# Distributed Systems 4 Project Report

Xiangyu Liu

A00279565@student.ait.ie

## Introduction

This project is a storage management system based on a RESTful style architecture, with Maven as the build tool and the use of Jersey, HSQLDB and Tomcat to implement the responses to HTTP requests for GET, PUT, POST and DELETE to the client. Table 1 illustrates the running environment for this project.

Table 1: Project Running Environment Table

|  |  |  |
| --- | --- | --- |
| Type | Name | Version |
| Operating System | macOS | Big Sur 11.3 |
| Database | HSQLDB | 2.3.4 |
| Server | Apache Tomcat | 9.0.37 |
| REST Framework | Jersey | 1.19.4 |
| Programming Langauge | Java | 1.8.0\_282 |
| Development IDE | IntelliJ IDEA | 2021.1 |

This project implements the use of GET, PUT, POST and DELETE requests to fetch, add, update and delete products in the storage management system, and uses XMLPullParser to parse the responses and output them to the GUI. The link to the presentation of the project is : <https://youtu.be/9PLRsaSkaV8> .

## Research

* 1. RESTful Web Services

REST defines a set of architectural principles by which Web services can be designed that focus on system resources, including how resource state is handled and transmitted over HTTP by a variety of clients written in different languages [2]. First and foremost, Web Services is a web interface designed to solve the problem of how to handle resources through the web.There are various design architectures for Web Services, such as the RESTful architecture and the RPC architecture. In a web application, everything should have a unique ID, represented by a unified concept of ID: the URI.URIs form a global namespace, and using URIs to identify your key resources means that they get a unique, global ID.REST requires developers to use HTTP methods explicitly and in a way that is consistent with the protocol definition. This basic REST design principle establishes a one-to-one mapping between create, read, update and delete operations and HTTP methods. According to this mapping:

* To retrieve a resource, the GET method should be used.
* To create a resource on the server, you should use the POST method.
* To change the status of a resource or to update it, the PUT method should be used.
* To delete a resource, the DELETE method should be used.
  1. Jersey

The Jersey RESTful Web Services Framework is the open source, production quality, framework for developing RESTful web services in Java that provides support for the JAX-RS API and acts as the JAX-RS (JSR 311 and JSR 339) reference implementation. the Jersey framework is more than just a JAX-RS reference implementation. Jersey provides its own API that extends the JAX-RS toolkit with other features and utilities to further simplify RESTful service and client development. jersey also exposes a number of extensions to the SPI so that developers can extend jersey to best suit their needs [2].

* 1. HSQLDB

HSQLDB (Super SQL Database) is a relational database management system written in Java which provides in-memory and table-based functionality. As a database, HSQLDB provides a small and fast database, offering both server mode and embedded mode. It has a JDBC driver and supports a larger subset of the SQL-92, SQL:2008, SQL:2011 and SQL:2016 standards. It provides a fast and small database.Additionally, it includes tools such as a minimal Web server, command line and GUI management tools [3].

## Requirements

Using Java API for RESTful Web Services (Jax-Rs):

1. Build a client application that sends all of the HTTP requests GET/PUT/POST/DELETE.
2. Build a server application using tomcat server, that responds to all of the HTTP requests GET/PUT/POST/DELETE".
3. The client application will parses the response using XMLPullParser and outputs to the GUI" + "A tomcat server that responds to all of the HTTP requests GET/PUT/POST/DELETE".
4. The data in the response will be taken from an HSQLDB database.

We can get the product data based on the product ID and the category name of the product. We can also update and delete the product by selecting it in the GUI, and we can also perform a purchase operation on the product by adding the name, category and quantity of the purchased product to the purchase table.

## Design and Implementation

This chapter focuses on the design of the project and the implementation of the functionality. It includes the interface of the project application, the presentation of the functionality and the corresponding source code.

* 1. Overall Project Structure

The project is built using Maven so that we do not have to manually import the jar packages ourselves. The overall project structure is illustrated in Figure 1.

文本

描述已自动生成

Figure 1: Project Structure Diagram

The four packages are dao, pojo, resource and view. dao is the code implementation of the CRUD methods for the data tables in the HSQLDB database. pojo contains the entity objects. resource contains the methods that use the JAX-RS annotations. view contains all the view classes for the project GUI. There are three modules in this project. The main page of the project is illustrated in Figure 2.

图形用户界面, 应用程序

描述已自动生成

Figure 2: Main Page Diagram

* 1. Project Database Table Design and Implementation

The project database is HSQLDB, which has two tables, the product table and the purchase table. The product table stores all product information, while the purchase table stores records of purchased items. The SQL script to create these two tables (omitting specific data) is shown in Figure 3.

文本

描述已自动生成

Figure 3: Database Table SQL Script

The initial data for the database product table are queried using the HSQLDB Database Manager as illustrated in Figure 4.

表格

描述已自动生成

Figure 4: Product Information Diagram

The initial data for the database purchase table are queried using the HSQLDB Database Manager as illustrated in Figure 5.

表格

描述已自动生成

Figure 5: Purchase Information Diagram

* 1. Search Module Design and Implementation

This section describes the design and implementation of the search module, whose main function is to send GET requests to fetch data from the HSQLDB database and render it into a JTable based on the ID of the product. The interface of this module is illustrated in Figure 6.

图形用户界面, 文本, 应用程序

描述已自动生成

Figure 6: Search Module Diagram

First we need to configure Jersey in web.xml and then add an annotation to the getProduct() method in the ProductResource class in the resource package (omitting the dao code display). The code is illustrated in Figure 3.

文本

描述已自动生成

Figure 7: getProduct() Method Core Code

We now start Tomcat and then request the relevant data by entering the correct URI and selecting the GET request on Postman. The data is illustrated in Figure 8.

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Figure 8: Postman XML Data Diagram

In the relevant view class, we use Apache's HttpClient package to build HTTP-aware client applications, which are implemented to send GET requests according to the relevant buttons. In the getProduct() method, we first need to get the number in the text box, then construct a URI and set the correct parameters. Then we create an HttpGet request and set its headers. Then we get an HttpClient and the client executes the request and gets the response entity from the response model. Finally the XML is parsed by the XmlPullParser to get the corresponding object and rendered into the JTable. The code is illustrated in Figure 9.

文本

描述已自动生成

Figure 9: getProduct() Method Core Code

* 1. Purchase Module Design and Implementation

This section describes the design and implementation of the purchase module, whose main function is to send POST requests to add products to the HSQLDB database. It can also send a PUT request to update the stock of an product. It can also send GET requests to query and render the purchase records into the JTable. As we have already covered the process of sending GET requests from the client in the previous section, we will not repeat it here. The POST request and the PUT request are described in detail. The interface of the module is shown in Figure 6.

图形用户界面, 文本, 应用程序

描述已自动生成

Figure 10: Purchase Module Diagram

For a POST request to add a new product. We need to create the addProduct() method in ProductResource. and use the @POST annotation. The code is illustrated in Figure 3.

文本

描述已自动生成

Figure 11: addProduct() Method Core Code

In the relevant view class, we first need to get the data in each text box, then instantiate a Product object and pass in the data using the corresponding set method. A new xml string is created based on the XmlRootElement and XmlType set in the Product. This is illustrated in Figure 12.

文本

描述已自动生成

Figure 12: String Xml Core Code

We then get such a string in XML format that we can wrap this string into an entity and pass this entity via the HttpPost.setEntity() method. Finally the client will send this POST request. Then the new product can be successfully added to the database. The code is illustrated in Figure 13. The interface for adding new items is illustrated in Figure 14.

文本

描述已自动生成

Figure 13:saveNewProduct() Method Core Code

图形用户界面, 表格

描述已自动生成

Figure 14: NewProduct Interface Diagram

The code implementation of the PUT request is very similar to that of the POST request; the PUT updates the resource, the corresponding updateProduct() method in the ProductResource class uses the @PUT annotation, and then the corresponding view class still gets the data in the modify field text box and instantiates a new object, still building An XML string is encapsulated into the entity and then passed in using the HttpPost.setEntity() method. Finally this PUT request is issued by the client and successfully updates the specified resource. The code is shown in Figure 15.

文本

描述已自动生成

Figure 15: updateProduct() Method Core Code

* 1. Stock Module Design and Implementation

This section describes the design and implementation of the stock module, whose main function is to send GET requests to retrieve different data from the HSQLDB by product category name. It can also send a PUT request to update all information about a product by selecting the product data in the JTable. It can also send a DELETE request to remove an product from the JTable. As we have already covered the process of sending GET, POST and PUT requests from the client in the previous sub-sections, we will not repeat it here. and focus on the DELETE request. The module's interface is shown in Figure 16.

表格

描述已自动生成

Figure 16: Stock Module Diagram

Regarding DELETE requests, we need to use the @DELETE annotation in the deleteProduct() method in the ProductResource class. The code is shown in Figure 17.

文本

描述已自动生成

Figure 17: deleteProduct() Method Core Code

The method in the relevant view class simply creates the appropriate URI and passes in the ID of the selected product before setting the header, and the client sends this HttpDelete request to successfully delete the specific product from the database.

The interface also has an Export button to export data from the JTable in the interface and store it in CSV format. Friendly message alerts pop up throughout the project when a number of operations are performed. Due to space constraints, a detailed demonstration of the project can be viewed at the link in Chapter 1.

## References

1. Rodriguez, A. (2008). Restful web services: The basics. IBM developerWorks, 33, 18.
2. Gulabani, S. (2014). *Developing RESTful web services with Jersey 2.0*. Packt Publishing Ltd.
3. Widianto, S. R., & Warmayudha, I. P. E. (2020). HSQL Database. *Jurnal Mantik*, *4*(3), 1717-1721.