* 1. Класс FullInformation

Класс, являющийся структурой данных, поля которой заполняются методом getSecond() главного класса, и экземпляр которого отправляется клиенту в качестве ответа на запрос о полной информации о человеке за период.

Содержит следующие поля:

* FirstName – строковая величина, хранящая имя найденного в ГИССЗ человека;
* MiddleName – строковая величина, хранящая отчество того же человека;
* LastName – соответственно фамилия его же;
* PassportID – номер паспорта;
* Address – строковая величина, обозначающая адрес;
* DateBegin – дата начала периода;
* DateEnd – дата окончания периода;
* SocialPayment – наименование социальной выплаты которую получает искомое лицо;
* AmountPayment – размер получаемой выплаты;
* Table – структура данных по типо двумерной таблицы, хранящая размер выплаты помесячно в течении указанного периода;
* Now – текущее время сервера.

Имеет три конструктора:

* по умолчанию – нужен лишь для инициализации экземпляра;
* с тремя строковыми параметрами – для быстрого заполнения ФИО;
* с двумя строковыми параметрами – для быстрого заполнения структуры с информацией об ошибке.

А связь представленного выше класса с другими классами системы представлена на рисунке 3.11.



Рисунок 3.11 – Связь класса FullInformation с остальными классами системы

* 1. Класс MessageSignature

Класс, содержащий в себе поля и методы нужные для создания электронных цифровых подписей как для действительного подписания файла любого формата, так и для создания хранилища ключей, генерирование запись в него пары ключей для заданного алгоритма в рантайме для тестирования сервиса. Рассмотрим данный класс подробней, для того что бы понять как он функционирует.

Класс MessageSignature содержит шесть закрытых (с модификатором доступа private) полей:

* keyPairGenerator – экземпляр класса KeyPairGenerator, нужный для генерации ключевых пар;
* keyPair – поле типа KeyPair, хранящее в себе пару ключей, а именно закрытый и открытый;
* privateKey – поле типа PrivateKey, хранящее в себе закрытый ключ, используемый для расчета цифровой подписи;
* publicKey – поле типа PublicKey, хранящее в себе открытый ключ, используемы для расчета цифровой подписи;
* signature – поле типа Signature, являющееся цифровой подписью;
* realSign – поле типа byte[], то бишь массив байтов, хранящий в себе значение цифровой подписи в виде потока байт.

Так же при реализации данного класса были созданы методы:

* MessageSignature() – конструктор данного класса, используется тогда имеется пара ключей, и они находятся в сертификате, и в хранилище ключей, параметрами данного метода являются алгоритм ключевой подписи и название крипто провайдера;
* MessageSignature() – конструктор данного класса, используется тогда когда нет пары ключей, данный конструктор сгенерирует их автоматически на основании параметров метода и сохраняет полученные ключи в поля класса; параметрами метода являются название алгоритма, для которого генерируется пара ключей, длина ключей, алгоритм электронной цифровой подписи и название крипто провайдера, который можно оставить неопределенным;
* signingMessage() – метод класса, который создает цифровую подпись из указанного открытого текста, параметрами метода являются поток ввода с открытым текстом, в нашем случае это передаваемое soap сообщение, и поток вывода с созданной цифровой подписью;
* verifyMessage() – метод, который проверяет действительность цифровой подписи, параметрами метода являются поток ввода с открытым текстом и поток ввода с цифровой подписью, возвращает результат проверки цифровой подписи, тип возвращаемого значения boolean;
* getSign() – метод, возвращающий цифровую подпись как массив байтов, нужен для того, что бы по этому массиву байт построить строку, которую и передаем клиенту в целях экономии трафика;
* savePrivateKey() – метод, сохраняющий закрытый ключ, имеет лишь один параметр поток вывода, в который и будет писаться ключ;
* savePublicKey() – метод, сохраняющий открытый ключ, так же имеет лишь один параметр поток вывода, куда и будет писаться ключ;
* readPrivateKey() – метод, параметром которого является поток ввода, считывает из заданного потока ввода закрытый ключ;
* readPublicKey() – метод, параметром которого является поток ввода, считывает из заданного потока открытый ключ;
* getPair() – метод, который возвращает пару ключей типа KeyPair из хранилища ключей с расширением .jks, параметрами метода являются поток ввода, где находится хранилище ключей, название сертификата открытого ключа, пароль для хранилища ключе и пароль для сертификата;
* setPrivateKey() – устанавливает закрытый ключ, который передается в параметре метода с типом PrivateKey;
* setPublicKey() – устанавливает открытый ключ, который передается в параметре с типом PublicKey;
* getPrivateKey() – возвращает закрытый ключ ,который хранится в поле класса privateKey;
* getPublicKey() – возвращает открытый ключ, который хранится в поле класса publicKey.

То как используется этот класс видно на диаграмме классов, фрагмент которой представлен на рисунке 3.12, показывающий взаимодействие данного класса и других классов сервиса.



Рисунок 3.12 – Взаимосвязь класса MessageSignature с другими классами сервиса

Таким образом мы рассмотрели все классы, включающие в себя поля и методы, которые реализованы в классах, разрабатываемого веб сервиса, указали каким образом они будут взаимодействовать друг с другом.

Полная диаграмма классов приведена на чертеже ГУИР.400201.010 РР.1.

Теперь же рассмотри то, каким образом функционирует данный веб сервис, с чего начинается работа, где и в какой последовательности вызываются те или иные методы, словом то, по результатам чего строится диаграмма последовательности.

* 1. Начало работы

Как было однократно написано выше, у нас имеется класс WebServiceImpl, который помечен как @WebService. Это говорит java, что данный класс является конечной точкой веб сервиса. Он разворачивается на сервере и ждет, пока не придет soap сообщение от клиента, с заданными полями. Если клиент попытается отправить неверно составленное soap сообщение (имеется в виду не правильно с точки зрения xml), то сервис автоматически отправит сообщение клиенту о том, что не удается прочитать параметры из того, что он прислал. Тоже самое касается и случая с неверными по типу параметрами: невозможно прислать строку вместо параметра с целочисленным типом. Это происходит из-за того, что использование веб сервиса напоминает вызов удаленной процедуры ( что этим и является так то). То есть если у нас есть @WebMethod с параметрами String и int, то и вызывается он с параметрами String и int и никак иначе. Так же, прибывающее soap сообщение проходит обработку в HandlerValidate, в котором выполняется проверка на превышение количества параметров сообщения, содержание вредоносных или заведомо безрезультатных параметров.

* 1. Первоначальная настройка

После того как сервис запущен, создаются экземпляры двух классов: HandlerValidate и WebServiceImpl – соответственно выполняется код, который написан в их конструкторах. Возможно, файл конфигурации не будет найден, а значит сервис будет работать в режиме по умолчанию. Его начальная настройка описывается значениями полей, находящихся в этих класса:

* host – примет значение «tri»;
* HashesFile – примет значение «HashesMD5.txt»;
* ModeHash – целочисленное поле, значение будет ноль;
* ModeAllowed – целочисленное поле, значение по умолчанию которого рано нулю;
* logfile – логи будут писаться в файл с именем «log.log»;
* KeyAlg – алгоритм по умолчанию для электронной цифровой подписи выбран DSA;
* provName – в силу того, что используется java, имя крипто провайдера по умолчанию выбрано «SUN»;
* KeyStorageFile – имя создаваемого для тестирования сервиса хранилища ключей будет иметь имя «server.jks»;
* storageKey – пароль для доступа к хранилищу ключей «serverpass»;
* aliasKey – пароль для доступа к паре ключей в хранилище ключей имеет значение по умолчанию «serverpass»;
* KeyLength – длина генерируемых ключей будет равна 2048;
* signAlg – так как алгоритм для которого генерировались ключи выбран DSA, соответственно и подпись будет DSA;
* aliasName – значение наименования псевдонима, под которым содержатся в хранилище ключей нужные нам ключи будет «server-pass».

Далее, считывается файл «config.ini» и заполняется большинство (или все) поля названные выше. Если что то вдруг пошло не так, и каких-то значений , необходимых для корректной работы сервиса, нет, то он начинает работу в режиме по умолчанию, то есть все поля устанавливаются в значения указанные выше.

То в какой последовательности выполняется настройка для класса HandlerValidate, и экземпляры каких классов создаются на данном этапе представлено на фрагменте диаграммы последовательности, расположенному на рисунке 3.13



Рисунок 3.13 – Начальная настройка HandlerValidate

То в какой последовательности настройка выполняется для класса WebServiceImpl представлено на на рисунке 3.14



Рисунок 3.14 – Начальная настройка WebServiceImpl

* 1. Проверка сообщения

После завершения стадии начальной настройки сервис ожидает запроса от клиента. Когда приходит soap сообщение от клиента выполняется проверка его параметров, так как перед тем как будет выполняться нужная клиенту функция с нужными ему параметрами, приходящее, опять таки, от него soap сообщение проходит проверку на:

* Содержание посторонних символов;
* Содержание вредоносных вставок;
* Соответствие дат формату «xx.xx.xxxx», где «x» означает произвольную цифру;
* Содержанию ФИО только русских букв обоих регистров.

Данный вызов представлен на рисунке 3.15.



Рисунок 3.15 – Вызов проверки сообщения

* 1. Создание ответа

После того как сообщение было проверено, и checker постановил, что сообщение не содержит ничего лишнего и опасного, вызывается один из нескольких методов класса WebServiceImpl, в зависимости от того, какая информация нужна клиента. Параметры для вызова метода берутся из полей тела soap сообщения принятого от клиента. Для того, что бы получить данные из ГИССЗ нужно установить соединение с ней (базой данных). Для этого проверятся есть ли у того, за кого выдает себя клиент посредством данных авторизации, доступ к получению информации. Это выполняется следующим образом:

* Создается строка, представляющая собой конкатенация строк-значений параметров метода login и password;
* По одному из алгоритмов кеш суммы считается кеш сумма данной строки;
* Проверяется список валидных кешей, взятый из файла с кешами;
* Если данная кеш сумма присутствует в списке, то клиенту разрешается получить информация, иначе выбрасывается исключение;

Далее выполняется соединение с базой данных по заявленным данным авторизации. Затем пытаемся найти человека, информация по которому нужна клиенту. Если его нет, то выбрасывается исключение, если есть, то создается экземпляр класса, который будет возвращен данным методом в качестве результата. После этого с помощью класса getmesmething, вытягиваются нужные для заполнения полей объекта-результата данные.

Последовательность вызовов представлена на рисунке 3.16.



Рисунок 3.16 – Последовательность создания ответа

* 1. Проверка на возникновение ошибки

Различного рода ошибки могут появиться во время обработки пришедших от клиента данных, а именно:

* Проверки на доступ к ГИССЗ;
* Соединение с ГИССЗ;
* Создания экземпляра класса для формирования ответа;
* Получения данных из государственно базы данных.

Для их перехвата используются стандартные действия, которые реализованы в java под именем блок try-catch. Он позволяет перехватывать все ошибки, возникающие во время выполнения кода, помещенного в блок try, и обрабатывать их в блоке catch. Если возникает ошибка, выбрасывается исключение, в блоке catch в зависимости от того какого рода возникла ошибка и где заполняются поля ответа особым образом:

* В поле FirstName помешается строка со значением «trouble», означающий, что возникла ошибка (беда);
* В поле MiddleName помещается код ошибки.

В главном методе класса HandleValidate происходит проверка поля FirstName на наличие строки, сигнализирующей об возникновении во время обработки ошибки. Если она есть, то исходящее сообщение модифицируется следующим образом:

* Все тело сообщения очищается;
* На основе кода ошибки генерируется ее словесное описание;
* В тело сообщения помещается тег faultcode, с содержанием кода возникшей ошибки;
* Так же в тело помещается еще один дочерний тег faultstring, содержащий сгенерированное словесное описание данной ошибки.

Таким образом, разрабатываемый сервис достаточно удобен для использования, так как если клиент что-то сделал не так, он об этом узнает.

Список ошибок и их кодов, отправляемых клиенту представлен ниже:

* Код «1» - превышено число параметров в запросе;
* Код «2» - параметры неверного формата;
* Код «3» - авторизация не удалась;
* Код «4» - не валидные данные авторизации, отказано в доступе;
* Код «5» - не найдено ни одной записи в базе;
* Код «6» - не хватает параметров в запросе;
* Код «7» - ошибка в построении soap сообщения;
* Код «8» - ошибка сервера, ответа не будет.
  1. Отправка клиенту

После того как был составлен ответ, и во время авторизации и формирования ответа не возникло ошибок, сгенерированный ответ, являющийся так же как и запрос soap сообщением нудно отправить клиента. Однако, так как отправляются данные из государственно базы данных, то есть данные достаточно важны как нашим клиентам, так и, простите за тавтологию, клиентам наших клиентов. Поэтому весь контекст тела ответного soap сообщения подписывается с помощью электронной цифровой подписи. Для этого создаем несколько полей в заголовке ответа, где указываем значение подписи и подписанный сертификат открытого ключа.

В сервисе предусмотрен вариант тестирования, при котором на ходу возможно сгенерировать хранилище ключей, сами ключи, а так же получить самоподписанный сертификат открытого ключа, однако в реальной жизни никто никому не доверяет, но все доверяют удостоверяющему центру. Поэтому предполагается, что в наличии уже есть хранилище ключей формата java keystore (.jks), в котором уже есть закрытый и открытый ключи (ну или не одна пара, это не важно, так как все пары хранятся под своими псевдонимами, и просто вытянем из хранилища нужную пару ключей), а так же подписанный доверенным центром сертификат открытого ключа.

Далее все просто:

* Записываем в поток ввода контекст тела ответного soap сообщения;
* Вызываем метод signingMessage из класса отвечающего за формирование эцп;
* Из указанного при создания эцп потока вывода считываем подпись;
* Модифицируем ответное soap сообщение, включая в него информацию об электронной подписи, то есть подписывает таки сообщение.

**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Главный класс и его интерфейс

При разработке программного обеспечения известного как Web-сервис взаимодействия ГИССЗ с АСУ «Занятость» началось с интерфейса для главного класса WebServiceInt, подключением библиотек для soap и указанием того, что данный интерфейс будет работать как веб-сервис:

import javax.jws.WebMethod;

import javax.jws.WebParam;

import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

@WebService

@SOAPBinding(style = SOAPBinding.Style.RPC)

public interface WebServiceInt {…}

Далее создавались методы, и перед каждым ставилась аннотация, что метод можно будет вызывать из веба:

@WebMethod

public BriefInformation getFirst;

@WebMethod

public PensionInformation getThird;

@WebMethod

public FullInformation getSecond;

А так же все методы должны иметь модификаторы доступа public для того, что бы метод был открытым и мог вызываться извне. Java настолько прекрасна, что сама сгенерирует wsdl описание нашего веб сервиса, по которому клиенты смогут построить класс, с помощью которого будут вызывать наши веб методы. Однако генерирует она конечно очень топорно и безразлично. Поэтому, что бы сделать описание, а значит и клиентский сгенерированный класс, более дружелюбным пользователем было принято решение изменить имена веб параметров с arg0 или arg2 на что то более понятное человеку. Таким образом полное объявление метода getFirst() приняло вид:

public BriefInformation getFirst(

@WebParam(name = "Login")String login,

@WebParam(name = "Password")String password,

@WebParam(name = "Name")String FirstName,

@WebParam(name = "Patname")String MiddleName,

@WebParam(name = "Surname")String LastName,

@WebParam(name = "BirthDate")String BirthDate);

Остальные методы описаны аналогичным образом, приводить их объявление не видится необходимым.

Далее началось написание основного класса WebServiceImpl. Примечательно, что перед ним стоит аннотация о конечной точке.

@WebService(endpointInterface = "pack.WebServiceInt")

public class WebServiceImpl implements WebServiceInt

Так же возникла проблема, по которой клиент мог в soap сообщении создать несколько полей с одинаковыми тегами (случайно или специально, это не важно), в таком случае сервис бы нормально обработал сообщение, только используя параметры из последних повторяющихся тегов. То есть если в xml:

<Name>никита</Name>

<Name>афоня</Name>

То в параметр Name метода сервиса будет записано значения из второго тега <Name>, а первое проигнорировано. Таким образом, появилась потребность в предобработчике для входящих сообщений. Решение появилось так же с помощью стандартных средств java. Используется аннотация к главному классу:

@HandlerChain(file = "handler-chain.xml")

4.2 Еще один обработчик сообщения

Далее создается xml файл, указанный в ней, при чем путь указывается относительно папки, в которой лежит исходник главного класса, а не полный путь. Содержание файла такое:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<handler-chains xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/javaee\_web\_services\_metadata\_handler\_2\_0.xsd">

<handler-chain>

<handler>

<handler-class>pack.HandlerValidator</handler-class>

</handler>

</handler-chain>

</handler-chains>

В теге <handler-class> указывает имя класса обработчика в формате «пакет.класс». Таких обработчиков может быть несколько, в таком случае они указывается в таких же тегах в <handler>. Данный обработчик должен реализовывать определенный интерфейс интерфейс SOAPHandler<SOAPMessageContext>, что собственно он и делает. Так же он должен реализовать все методы заявленные в интерфейсе.

@Override

public boolean handleMessage(SOAPMessageContext context)

Метод handleMessage принимающий параметром контекст soap сообщения и является собственно тем самым предобработчиком. Однако, он используется не только как предобработчик, но и как постобработчик ответа. Данный метод выполняется всякий раз когда через порт, на котором развернули сервис проходит soap сообщение, то есть не только от клиента, но и от сервиса к клиенту. Таким образом код данного метода разделен на две части:

* Обработка сообщения от клиента;
* Обработка ответа клиенту;

Boolean isRequest = (Boolean) context.get(MessageContext.MESSAGE\_OUTBOUND\_PROPERTY);

if(!isRequest){…}

else{…}

Выше кусок кода, который занимается определением того какой сообщение поймалось, и в зависимости от этого проверять его или же проверять на ошибки или подписывать.

Проверка сообщения от клиента выполняется посредством подсчета числа дочерних тегов для тела входящего сообщения:

FN = body.getElementsByTagName("Name");

MN = body.getElementsByTagName("Patname");

LN = body.getElementsByTagName("Surname");

BD = body.getElementsByTagName("BirthDate");

login = body.getElementsByTagName("Login");

pass = body.getElementsByTagName("Password");

int length = FN.getLength() + MN.getLength() + BD.getLength() + LN.getLength() + login.getLength() + pass.getLength();

Если какого то тега не окажется в сообщении, то строка с определением количества выбросит исключение об обращении по нулевому указателю. Такая же проверка выполняется и для метода с временными рамками, суть вроде бы должна быть ясна. Далее не поймав исключительной ситуации забираем из этих переменных значения их первых элементов:

FN.item(0).getTextContent()

И отправляем их на соответствие заданному формату с помощью класса checker.

Данный класс представляет собой несколько методов, отличающихся по большей части количеством передаваемых параметров. Для проверок используется java.util.regex.\*, то есть пакет java для работы с регулярными выражениями. Далее создаем пару экземпляров нудных классов и используем определенные методы. Полагается, что лучше в такое вникать используя примеры:

Pattern p = Pattern.compile("^[а-яА-Я]+$");

Matcher m = p.matcher(str);

return m.matches();

Для проверки имени фамилии и отчества. И для дат:

Pattern.compile("^\\d\\d\\.\\d\\d\\.\\d\\d\\d\\d$")

Если вдруг проверка сообщение не прошло, то вызывается метод generateSOAPErrMessage, к примеру с кодом 1:

generateSOAPErrMessage(soapMsg, "1");

Он генерирует soap сообщение об ошибке.

SOAPBody sB = msg.getSOAPPart().getEnvelope(). getBody();

SOAPFault soapFault = soapBody.addFault();

soapFault.setFaultString(getFaultString(code));

soapFault.setFaultCode(code);

throw new SOAPFaultException(soapFault);

То есть, получаем тело сообщения, добавляем туда тег об ошибке, получаем по коду ошибки ее словесное описание и вызываем исключение, которое отправляет клиенту сообщение об ошибке.

4.3 Формирование подписи

Если же все хорошо, то метод возвращает true и передает сообщение либо другому такому же обработчику либо уже главному классу, то есть вызывается веб метод. Если же пришел ответ, то он аналогичным образом проверятся на беду, а именно ищутся значения тегов FirstName и MiddleName, в которых хранятся либо данные о найденном человеке, либо флаг и код ошибки. Если ответ был составлен верно, и ничего плохого не произошло, то выполняется подписание сообщения. Для этого используется класс MessageSignature. В конструкторе данного класса было проведено считывание пути к хранилищу ключей, паролей для доступа к нему и интересующей нас паре ключей и создан экземпляр класса MessageSignature с заданными параметрами:

MS = new MessageSignature(KeyAlg,provName);

File f = new File(KeyStorageFile);

FileInputStream in = new FileInputStream(f);

char[] alias = StringToChar(aliasKey);

char[] storage = StringToChar(storageKey);

KeyPair kp = MS.getPair(in, aliasName, storage, alias);

MS.setPrivateKey(kp.getPrivate());

MS.setPublicKey(kp.getPublic());

Таким образом у нас на стадии подписание уже имеется пара ключей и сертификат, осталось дело за малым – подписать таки сообщение. Подпись генерируется с помощью java пакета java.security. Безусловно из курса по защите информации известны определенные алгоритмы шифрования, однако тут мы имеем дело с ключами большого размера, и если писать все обработчики по длинной арифметике самому, то проект становится экономически не выгоден. Посему была прочитана документация по покету, и создание подписи вылилось в такой кусок кода:

signature.initSign(privateKey);

BufferedInputStream bufRead = new BufferedInputStream(msgPath);

byte[] byteMsg = new byte[bufRead.available()];

bufRead.read(byteMsg);

signature.update(byteMsg);

bufRead.close();

realSign = signature.sign();

sgnPath.write(realSign);

Далее с помощью soapHeader.addHeaderElement() изменяем заголовок пакета, к примеру добавление значения электронной цифровой подписи делается таким образом:

soapHeader.addHeaderElement(new QName("ds","Secur ity")).addChildElement("SignatureValue").setTextContent(new BigInteger( 1, MS. getSign()).toString(16));

Таким образом добавляем в заголовок еще сертификат открытого ключа, что бы клиент мог получить открытый ключ, заверенный удостоверяющим центром, и проверить целостность сообщения и то, что оно было прислано именно нами, а не мошенниками.

Если же мы хотим протестировать сервис, то можно создать свое хранилище ключей, создать там пару ключей и сгенерировать самоподписанный сертификат, будто бы мы и есть удостоверяющий центр. Для этого используется поставляемая с java standart edition утилита под названием keystore. А так то, для создания полноценной заверенной подписи нужно выполнить несколько шагов (исключается случай, когда вы покупаете ключ):

* Сгенерировать пару ключей (закрытый и открытый), при чем в java при генерации ключей сразу создается самоподписанный (self-signed) сертификат, который можно сразу же использовать для тестирования;
* Сгенерировать запрос на получения сертификата;
* Получить сертификат, подписанный доверенный удостоверяющим центром;
* Импортировать сертификат сделанный удостоверяющим центром в ваше хранилище ключей;
* Импортировать этот же сертификат в ваше хранилище доверенных сертификатов (truststore) как «trusted certificate»

Для генерирования ключей используется команда:

keytool -genkey -keystore server.jks -dname "CN=localhost, OU=22, O=22, L=Unknown, ST=Unknown, C=RU" -storepass serverpass -alias server-test -keypass serverpass

где:

* genkey – команда для генерации ключей
* keystore – команда для задания имени хранилища ключей, соответственно если хранилище не найдено, будет создано новое;
* dname – указываем наши данные, как владельца подписи
* CN – имя владельца
* OU – департамент или отдел
* O – имя организации
* L – месторасположение
* ST – имя области, штата, провинции или типо того;
* С – страна, двубуквенное обозначеное;
* Storepass – команда, после которой идет указание пароля для доступа к хранилищу ключей;
* Alias – команда после которой указывается псевдоним, под которым будут храниться ключи;
* Keypass – команда, после которой указывается пароль для доступа к ключам.

Часть из атрибутов указываемых под кавычками может быть пропущена, ну а если хранилище создается лишь с целью тестирования, то значения им можно присваивать абсолютно любые. Сертификат создастся в формате X.509, в котором в качестве идентификатора владельца используется distinquished name (сокраoщенно DN). Внутри каждого такого сертификата хранится пара DN:

* первый принадлежит владельцу сертификата
* второй же указывает идентификатор certificate authority (удостоверяющий центр выше, сокращенно CA).

Для того, что бы сгенерировать запрос на подпись сертификата, который можно отправить в CA для получения надежного сертификата, используется команда «-certreq»:

keytool -certreq -alias server-test -file test.csr -keystore server.jks

где:

* certreq – команда, которая указывает, что будет создаваться сертификат;
* alias – псевдоним, под которым хранятся ключи в хранилище ключей;
* file – имя, под которым будет сохранен сертификат, желательно что бы оно имело расширение .csr;
* keystore – название хранилища ключей.

Далее если вы получили подписанный сертификат от CA, то его можно импортировать в хранилище ключей. Следующая команда импортирует сертификат под именем «cert.crt» в хранилище ключей «server.jks» под псевдонимом «server-test»:

keytool –importcert –trustcacerts –file cert.crt –alias server-test –keystore server.jks

4.3 Авторизация проверкой кеш-суммы

Для принятия решения о разрешении на собирание ответа использовалось сравнение кеш суммы строки, полученной в результате конкатенации параметров login и password, присланными клиентом. Было изучено несколько алгоритмов, а именно MD5 и SHA1. Для подсчета кеш сумм использовался пакет java.security, в котором можно уже реализован ряд алгоритмов. Таким образом подсчет кеш суммы выполняется так:

MessageDigest res;

byte[] temp = new byte[0];

res = MessageDigest.getInstance("MD5");

res.reset();

res.update(source.getBytes());

temp = res.digest();

Однако мы столкнулись с проблемой того, что результат подсчета надо бы перевести в строку, что бы сэкономить память, да и наглядней выходит, чем просто массив байтов. Простым методом toString() не выйдет, поэтому вначале идет перевод в BigInteger, а зачем уже, здоровенное число в строку:

BigInteger bigInt = new BigInteger(1, temp);

String MD5H = bigInt.toString(16);

while( MD5H.length() < 32 ){

MD5H = "0" + MD5H;

4.4 Работа с базой данных

Когда появились и стали развиваться базы данных, то для многих разработчиков встал вопрос как ими пользоваться на уровне языка программирования? Дело в том, что у всех баз данных различные интерфейсы, разный sql, свои тонкости, и что бы изучить все это потребовалось бы очень много времени. Однако компания sun, разработчик языка java пошли иным путем. Вместо того, что бы как то добавлять различного рода методы для работы с кучей разного рода баз данных, они сделали такую вещь как «Java DataBase Connectivity» (далее jdbc), что по-русски означает соединение с базами данных на java. Это так называемый промышленный стандарт взаимодействия java-приложения с различными системами управления базами данных. Он представляет собой пакет java.sql и входит в состав java standart edition. Данный пакет основана драйверах, которые загружаются в рантайме, то есть во время работы программы. Когда используешь метод из данного пакета, он в свою очередь использует методы из драйвера для определенной базы данных. То есть, для того что бы соединиться с какой либо базой данных на java нужно получить в свое распоряжение драйвер, который заточен под работу с той базой данных которая вам нужна. Конкретно для работы с oracle database нужен jdbc драйвер скачанный с официального сайта sun, и такой же версии как и база данных, с которой будет производить соединение и работа. То есть прекрасно то, что:

* разработчик может не знать специфики базы данных, с которой он работает, будь то oracle database, mysql, postgreSQL;
* при переходе компании (работы) на другу базу данных, код практически не меняется, разве что немного из-за различий sql различных систем управления базами данных;
* не нужно никаких дополнительных установок, за исключением небольшого драйвера ля нужной базы данных;
* к любой базе данных можно подключиться через легкий единый указатель ресурса.

Итак, опишем действия, которые проводились для работы с базой данных ГИССЗ. Для начала нужно зарегистрировать jdbc драйвер:

Class.forName("oracle.jdbc.OracleDriver");

Этим мы загружаем драйвер в оперативную память и даем возможность методом из пакета java.sql использовать функции для работы с oracle database. Для работы с другой базой данных нужен другой единый указатель ресурса. Для того, что бы компилятор нашел драйвер, нужно либо в среде разработки добавить его в ресурсы либо указать драйвер сторонним ресурсом при ручной компиляции.

connection = DriverManager.getConnection("jdbc: oracle:thin:@localhost:" + port+"/" + SID, login, password);

Данная строчка устанавливает соединение с базой данных oracle, которая используется порт под номером port, имя базы данных хранится в SID, а login и password означают имя пользователя и пароль, словом, данные пользователя под которым мы подключаемся к базе данных.

Для того что бы что то получить из базы данных или изменить ее нужно получить ее текущее состояние. Делается это с помощью опять таки того же пакета java.sql, а именно тип Statement, метод для класса Connection createStatement:

Statement statement = null;

statement = connection.createStatement();

ResultSet rs = null;

rs = statement.executeQuery(sql);

В представленной выше части кода создается состояние базы данных, а так же создается переменнтая типа ResultSet. Это специальный тип пакета java для работы с базами данных, который в упрощенном виде можно представить в виде таблицы и хорошему числу методов к ней. Но на деле, rs будет чем то вроде указателя на текущую запись (строка в таблице-результате), полученную в результате выполнения sql-запроса.

Все методы, которые реализованы в классе getmesomething, имеют одинаковую структуру:

* создание переменной String sql = "", к которой хранится нужный для этого метода sql запрос;
* вызов метода SimpleSelect(String) из класса DB, то есть получение таблицы-результата;
* доставание из полученной таблицы нужное поле.

В коде выглядит это следующим образом:

String sql = " select \* from peoples where…."

rs = db.SimpleSelect(sql);

while (rs.next())

int temp = rs.getInt(4);

res = "г. "+rs.getString(1);

house = rs.getString(2);

house\_x = rs.getString(3);

if (temp == 1 || temp == 3) break;

Очевидно, так как разработка ведется для министерства труда, мы не можем просто взять и показать те sql запросы, которые использовались для получения данных из ГИССЗ, так как это секретная информация, и ее разглашение влечет за собой как уязвимость государственной базы данных для мошенников, так и некоторое наказание.

В большинстве случаев sql запрос составлен таким образом что бы результат состоял лишь из нескольких полей, и легко доставался. Однако для получения таблицы с размерами выплат помесячно потребовалось использовать более интересный sql нежели просто какой то один select. Было создано несколько типов, написан ряд функций. В конечно итоге вылилось в хорошую функцию, которую используют сотрудники, и для которой нужны четыре параметра:

* номер дела, по которому человек получает выплату;
* тип этого дела;
* дата начала периода;
* дата окончания периода.

4.5 Первоначальная настройка

Когда получен работающий макет, надо бы сделать его более пригодным для реального использования, сделать его более гибким в настройке, ведь ветер может перемениться, сегодня мы используем MD5 в качестве алгоритма подсчета хеш суммы, а завтра хеш сумму сможет подделать какой нибудь школьник, потому что вдруг фейри таблицы стали общедоступными. Фейри таблицы – это таблицы аля полный перебор, пароль и хеш сумма, нужны они для поиска коллизий. Таким образом если будет возможно по-быстрому сменить алгоритм подсчета хеш суммы(или практически любой другой важный кусок сервиса), то это будет прекрасно. Для таких целей разработчики обычно создают файлы с расширением .ini (от английского слова initialization). Казалось бы, они могут быть очень сложными, возможно в каких то больших проектах да, но проанализировав один из файлов настройки игры «World Of Warcraft» от компании Blizard,, можно увидеть что файл состоит из строк такого типа «SET имя\_переменной значение». Вот кусок из этого файла, который находится в свободном доступе:

SET checkAddonVersion "0"

SET lastAddonVersion "70200"

SET accountName "mikita.vaseryn@gmail.com"

SET accountList "!WoW1|"

SET lastCharacterIndex "3"

SET worldPreloadNonCritical "0"

SET lodObjectFadeScale "80.000000"

SET RAIDlodObjectFadeScale "80.000000"

SET raidGraphicsTextureResolution "2"

SET gxApi "D3D11"

SET Sound\_EnableDSPEffects "0"

SET Sound\_NumChannels "64"

SET raidGraphicsViewDistance "4".

В данном проекте было решено не городить огород, а пойти таким же путем. Как говорится простота – путь к успеху. Был изучен пакет java.Properties, который позволяет легко читать файлы формата ключ значение. С его помощью можно задавать разделяющие символы, читать массивы, разные сложные структуры данных ,однако в данном проекте мы смогли обойтись простотой. Файл config.ini состоит из строк по типу «ключ = значение». То есть если нужно поставить параметр ModeAllowed в значение один, то следует в файле конфигурации написать «ModeAllowed = 1». Более подробно о настройке приложения можно прочитать в разделе руководство пользователя, а пока что остановимся на чтении параметров из файла с помощью java.

FileInputStream cfg;

Properties property = new Properties();

cfg = new FileInputStream("config.ini");

property.load(cfg);

KeyLength = Integer.valueOf(property.getProperty ("KeyLength","0"));

aliasName = property.getProperty("aliasName");

Как видно в начале создается поток ввода, затем экземпляр класса Properties, и с помощью метода load в него загружается файл конфигурации. Затем используя метод getProperty, указывая в нем значения ключа тянем из файла конфигурации значения. Причем возвращаются значения типа String, а значит если мы хотим инициализировать переменную типа int, то придется воспользоваться преобразованием типов, а так как значение null никак не может преобразоваться в целочисленный тип, нужно указать значение по умолчанию, которое вернется, если не было найдено в файле конфигурации значения для заданного ключа.