Faculty of Science
Department of Computing

ITEC833: Web Services

Report for Group Assignment

Student No.	Student Name
42431646	Poopak Alaeifar (Mary)
42756561	Shamim Ahmed

Date: 25/5/12

Declaration

We hereby declare that the work presented is our own. When other sources of information have been used, they have been acknowledged.

Introduction

Our team is trying to build a portal-like system that allows a user to perform customized search based on specific keywords. The user can register and then login to the application. After login, he has the option to specify a set of keywords. Then the user can initiate a search for a specific type of content and search results are grouped according to the keywords he specified earlier.

The preference of the user is saved in persistent storage so that it is available for future sessions.

We support search for the following types of content:

- News from The Guardian (http://www.guardian.co.uk)
- Tweets (http://www.twitter.com)
- YouTube Videos (http://www.youtube.com)
- Flickr (http://www.flickr.com)

Apart from these contents, our application also allows users to view weather forecast for a particular city and country. We use a SOAP-based service called 'GlobalWeather' as the source of weather data.

The system has two key parts:

- A web service that communicates with external content providers and save user preferences in database.
- A client webapp that communicates with the web service.

Web Service Technologies

Our system is Java-based. The technologies we're using in our system are:

- SOAP
- REST
- WSDL
- WS-Security

SOAP

SOAP is the messaging framework for communication between the web service and the client webapp. We are using Apache Axis2 as the SOAP engine.

The interaction with external weather forecast provider is also done via SOAP. However, in this case, the SOAP messages are constructed and consumed manually (Axis2 is not used).

REST

Our web service communicates with most external content providers via their RESTful APIs. We depend on the REST-based API of the following content providers:

- The Guardian
- YouTube
- Twitter
- Flickr

We've chosen these content providers because all of them have public APIs that can be used without any need for sign-up and/or authentication and the APIs are well-documented.

We typically specify the following parameters for filtering the content when we use these APIs:

- A keyword
- The number of results to be returned
- Sorting criteria (e.g., latest items first)

WSDL

The web service exposes its interface to the client webapp via WSDL. For generating the WSDL interface, we use Apache Axis2. Axis2 also comes with an integrated set of tools that can be used to parse the generated WSDL and generate client-side proxy classes from it.

The methods exposed by the web service over WSDL interface are listed below:

- register
- login
- logout
- addTag
- removeTag
- getTags
- getVideos
- getNews
- getTweets
- getForecast

WS-Security

We use Apache Rampart, which is a module of Apache Axis2, for incorporating some features of WS-Security in our application.

We'll support security in three modes:

- message signing

- message encryption
- message signing and encryption

Some configuration files on both client webapp and the web service need to be changed to enable each of these modes.

Other Tools

Apache Maven

We use Apache Maven as the build tool for the project. Maven allows developers to specify dependencies on external libraries declaratively (via xml) and facilitates the build and test process.

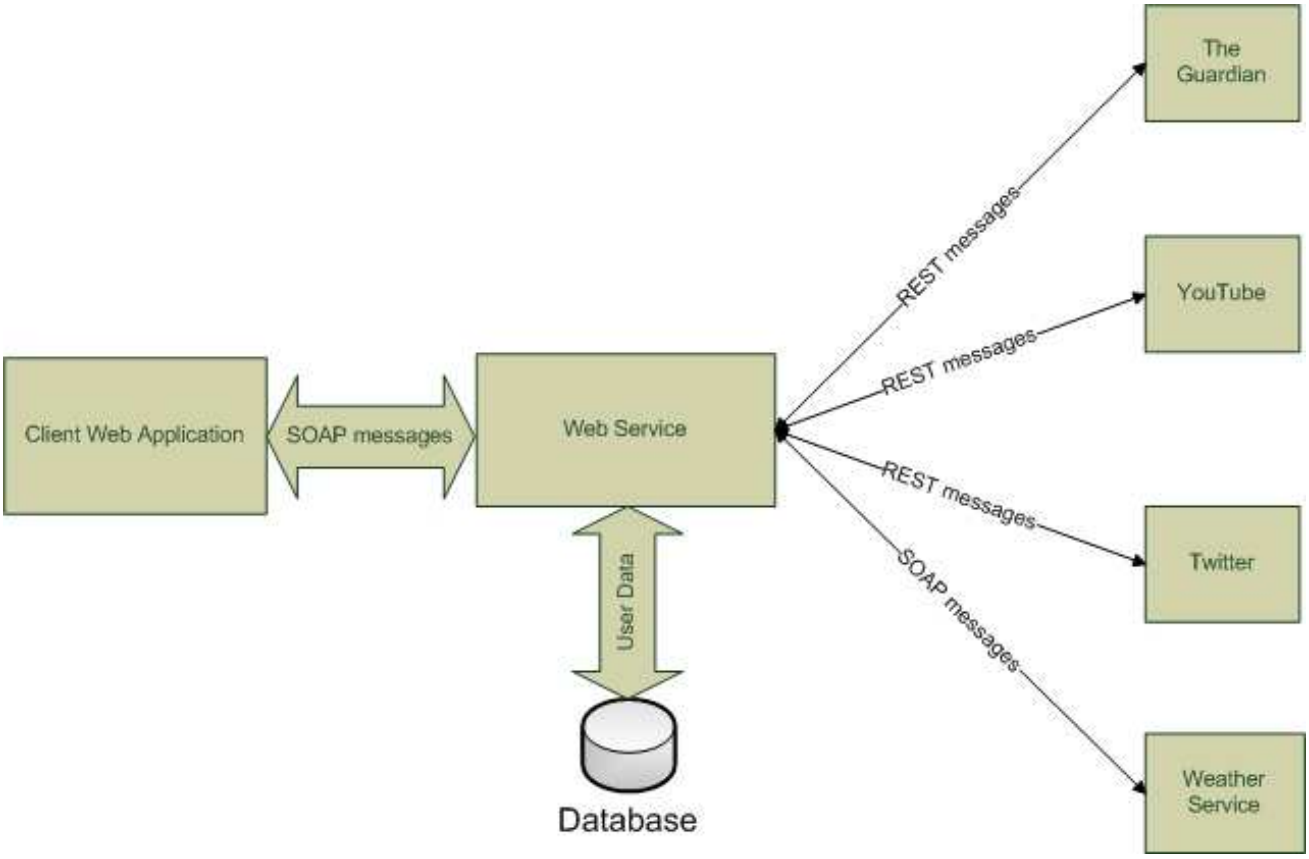
Apache Ant

Apache Ant is used in conjunction with Maven to perform some custom steps during the build process (e.g., packaging the right configuration files for security with the web archive file)

System Architecture

Architecture Diagram

The following diagram shows the high-level architecture of our system:



Key Aspects of System Architecture

The most important aspects of our system are listed below:

- All components of the system are implemented using Java and J2EE technologies.
- The web service is the central component in our system. It is an Axis2-based web service that is deployed in a J2EE web container. After successful deployment, the web service exposes its interface via WSDL.
- We use tomcat as the Servlet container in which the Axis2 web archive file (.war file) is deployed.
- We use the database connection pooling feature of Tomcat to make our database transactions more efficient.
- The web service is responsible for saving user credentials and preferences in a database. We use Apache Derby as the database with which the service connects and performs transactions. We've chosen Derby because it is shipped with the Java Development Kit (along with the necessary libraries), so installing an additional product is not required.
- The web service is responsible for managing communication with external content providers, depending on the action the client webapp requested. The communication between the web service and the external content providers may be REST or SOAP-based.
- For sending HTTP requests and receiving HTTP responses, we use libraries from Apache HttpComponents project. These libraries provide convenient utilities for low-level tasks related to HTTP protocol.
- The web service parses the XML responses it receives from external content providers and presents it to the client webapp in a standard, consistent format. The response from the web service contains all necessary information required by the client webapp. In this way, the client webapp is shielded from various differences in data formats exhibited by different external services.
- All communication between the client webapp and the web service takes place via SOAP. The client webapp only needs to know the service endpoint address. Once this is provided, it can use utility classes from the Axis2 API to invoke the operations exposed by the web service via WSDL.
- The client web application may or may not be deployed in the same web container as the web service. However, if they are deployed in different web containers, then required Axis2 libraries must be present in the server class path in both cases.
- Session management is handled as part of the web service. When a user logs in, a token is generated and returned by the web service. For each subsequent request, this token needs to be sent back to the service. We pass this token as part of SOAP header.

- In order for WS-Security to work properly, both the client webapp and the web service need to be configured to use the same kind of security mechanism. For example, if we want to use encryption, then both the client webapp and the service need to be deployed with the right configuration files so that the requests and responses are properly understood by both parties.
- For implementing various functionalities of the client webapp, we use Apache Struts 2, which is a popular Model-View-Controller (MVC) framework for Java web applications.
- The client webapp uses web technologies like Java Server Pages (JSP), HTML5 and CSS to render the information returned by the web service.
- The client webapp uses technologies like XSLT to convert the response from the web service to the target format (e.g., HTML).

Short Description of External Services

The Guardian

The Guardian has an open API for content searching. It is a REST-based API that accepts a number of search parameters and returns the response either in XML or JSON format. Some of the important search parameters are listed in the following table:

Parameter	Meaning
q	Search keyword
format	The format of the returned result ('xml' or 'json')
page-size	The number of items to be returned
order-by	The criteria for ordering the search result (e.g., 'newest')

For a detailed explanation of all available options, please refer to the link provided as part of the references.

YouTube

YouTube has a public API which is REST-based. It allows the user to perform a variety of tasks, some of which (e.g., video upload) require authentication. However, our goal is only to perform a search for relevant videos, which can be done without any authentication or authorization.

Some important parameters from the YouTube API (which are relevant to our purpose) are listed below:

Parameter	Meaning
q	Search Keyword
max-results	Maximum number of search results to be returned
start-index	The index from which the listing will start
v	The version of the API

orderby	The sorting criteria for the returned videos (e.g., 'published')

For a comprehensive list of all YouTube API parameters, please refer to the link provided as part of the references.

Twitter

Twitter has a REST-based public search API. It offers a number of parameters to customize the search. It can also return search results in multiple formats.

Some of the important search parameters in Twitter API are listed below:

Parameter	Meaning
q	The search keyword
rpp	Number of results per page
result_type	Specifies what type of result is required (e.g., 'mixed', 'recent', 'popular')

For a detailed list of all search parameters, as well as examples, please refer to the link provided as part of the references.

Flickr

Flickr has a REST-based public search API. It offers a rich interface to retrieve various kinds of information on the photos uploaded by users. Some of the parameters that we used in our application are mentioned below:

Parameter	Meaning
api_key	The API key required to submit the search request
method	The method being invoked. We use the methods 'flickr.photos.search' and 'flickr.people.getinfo'
format	This determines the result format. We use 'rest' as the value of this parameter.
per_page	Number of results to be returned
tags	The keywords to search for

For a detailed list of all search parameters in Flickr API, please refer to the link provided as part of the references.

GlobalWeather

Apart from the REST-based external services described above, we've also integrated our application with a SOAP-based service called 'GlobalWeather'. This service is queried to fetch weather forecast for a particular city and country. This service is also publicly available over the Internet, requiring no registration or authentication.

We communicate with this SOAP web service using the same HTTP client libraries that are used in case of REST-based services. However, the process is slightly more involved in this case because we need to manually construct a SOAP message of the right format and then send the message in the body of a POST request.

According to the documentation, this web service exposes a method 'GetWeather' that takes two parameters: 'City' and 'Country'. The result is returned in XML format.

In order to see the exact format of the SOAP messages for this service, please refer to the link provided as part of the references.

Progress Report

Key Milestones

The key milestones for this project are listed in the following table:

Milestone	**Completion Date**
Submission of Initial Project Proposal	April 6, 2012
Submission of detailed proposal and background documentation	4 May, 2012
Development of the proposed system	24 May, 2012
Final submission of the project	25 May, 2012
Submission of Individual Report	29 May, 2012

Group Members

The following tables identify the role of each group member in the project:

Student Name	Poopak Aaleifar (Mary)
Student ID	42431646
Contribution	- Exploration of the API of external content providers. - Integration of the web service with external content providers. - Parsing the contents returned by external services, and restructuring them to a standard format (to be consumed by the client webapp). - Writing the Struts actions and other. miscellaneous classes for the client webapp. - CSS design for the client webapp.
Approximate percentage of project work done by this student	50%

Student Name	Shamim Ahmed
Student ID	42756561
Contribution	- Implementing the database transaction related functionalities. - Implementing the Axis2 integration on both the client webapp and the web service. - Implementing session management. - Writing the JSP pages for the client webapp.
Approximate percentage of project work done by this student	50%

References

The Guardian 2012, *Content API: Content Search Reference Guide*, The Guardian, viewed May 2, 2012 <http://www.guardian.co.uk/open-platform/content-api-content-search-reference-guide>

Google Developers 2012, *YouTube API v2.0*, Google Inc., viewed May 2, 2012 <https://developers.google.com/youtube/2.0/developers_guide_protocol#Retrieving_and_searching_for_videos>

Twitter Developers 2012, *GET Search*, Twitter Inc., viewed May 2, 2012 <https://dev.twitter.com/docs/api/1/get/search>

WebServiceX.NET 2012*, GlobalWeather*, WebServiceX.NET, viewed May 2, 2012 <http://www.webservicex.com/globalweather.asmx?op=GetWeather>

The Apache Foundation 2012, *Apache Axis2 User Guide*, Apache Axis2, viewed May 2, 2012 <http://axis.apache.org/axis2/java/core/docs/userguide.html >

The Apache Foundation 2012, *Apache Rampart – Axis2 Security Module*, Apache Rampart, viewed May 2, 2012 <http://axis.apache.org/axis2/java/rampart/>

The Apache Foundation 2012, *HttpComponents User Documentation*, Apache HttpComponents, viewed May 2, 2012 <http://hc.apache.org/user-docs.html>

Yahoo Inc. 2012, *Flickr API*, Yahoo, viewed May 20, 2012 <http://www.flickr.com/services/api/flickr.photos.search.html>

Appendix A: How to build the code

Prerequisites

You need to have the following software installed in your system:

- Java Development Kit (>= 1.6)
- Apache Ant
- Apache Maven
- Apache Axis2 1.6.1 installation with Apache Rampart 1.6.1.

Build Instructions

The following is a step-by-step procedure to build the application:

- Extract the archive. You should see a folder named ‘ws-axis2-project’.
- Go to ‘ws-axis2-project/soap-ws/src/main/ant’ folder. Edit build.properties so that ‘axis2.home’ property is set correctly. This property should point to the directory of axis2 installation.
- Go to ‘ws-axis2-project/demo-webapp/src/main/ant’ directory. Edit build.properties and set ‘axis2.home’ property again.
- Now return to ‘ws-axis2-project’ directory and run the command :
  mvn clean compile package
- After the command completes, go to ‘ws-axis2-project/soap-ws/target’ folder. You should see a file named ‘portalservice.aar’. This archive represents the web service. It can be deployed in tomcat as part of Axis2 webapp.
- The demo webapp can be found in ‘ws-axis2-project/demo-webapp/target’ folder as a war file named ‘demo-webapp.war’. This file should be deployed in tomcat as an independent web application.

Appendix B: How to Create the Database

In the following discussion, we assume that Apache Derby is already installed and derby-related binaries are included in the PATH variable.

The following is a step-by-step procedure to create the required tables in database:

- Create a directory where the database related files will be kept. For the sake of discussion, let's call it 'database'.
- Copy the 'derby.sql' and 'derby.properties' files from 'ws-axis2-project/soap-ws/src/main/sql' folder to 'database' folder.
- Go to database folder and run the following sequence of commands:
  ij -p derby.properties
  connect 'jdbc:derby:itec833;create=true';
  run 'derby.sql';
  exit;
  This will ensure that a database with name 'itec833' is created and all the required tables are present in that database.
- In order to start the database, go to 'database' folder and run the command:
  startNetworkServer

Appendix C: How to Configure Tomcat for Database Connection Pooling

Our web service must be deployed in Apache Tomcat as part of the axis2 webapp. In order to connect to the database, it requires a specific JNDI resource.

The following is a step-by-step procedure to configure tomcat for database connection pooling:

- Copy the file 'derbyclient.jar' from your derby installation to '$CATALINA_HOME/lib'
- Open the file '$CATALINA_HOME/conf/context.xml' for editing.
- Add the following XML fragment under <Context> element:

```
<Resource name="jdbc/itec833"
          type="javax.sql.DataSource"  auth="Container"
          description="Derby database for itec833 project"
          maxActive="100" maxIdle="30" maxWait="10000"
          username="root" password="admin"
          driverClassName="org.apache.derby.jdbc.ClientDriver"
          url="jdbc:derby://localhost:1527/itec833"/>
```

Appendix D: Miscellaneous Configuration Issues

- The AXIS2_HOME environment variable must be set properly.
- Our application depends on some of the libraries distributed with Apache Rampart, so Apache Rampart must be installed correctly before the axis2 web application is built.
- Our web service needs jdom-1.1.jar file, which is unfortunately missing from the axis2 distribution. We've found that including it as part of the '.aar' file does not work, so this file needs to be manually copied to '$AXIS2_HOME/lib' folder before the axis2 war file is built (by running the build script distributed with the standard archive).