SENG3400/6400 Network and Distributed Computing Assignment 2 – 25%

Due Date: 11:59PM Friday 2nd November 2018 (Week 12)

Note: Please read the important information at the end of this specification!

Before you begin, you should familiarise yourself with Apache AXIS by following the example given in the lecture slides for Java Web Services and SOAP. This example will lead you through setting up a Tomcat web server to deploy web services and will show you how to generate a WSDL and stub code for clients. It also shows you how to setup your environment to work correctly with Tomcat and AXIS.

Microservices

Microservices are currently a 'fad' in the realm of web engineering. Instead of having a single monolithic application which implements diverse but distinct features, microservice-based applications are composed of multiple distinct applications, each of which implements a distinct feature. Each microservice will implement a well-defined piece of functionality exposed as an *'Endpoint'*, and delegate all unrelated functionality to another microservice.

Your task in this assignment is to implement four distinct applications. Two applications will be architected in a microservice-based manner running in separate web apps on the same instance of Apache Tomcat. Two applications will be command-line applications consuming the endpoints exposed by the micro services.

Application 1 – Identity Service

The identity service provides user authorisation. It will expose two endpoints. It should be deployed to the context path 'identity'.

The first endpoint is named Login. It exposes two methods:

```
String login(String username, String password)
```

The login method accepts a username and password. You do not have to store a list of known usernames and passwords, you only need to authenticate at a minimum the following users:

```
U: hayden, P: 1234
U: josh, P: 4321
```

If an invalid username/password pair is provided the method call will return the string “INVALID”.

On successful login, the application will return a random 5-character key which will be used to identify the user. This is called a *session key*. The key is randomly generated on every call to this method. It has to be unique for each logged in user (i.e. two users cannot have the same session key when logged in at the same time). The server will have to remember the session key provided to the user. The Currency Service will use the Authorisation endpoint to validate this key. There is no timeout for a key. An existing key will be overridden if the user makes subsequent calls to this method (i.e. logs in again). You only have to remember the most recent key.

```
Boolean logout(String key)
```

The logout method will invalidate a user’s key. i.e. the server will ‘forget’ that the key is associated to the user, and any subsequent authorisation requests will fail if this key is passed. True is returned if there was a valid session that is now logged out, otherwise false.

The second endpoint is named Authorisation. It will expose one method:

```
Boolean authorise(String key)
```

The authorise method validates the passed key. If the key is currently associated with a user, (i.e. the user has logged in) it will return true. Otherwise, false.

Application 2 – Currency Service

The currency service manages all aspects of currency listing and conversion. It will expose two endpoints. It should be deployed to the context path ‘currency’.

The first endpoint is named Conversion. It will expose three methods:

```
String[] listRates()
```

Returns an array of all known conversion rates formatted as:

```
<from>-<to>:<rate>
E.g. AUD-USD:0.73
```

```
Double rateOf(String fromCurrencyCode, String toCurrencyCode)
```

Returns the conversion rate (as a double) of the specified conversion pair.

E.g. If the conversion rate from AUD to USD is 0.73, this call will return 0.73
If the specified pair does not exist, returns -1.0

```
Double convert(String fromCurrencyCode, String toCurrencyCode,
               Double amount)
```

The currency conversion service charges a 1% conversion fee. Conversions are calculated using the equation:

```
convertedAmount = amount * rate * 0.99
```

Not all currencies can be traded. A currency can only be traded if a conversion rate exists between the currencies. If a conversion between the specified pair does not exist, a value of -1.0 is returned.

The second endpoint is named Admin. It will expose eight methods:
All calls to the Admin endpoint require a session key. The implementation of the endpoint will need to validate the session key with the Authorisation Endpoint. If the authorisation fails, the Admin endpoint needs to throw an exception. *You may decide what exception to throw (either Java provided or custom).*

```
Boolean addCurrency(String sessionKey, String currencyCode)
```

Registers a currency to be converted by the Currency Service. This does not make the currency tradeable, a currency is only tradeable if it has a conversion rate set.

```
Boolean removeCurrency(String sessionKey, String currencyCode)
```

Removes a currency, and all associated conversion rates from the conversion service.

```
String[] listCurrencies(String sessionKey)
```

Returns an array of supported currencies.

```
String[] conversionsFor(String sessionKey,
                        String currencyCode)
```

Returns a list of all possible conversions with rate for the specified currency code.

```
E.g. for currency code AUD:
AUD-USD:0.7
AUD-GBP:0.55
AUD-NZD:1.09
```

```
Boolean addRate(String sessionKey, String fromCurrencyCode,
                String toCurrencyCode, Double conversionRate)
```

Adds a conversion rate between the two passed currency codes.

This method will add a conversion rate between both specified currencies. If a conversion rate *r* from currency *A* to currency *B* is inserted, a conversion rate from *B* to *A* must recorded at (1/*r, i.e. the inverse)*.

Returns false if any passed currency codes are not registered, if the passed currency codes already have a specified conversion rate, or if the conversionRate is less than or equal to zero.

```
Boolean updateRate(String sessionKey, String fromCurrencyCode,
                   String toCurrencyCode, Double rate)
```

Updates the conversion rate between the specified currencies.

This method will have to add a conversion rate between both specified currencies. If a conversion rate *r* from currency *A* to currency is updated, a conversion rate from *B* to *A* must recorded at (1/*r)*.

Returns false if any passed currency codes are not registered, if the passed currency codes do not have a specified conversion rate, or if the conversionRate is less than or equal to zero.

```
Boolean removeRate(String sessionKey, String fromCurrencyCode,
                   String toCurrencyCode)
```

Removes **all** conversion rates between the specified currencies.

Returns false if any passed currency codes are not registered, if the passed currency codes do not have a specified conversion rate.

```
String[] listRates(String sessionKey)
```

Returns an array of all known conversion rates formatted as:
```
<from>-<to>:<rate>
E.g. AUD-USD:0.73
```

Application 3 – Admin Client

The Admin Client is a command line application which allows an administrator to manage the Conversion Service. It will consume services offered by the Admin endpoint. **It must be implemented in a Java class called 'AdminClient'. Do not place this class in a package!**

The functionality of the Admin Client mirrors the functionality offered by the Admin endpoint. It is simply exposing it through a terminal interface. However, the client must facilitate the ability for a user to authenticate with the Identity Service, and hence pass the session key to all calls to the Admin endpoint.

The user will start the Admin Client through the command:
`java AdminClient <username>`

The client will first prompt the user for their password. The application must handle this input securely, see:
https://stackoverflow.com/questions/8138411/masking-password-input-from-the-console-java

The application will request for the Login endpoint to validate the username and password. If a valid session key is returned, the application will record it. If the username and password is incorrect, the application will request the user for their username and password two more times, before exiting on the third invalid attempt.

On successful login, the application will present the user with a prompt. This is a free-form text prompt. The user may type one of 9 commands at the prompt, supplying all required parameters. After a command is executed, the application will return to the prompt. If an invalid command is supplied, the application will present an error and return to the prompt. The commands (with parameters) consist of:

1. `addCurrency <currencyCode>`
2. `removeCurrency <currencyCode>`
3. `listCurrencies`
4. `conversionsFor <currencyCode>`
5. `addRate <fromCurrency> <toCurrency> <rate>`
6. `updateRate <fromCurrency> <toCurrency> <rate>`
7. `removeRate <fromCurrency> <toCurrency>`
8. `listRates`
9. `logout`

The specified commands are not to be passed at command line, only through an interactive prompt internal to the application.

All specified commands mirror the services supplied by the Admin endpoint, but the logout command. The logout command will call the logout method of the Admin endpoint, report failure/success and exit the program.

When a command is entered by the user, your application will identify what command is to be executed and invoke the correct method on the admin endpoint client. All calls to this client require the user's session key passed as the first argument.

You are expected to handle all errors and exceptions correctly. If an exception is returned by any of the admin endpoint methods, you are to inform the user what the cause of the exception is and return back to the prompt.

Application 4 – Conversion Client

The Conversion Client is a command line application which allows a user to calculate currency conversions. It will consume services offered by the Conversion endpoint. **It must be implemented in a Java class called 'CurrencyClient'. Do not place this class in a package!**

The application does not require a user to login, all functionality is unrestricted. The application will present the user with a prompt. This is a free-form text prompt. The user may type one of 4 commands at the prompt, supplying all required parameters. After a command is executed, the application will return to the prompt. If an invalid command is supplied, the application will present an error and return to the prompt.

```
1. convert <fromCurrency> <toCurrency> <amount>
2. rateOf <fromCurrency> <toCurrency>
3. listRates
4. exit
```

The first 3 commands mirror the commands provided by the Currency Conversion Service Endpoint. The exit command will exit the program.

You are expected to handle all errors and exceptions correctly.

Default currencies and conversion rates

You may add any currencies and conversion rates to your implementation of the system, however the following currencies and conversion rates must be listed by default:

From Currency	***To Currency***	***Rate***
AUD	USD	0.7
AUD	NZD	1.09
AUD	GBP	0.55

Note: Conversions need to be inserted between both currencies. If a conversion rate *r* from currency *A* to currency is required, a conversion rate from *B* to *A* must recorded at (1/*r*).

Note: There is no requirement for all known currencies to have conversion rates. If the user requests a conversion between two currencies that do not have a conversion rate, you are not to try and find 'intermediate' conversions to facilitate the transaction. If the conversion cannot take place, simply inform the user in the client that the conversion is not supported.

Getting Started

It may appear that this assignment is complicated, but that is only superficial. In industry it is commonplace to integrate multiple applications by consuming some type of service. The recommended plan of attack for this assignment is to implement the applications in the following order:

1. Identity Service
2. Conversion Service
3. Conversion Client
4. Admin Client

You are free to implement the applications as you wish within the bounds of the specification.

A simple usage of the admin client application may be:

```
java AdminClient Hayden
Enter password >> ******
Welcome, Hayden

What would you like to do >> listCurrencies
AUD
USD
GBP
NZD

What would you like to do >> addCurrency BTC
Currency 'BTC' Created

What would you like to do >> addRate AUD BTC 9000.00
Rate AUD-BTC:9000.00
```

```
What would you like to do >> logout
Goodbye!
```

The clients effectively wrap the web services, exposing them to the client. The admin client performs a small amount of pre-processing to allow a user to authenticate and hence grant access to the admin endpoint. All numerical values supplied through the command line are expected to be doubles. Numerical values are to be printed to 4dp to the user.

The specified services must be deployed to the correct context path (i.e. the identity service must be implemented in a folder named 'identity' in the Tomcat webapps directory). The specified endpoints must be implemented in corresponding classes. (i.e. the Login endpoint must be implemented in a .jws file named Login.jws). This specification lists the bare minimum classes present, but you may implement as many classes as you require for the application.

You are expected to provide appropriate feedback to the marker as to what is occurring in your web services. i.e. on each request tell us what is happening, the result of any service calls, why an error occurred, etc… You are expected to handle exceptions gracefully. You must catch all exceptions which may occur with network communication. In each client, exit the program if a networking-related exception is thrown (informing the user of the reason. In the services if an exception occurs, print this to stdout and return the default/error response for the method call.

Submission

You are not to use any third-party libraries apart from Apache AXIS. If you server requires beans to be compiled, please specify this in your submission.

Compiling the services & clients should be as simple as javac *.java. You must include all sources. If there are any extra steps to compilation, please document this in your submission. You may assume that any Apache Axis dependencies are correctly configured in the markers Classpath.

All assignments are to be submitted via Blackboard. Assignments are to be implemented using Java 8 & Axis 1.4, running on Apache Tomcat 8.5.

Assignments which do not follow this specification will not be marked. Submissions are as Zip files only! Your submission will include the four applications (each in their own folder & clearly labelled as per the application name) with all required sources. You will also submit the four generated WSDL files (one for each endpoint) in their own folder named 'wsdls'. Do not submit nested zip folders. Name your submission c<stdno>_a2.zip.

Marking Scheme

Total marks: 25 (25% of your final course mark)

Conformance to Specification – 10 marks (2.5 marks per application)

- Are your web services correctly named?
- Are your clients correctly named?
- Do you implement the described processes?

Functionality – 10 marks (2.5 marks per application)

- Do you implement all required functionality?
- Do you have bugs?
- Do you handle errors correctly?
- Do you implement the described processes?

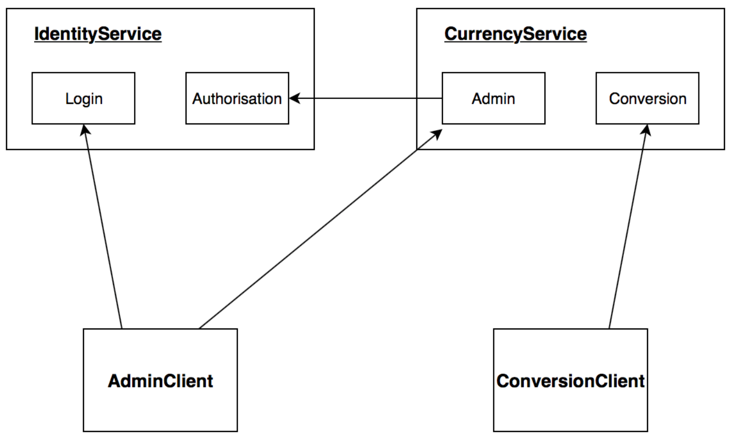
Code Quality – 5 marks

- Naming conventions?
- Commenting & Documentation?
- Formatting?
- Readability?
- Error handling?

We won't be testing your implementation with a 'correct' implementation in this assignment. Assignments will be marked in respect to conformance to the specification and the quality of your code & implementation.

You are not to place all web service endpoints into the same web app context, or the same .jws files. If you do so, you will be deducted marks. If you do not name your service endpoints or methods according to the spec, you will lose marks.

Conceptual Overview


IdentityService
Login
Authorisation
CurrencyService
Admin
Conversion
AdminClient
ConversionClient