**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное учреждение высшего образования

**“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ ИТМО**”

**Факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА № 3**

**ПО ДИСЦИПЛИНЕ «ТЕХНОЛОГИИ ВЕБ-СЕРВИСОВ»**

Студент: Носков Антон Андреевич

Группа: P41142

Преподаватель: Дергачев Андрей Михайлович

Санкт-Петербург

2020

1.Техническое задание

Основываясь на информации из раздела 2.8, добавить поддержку обработки ошибок в сервис реализованный во второй лабораторной работе.

В соответствии с изменениями сервиса необходимо обновить и клиентское приложение.

2.Описание работы

Сервер написан на ejb, с использованием jdbc для подключения к базе данных. В качестве базы данных используется MySQL. Таблица использованная в лабораторной работе называется People и состоит из 5 полей: Id: int, Name: varchar(20), SecondName: varchar(20), Age: int, Sex: varchar(5).

Было добавлено два класса ошибок: MissFormatException – ошибка, возникающая при несовпадении форматов полей и OutOfRangeException – ошибка, которая возникает, когда клиент пытается удалить или обновить запись с id которого не существует.

Лабораторная работа и консольный клиент расположены на GitHub по адресу: https://github.com/anton-mix/WebLabs

3.Вопросы на защиту

1. Предположим, при реализации SOAP веб-сервиса Вам необходимо передавать двоичные данные (аватары пользователей, архивы...). Какие стратегии Вы можете предложить для работы с binary attachments? Реализуйте одну из них.

Ответ на этот вопрос есть в вопросе 3.

1. Для чего нужны аннотации @Produces и @Consumes при реализации REST-сервиса?

Аннотация @Produces нужна чтобы указать какой MIME-тип сервер отправит клиенту при обращению к методу. Аннотация может применяться, как и к отдельному методу так и к целому классу, указывая MIME-тип для всех его методов.

Пример:

@Produces({"image/jpeg,image/png"})

Аннотация @Consumes нужна чтобы указать какой MIME-тип сервер получит от клиента при получение ответа от клиента. Аннотация может применяться, как и к отдельному методу так и к целому классу, указывая MIME-тип для всех его методов.

Пример:

@Consumes ({"image/jpeg,image/png"})

1. Модифицируйте REST и SOAP сервисы так, чтобы для CREATE, DELETE, UPDATE операций требовалась Basic аутентификация. Соответственно внесите такие же изменения в клиентские приложения. Login / Password для простоты можно захардкодить.

Basic аутентификация в SOAP происходит следующим образом:

* 1. Клиент при отправке SOAP сообщения, отправляет в заголовках сообщения логин и пароль.
  2. Сервер из контекста сообщения достаёт логин и пароль и проверяет совпадают ли они и если совпадают, то разрешает операцию, если нет, то генерирует исключение.

Basic аутентификация в Rest происходит следующим образом:

* 1. Клиент при отправки сообщения создаёт заголовок с именем authorization и кладёт туда зашифрованные логин и пароль в формате “username:password”
  2. Сервер при получении сообщения достаёт из него заголовок authorization, расшифровывает его и если такой пользователь есть, то разрешает операцию если нет, то генерирует исключение.

1. Предположим, что одним из нефункциональных требований к вашему сервису является поддержка throttling (умышленное ограничение количества одновременно выполняемых запросов; если поступает новый запрос, а в это время уже выполняется максимально разрешенное количество, то необходимо прервать выполнение запроса, например, путем выброса ThrottlingException (класс исключения нужно создать самостоятельно)). Как можно это реализовать?

Одним из решений этой проблемы будет использовать общие ресурсы сервиса. Можно создать три переменных: Максимальное количество клиентов, текущее количество клиентов и статическую переменную класса object для синхронизации. Затем в начале каждого web-метода добавить блок synchronized и проверку количества пользователей, а на выходе из web-метода уменьшать это количество. Если количество пользователей превышает максимальное, то выбрасывается ThrottlingException.

1. Реализуйте endpoint, позволяющий юзеру загружать файлы (для простоты не нужно делать никаких проверок корректности имени файлов, дубликатов файлов, итп). Также реализуйте клиентскую часть, взаимодействующую с сервисом.

Для работы с файлами можно воспользоваться механизмом MTOM. Он позволяет SOAP отправлять бинарные файлы. В тело сообщения на место файла ложится специальный идентификатор файла, а сами бинарные данные идут после soap сообщения.

1. В чем принципиальное различие при реализации обработки ошибок в REST и SOAP-сервисах с точки зрения разработки сервиса, а также с точки зрения реализации клиента?

Для сервера реализация исключений в soap от rest отличается стандартизацией. В soap класс исключения должен быть помечен специальной аннотацией @WebFault, класс должен иметь два конструктора со специальными параметрами и реализовывать два метода getFaultInfo() и getMessage(). Так же для работы с этим исключением понадобится специальный bean, который будет занимается отправкой.

В rest не требуется стандартизации. Достаточно наследовать класс исключения от класса Exception и реализовать mapper для работы с исключением.

Для клиента при возникновении исключения в soap приложение само определит тип исключение и породит его для клиента. Поймать его можно с помощью try-chach.

В rest исключение можно определить по коду ответа от сервера. В теле ответа будет сообщение исключения.