ATHLONE INSTITUE OF TECHNOLOGY

Item Storage Management System

Distributed Systems – Assignment Report

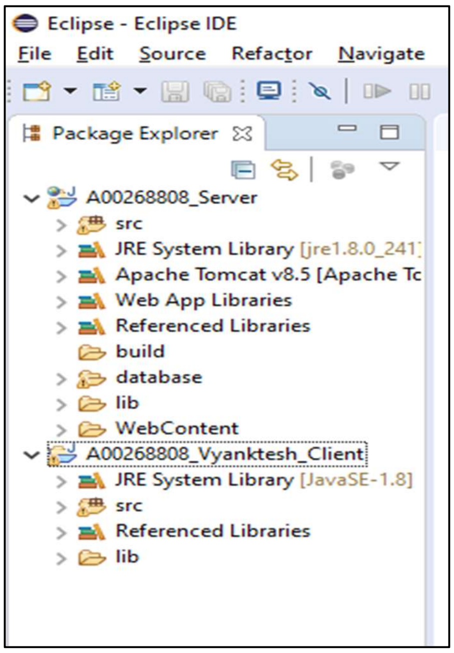
A00268808

Vyanktesh Chandurkar

CONTENTS

Overview

Item Storage Management System has main modules as Items Module which is used for storing the details of the the item stored like name of the customer, mobile number of the customer and cabinet number where the item is stored.



```
A00268808_Server
  src
  JRE System Library [jre1.8.0_241]
  Apache Tomcat v8.5 [Apache Tc
  Web App Libraries
  Referenced Libraries
  build
  database
  lib
  WebContent
A00268808_Vyanktesh_Client
  JRE System Library [JavaSE-1.8]
  src
  Referenced Libraries
  lib
```

The System is implemented using two applications:

One is the Server-Side Application that is named as A00268808_Server which has package which is server paths running through servlet in the web.XML file.

Second application is the Client-Side Application that is named as A00268808_Vyanktesh_Client, this application has the GUI that sends all the requests to the Servers and then displays all the received responses to the GUI. The project contains a ServerRequest package which contains ParseItem.java class, which is the class at the client side that processes all requests to the server and accepts the response. From this ParseItem Class the responses are then passed to the XMLPullParsers which convert the received XML Response to Plain Text. The XMLPullParser which Is named as ParseItem is used for parsing the response of the doParsestorage () method that is returning all the data present in the table.

Setting Up the Project

1. RUN APACHE TOMCAT SERVER (VERSION USED 8.5)

2. RUN managed_db ANT script in ANT (Present in the databa folder in the Server-Side application)

3. RUN Server-Side application on the Tomcat Server.

4. RUN CLIENT-SIDE APPLICATION (MainClass) as a Java Application.

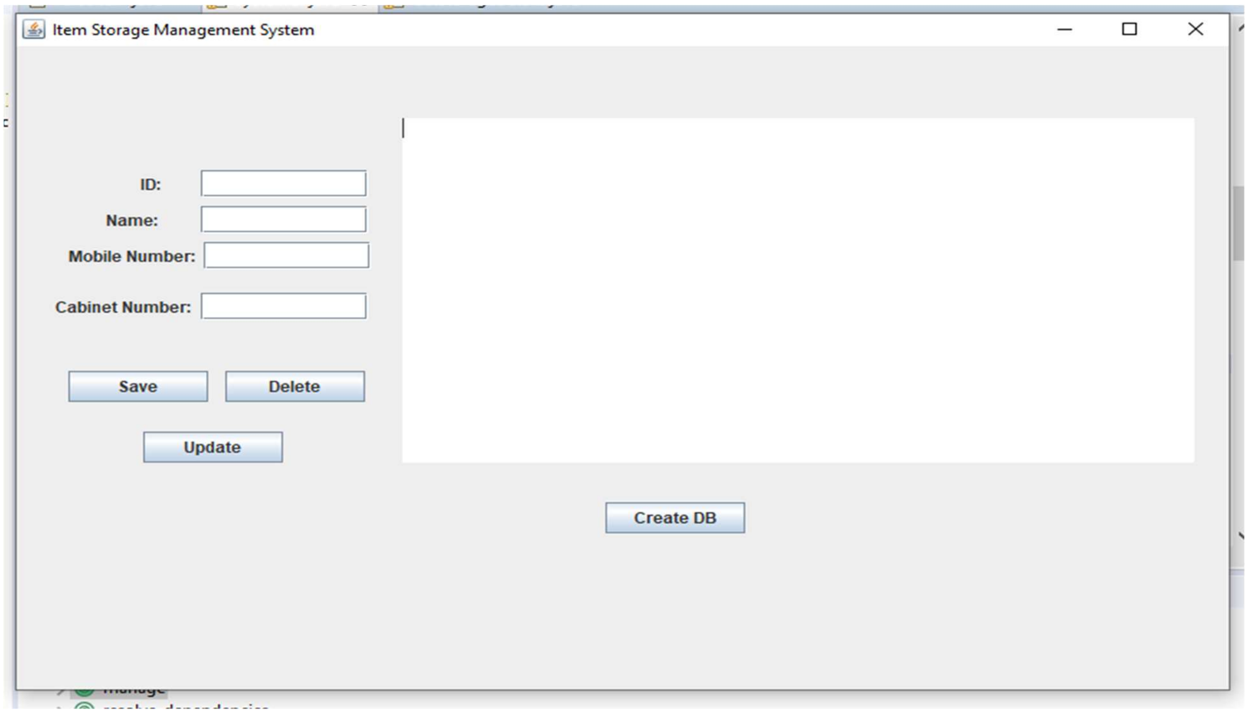
   If database table Items don't exist, table will be created when main class is runned.

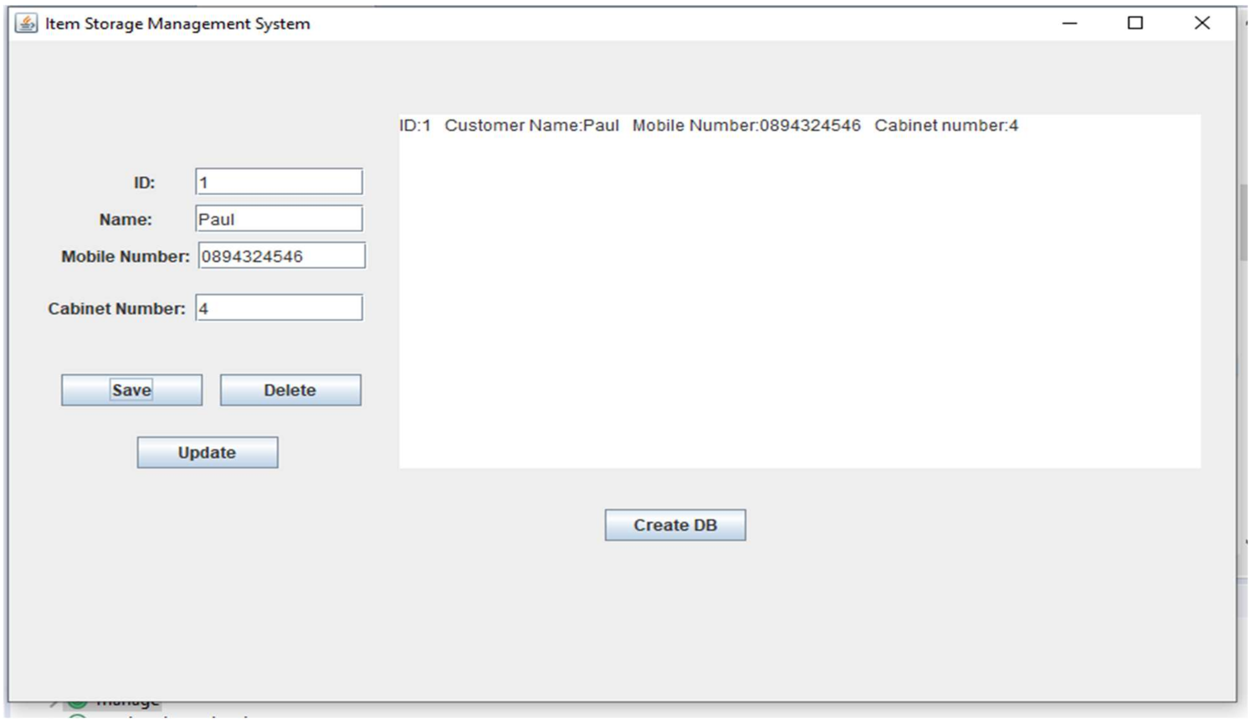
Main Functions

- Save – For Inserting the item details into Items Table and viewing Items from the Items table after inserting.
- Delete – For Deleting the item details from Items Table and viewing Items from the Items table after inserting.
- Create DB – For creating Items table.
- Update – For Updating the details of the items stored for example if cabinet is changed by the staff then the cabinet number will need to be updated in the items tables in the database.

Description

Build a client application that sends all of the HTTP requests GET/PUT/POST/DELETE.



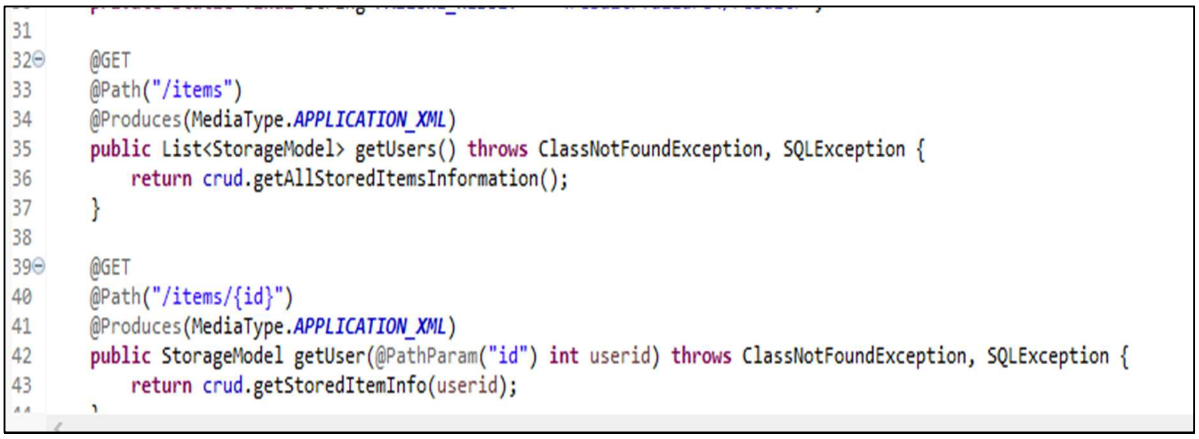


ID: 1
Name: Paul
Mobile Number: 0894324546
Cabinet Number: 4

```
ID:1   Customer Name:Paul   Mobile Number:0894324546   Cabinet number:4
```

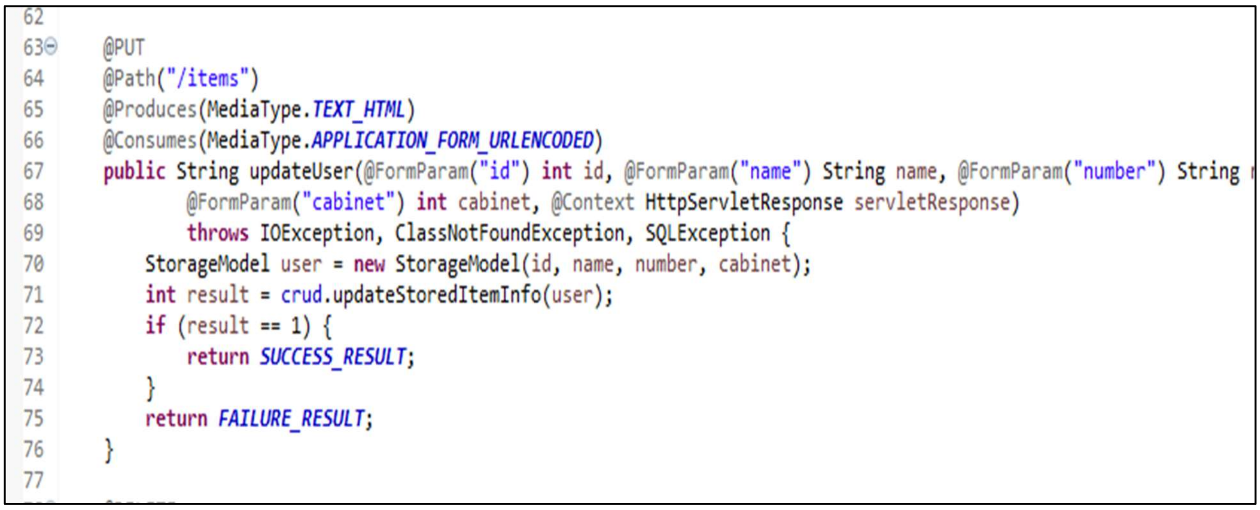
Build a server application using tomcat server, that responds to all of the HTTP requests GET/PUT/POST/DELETE"

GET: GET is used for returning the values from the database to the GUI, there are different use of the GET in the project. One GET is used for fetching all the values present in the database table. Second GET is used for fetching the values present at a specific ID, in this the ID is passed as a PathParam that comes in the URI. GET returns the response in the XML format to the client.



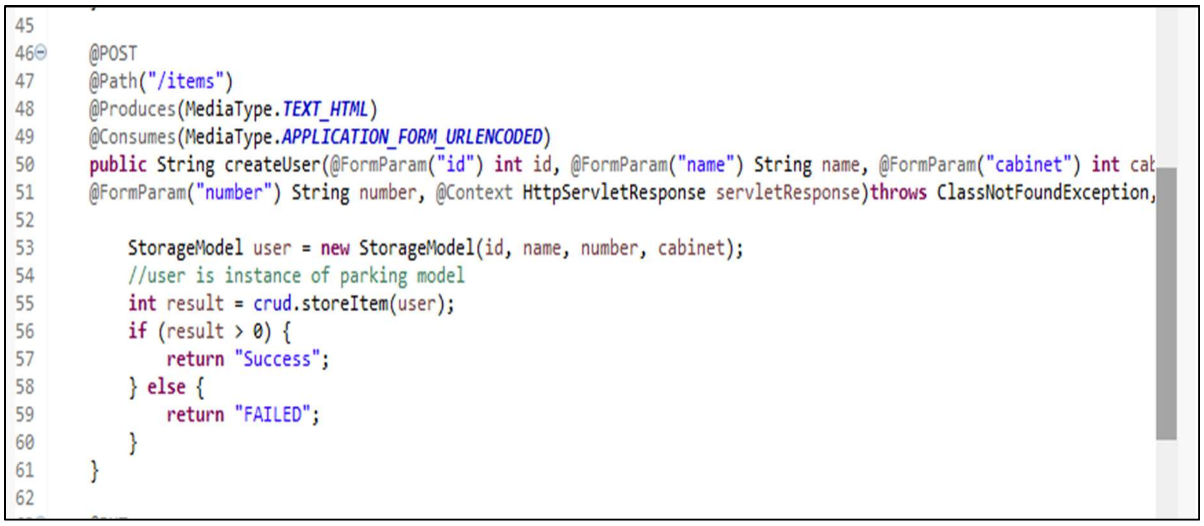
```
    @GET
    @Path("/items")
    @Produces(MediaType.APPLICATION_XML)
    public List<StorageModel> getUsers() throws ClassNotFoundException, SQLException {
        return crud.getAllStoredItemsInformation();
    }

    @GET
    @Path("/items/{id}")
    @Produces(MediaType.APPLICATION_XML)
    public StorageModel getUser(@PathParam("id") int userid) throws ClassNotFoundException, SQLException {
        return crud.getStoredItemInfo(userid);
    }
```

PUT: PUT is used for updating a specific row at the database table that is present at a specific ID. PUT accepts parameters as FormParams and this parameters at the client side are passed to the Server.



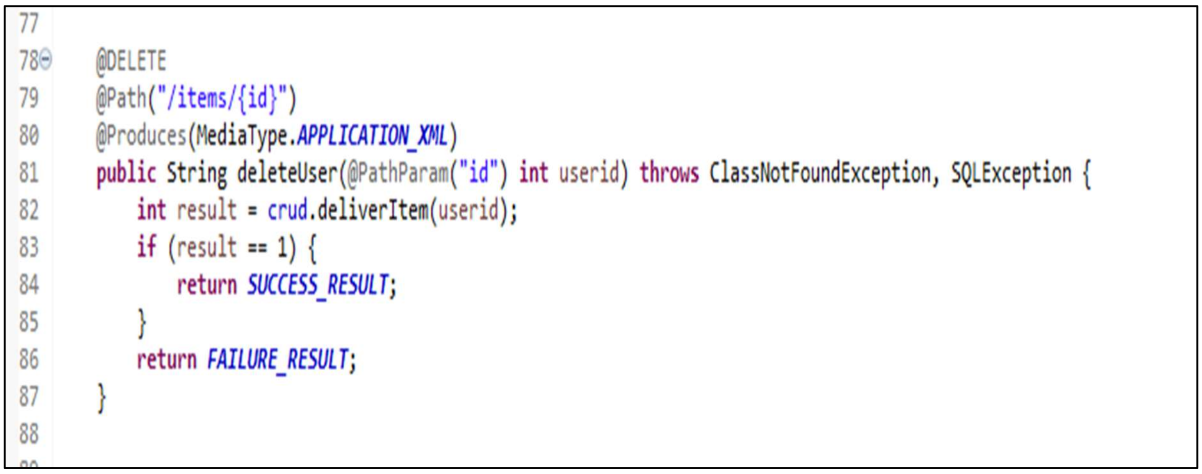
```
    @PUT
    @Path("/items")
    @Produces(MediaType.TEXT_HTML)
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public String updateUser(@FormParam("id") int id, @FormParam("name") String name, @FormParam("number") String 
            @FormParam("cabinet") int cabinet, @Context HttpServletResponse servletResponse)
            throws IOException, ClassNotFoundException, SQLException {
        StorageModel user = new StorageModel(id, name, number, cabinet);
        int result = crud.updateStoredItemInfo(user);
        if (result == 1) {
            return SUCCESS_RESULT;
        }
        return FAILURE_RESULT;
    }
```

POST: POST is used for inserting data in the database tables. POST at the server side accepts all the parameters as FormParams and then we cast it to Model class object by passing them to the Model class constructor and then this Model class object is passed to the DAO class which uses the getters of the Model class to extract the parameters and insert the data in the database table.



```
    @POST
    @Path("/items")
    @Produces(MediaType.TEXT_HTML)
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public String createUser(@FormParam("id") int id, @FormParam("name") String name, @FormParam("cabinet") int cat
    @FormParam("number") String number, @Context HttpServletResponse servletResponse)throws ClassNotFoundException,

        StorageModel user = new StorageModel(id, name, number, cabinet);
        //user is instance of parking model
        int result = crud.storeItem(user);
        if (result > 0) {
            return "Success";
        } else {
            return "FAILED";
        }
    }

```

DELETE: DELETE is used for deleting the data present at the specific row from the database table which is identified by the ID passed by the client side. DELETE accepts the ID as PathParam that is the id is passed with the URI from the client side to the server.



```

    @DELETE
    @Path("/items/{id}")
    @Produces(MediaType.APPLICATION_XML)
    public String deleteUser(@PathParam("id") int userid) throws ClassNotFoundException, SQLException {
        int result = crud.deliverItem(userid);
        if (result == 1) {
            return SUCCESS_RESULT;
        }
        return FAILURE_RESULT;
    }

```

The client application will parse the response using XMLPullParser and outputs to the GUI" + "A tomcat server that responds to all of the HTTP requests GET/PUT/POST/DELETE"

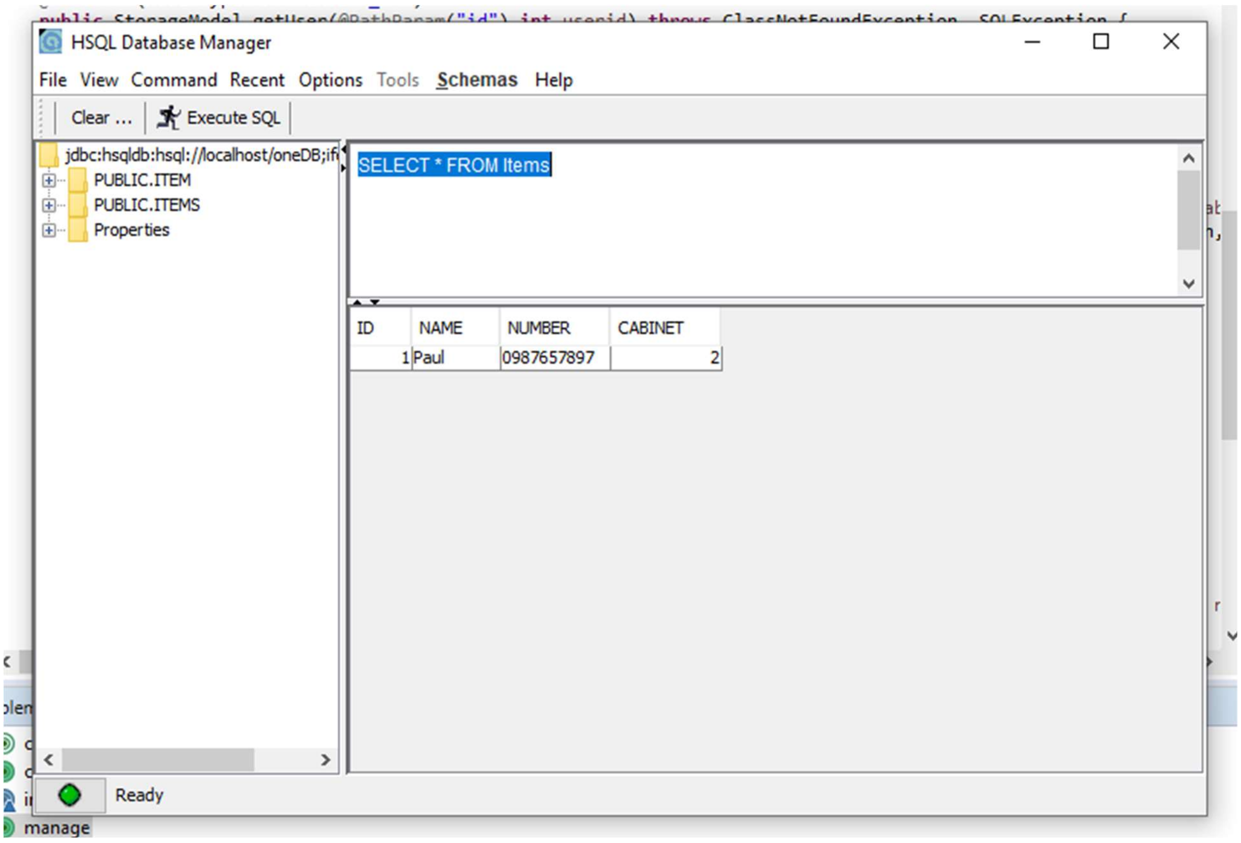
I have succesfully implemented the Client Side and Server-Side application and We can see the output in the GUI.

Example of GET: When we run Client side application, the client side passes a HTTP GET Request to the server which then accepts this request and the GET method at the server calls the getAllValues at the DAO class which returns all the values as an Arraylist, this Arraylist is then converted in XML and then this XML response is passed to the Client.

When Client receives this response, it passes the XML response to the XML Parser class which then decodes the data to normal text and then casts it to DefaultTableModel which then is send to the Products Report class which sets the returned DefaultTableModel to the Textarea present in the JFrame.

The data in the response will be taken from an HSQLDB database.

I have used HSQLDB as the data model for storing and accessing the data. The HSQLDB is directly connected to the Server-Side application which does all the operations on the database and just passes the response to the client side.


HSQL Database Manager
File View Command Recent Options Tools Schemas Help
Clear ...
Execute SQL
jdbc:hsqldb:hsql://localhost/oneDB;ifi
PUBLIC.ITEM
PUBLIC.ITEMS
Properties
Ready

```
SELECT * FROM Items
```

ID	NAME	NUMBER	CABINET
1	Paul	0987657897	2

SQL QUERY FOR CREATING THE TABLES:

ITEMS TABLE

CREATE TABLE Items (id INTEGER IDENTITY, name VARCHAR(30), number VARCHAR(30), cabinet INTEGER);