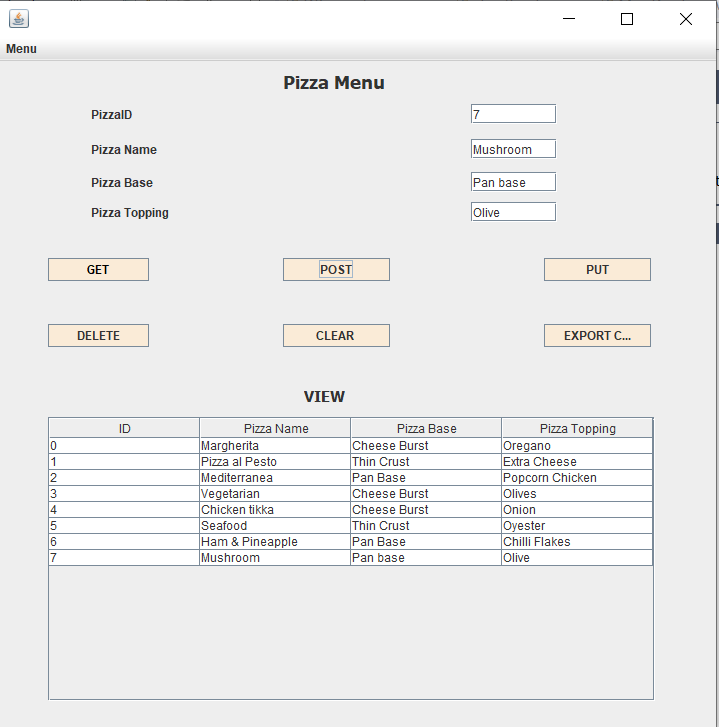
# Distributed Systems Assignment 2020

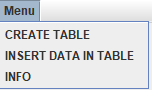
# DRISHTI SINGH – A00268741

**Project Overview**

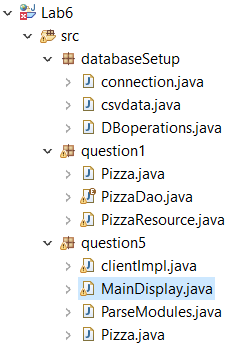
**Main Window**



**Menu Options**



**FILES**



This project holds a database for storing and accessing information about pizza menu and runs on a Tomcat 9.0 server. The information is stored across ONE HSQLDB table to hold an id (primary key), Name, Base, and Topping. Information is retrieved and processed using an XMLPullParser to identify it using tags. The server uses the JAX:RS api to access/display the data.

**Main Functions:**

**Get** - Return the information a single entity by entering a Id.

**Delete** - Enter an Id number and it is deleted from the system.

**Put** – Enter the current id of a pizza and the name that it will be changed to and the table is updated.

**Post** – Enter the details for a new pizza and create it on the server.

**Clear** – Removes all data from the table.

**Export CSV** - Prints all the data from table to an excel sheet (comma separated values).

**Insert data in Tables (menu bar option)** - Creates three entities automatically for the table.

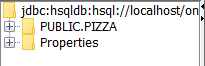
**Create Tables (menu bar option)** - Runs code to create the table.

**Info (menu bar option) -** Small popup box with student number and name.

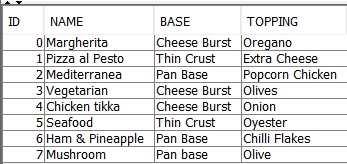
**1. Requirement**

"Multiple Tables and/or tables created using code at runtime plus code to select, insert, update and delete data"

**Completion**



**SELECT \* FROM Pizza**



**Creation SQL:**

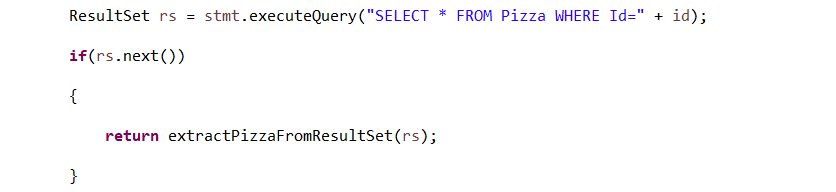
CREATE TABLE Pizza( Id INTEGER IDENTITY PRIMARY KEY, Name varchar(30) NOT NULL, Base varchar(30) NOT NULL, Topping varchar(30) NOT NULL);

**Steps for First Time Use:**

1. **Run Tomcat Server.**
2. **Run Ant.**
3. **Menu Bar -> Create Tables.**
4. **Menu Bar -> Fill Tables.**

The first time the project is run the user must click the Create Tables option of the menu bar. This will run the create objects init() function to create the two tables. Once the tables are set up this does not need to be run again. This is probably not the best way to create the tables as it requires the user to connect directly to the server but the feature seemed more important to include than to leave out altogether. Create is the only function that works like this, every other command uses Jax:Rs and the PizzaDao.

The select, update, delete and insert functions are called from the PizzaDao (data access object) files and connect to the HSQLDB server with a direct connection. A data access object acts like an interface between the client and the HSQLDB server, allowing for functions to be called without giving away too much information about the server. This is an example of the select command for getting a single Pizza by its Id:



As code is not required in this write up please see the PizzaDao java file for more implementations of SQL commands.

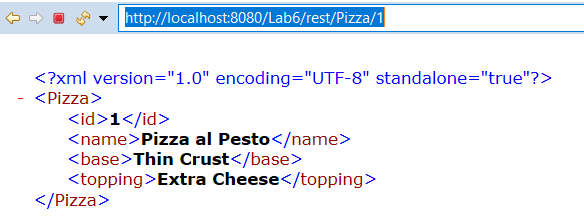
**2. Requirement**

"A Jax-Rs/Jersey client that sends all of the HTTP requests GET/PUT/POST/DELETE, parses the response XMLPullParser and outputs to the GUI" + "A tomcat server that responds to all of the HTTP requests GET/PUT/POST/DELETE"

**Completion**

**Get:** 2 different @GET JAX:RS commands implemented, one which simply returns all the pizza stored on the server and one for returning a specific Pizza by id. The single get works by supplying a pizza id as part of the path.

Example of a single pizza returned:



As part of the get command the XMLPullParser was used to separate the information returned by its tags which can be seen in the ParseModules file. When the information is returned it is pulled apart using a series of if statements to examine which tag of the XML the parser is currently in. E.g. as the parser iterates through the specific pizza it will meet the start of a tag such as <id> then it will set the text between that and the closing tag of </id> as the id for a temporary Pizza. The Pizza will be returned. When the </Pizza> tag is met as that is the end of information about that specific Pizza.

**Put:** The @put command is used to update the name for a pizza to a new name. It takes in the current id as a parameter and the new name that it will be changed to. These are parsed by adding them as BasicNameValuePairs to a nameValuePairs list and set as the URLEncodedFormEntity for the httpPut request.

The old and new names being read in as @FormPararm so they can be then sent as parameters to the PizzaDao:



**Post:** Takes in 4 parameters from the user for a new par and then passes to the PizzaResource object in a similar way to the PUT (I.e. with parameters) and then on the PizzaDao to create the object in SQL.

**DELETE:** There are two versions of the @DELETE command implemented, one that takes a id provided by the user as a parameter and one which just deletes everything in the table. The single delete works very similarly to the @GET, except it calls a delete command in sql based on the name provided. When the Delete all command is called it will also reset the auto incrementation of the pizza id back to 0.

Every time a command is called other than the single get then the getAllPizza() function is called to display the newly updated tables on the main text area screen. This was done to make sure the user gets some feedback when they make a change to the table.

**3. Requirement**

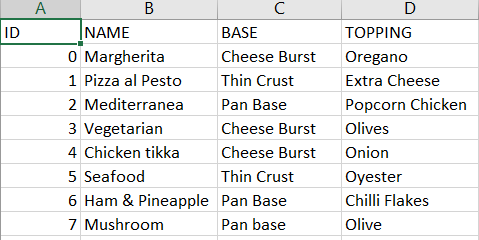
"GUI that handles the GET/PUT/POST/DELETE actions"

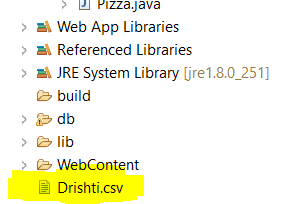
**Completion**

Most screenshots of the GUI have already been included but there are some extra features that will be highlighted here.

**Print to Excel**

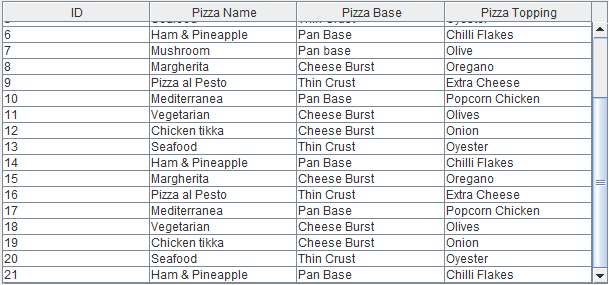
All data from files printed to an excel file. This works the same way as the get all command to return all the pizza and their info and uses the XMLPullParser to extract the data to be printed. This was added as an additional feature to increase interactivity.





**Scroll Bar**

If a lot of bars are added to the main text display, then a scroll bar will appear to let the user scroll through their entries.



SCREENCAST LINK <https://youtu.be/PyV5gv3t3V0>