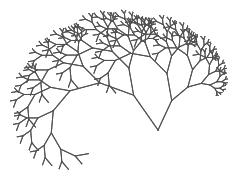
**MOVIE APPLICATION REPORT**

**DISTRIBUTED SYSTEM – ASSIGNMENT 2019**



**PRABHANSHU PINCHA**

A00258748

Software Engineering (Batch - A)

**INTRODUCTION**

The main objective to build this project is to implement Client application that sends all the HTTP request like – GET, PUT, POST, DELETE. And Server application using tomcat server, that respond to all the HTTP requests GET, PUT, POST, DELETE. The application parses the response in XML format and give output in GUI. The data in the response is taken from an HSQLDB database.

For implementing all this functionality, I have taken an example of Movie Application, implemented a server, client and database which sends and handles all the request.

**MAIN FUNCTIONALITY**

POST – Enter the details and it will create it on the server (insert) and database table will get updated by this request.

PUT – Enter the details and it will get change in database and table will be updated.

GET – Return the information by entering the id.

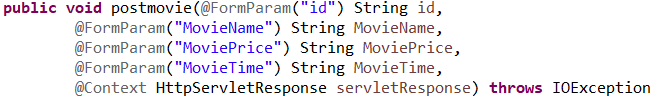
GET ALL – Return the information with all the attributes (id, name, price, time).

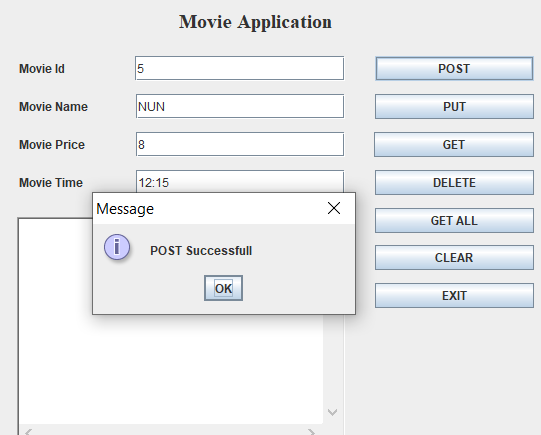
DELTE – Delete the table from database by entering the id.

**REQUIREMENT**

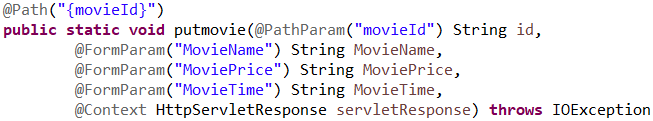
Build a client application that sends all the HTTP requests GET/PUT/POST/DELETE. And the client application will parses the response using XMLPullParser and outputs to the GUI" + "A tomcat server that responds to all of the HTTP requests

***POST* -** In this method application is taking 4 parameters (id, name, price, time) now these parameters is parsed by BasicNameValuePair list and set as the URL Encoded Form Entity for the HTTP POST request. The data being read in as @FormParam so now they can be sent as parameters to the file MovieDao.





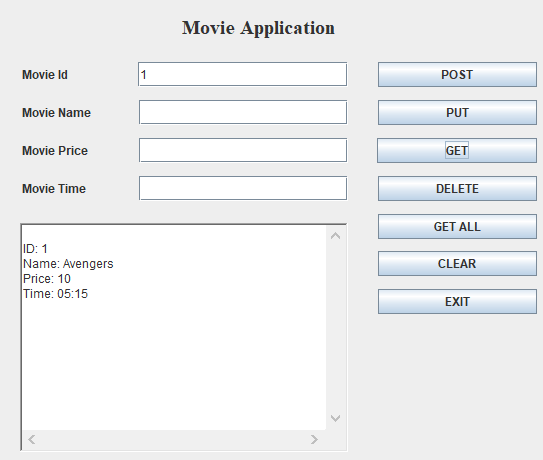
***PUT*** – In this method application is updating the details of the movie. Updating details using the movie id as you can see the path is already set to **movieId**. These are parsed by adding them as BasicNameValuePairs to a nameValuePairs list and set as the URL Encoded Form Entity for the HTTP PUT request. The old and new details being read in as @FormPararm so they can be then sent as parameters to the MovieDao.



***GET*** – In this @GET method application is returning details about the movie by referring id as you can see the PATH is set to movieId. As a part of @GET the XMLPullParser is used to separate the information returned by its tag which can be seen in the Parse Movie. When the information is returned it is pulled apart using a series of if statements to examine which tag of XML the parser is currently in. As the parser iterates through the specific movie it will meet the start of a tag such as <name> then it will set the text between that and the closing tag of </name> as the name for a temporary movie. The movie will be returned. When the </movie> tag is met as that is the end of information about that specific movie.

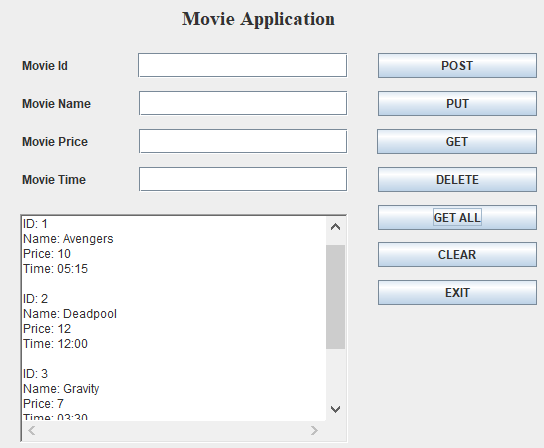


The information returned is the display in the single get:



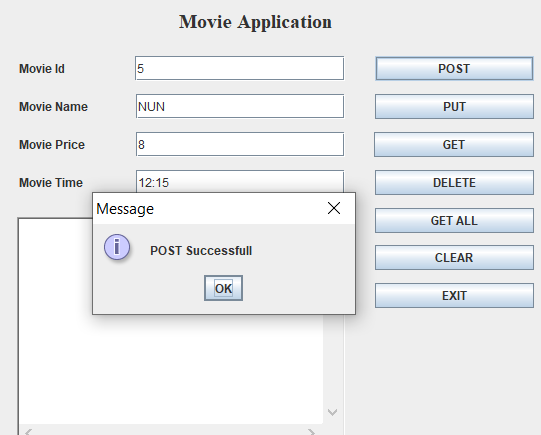
***GET ALL*** – @GET GET ALL works similarly but uses the ParseMovies file. This returns a list of all the movies and is used to in the same way that the single get was.





***DELETE*** – @DELETE command implemented, it takes a movie id by the user as parameter and delete. This works similarly to the @GET, except it calls a delete command in the sql based on the movie id.





**REQUIREMENT**

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

stmt.executeUpdate("CREATE TABLE IF NOT EXISTS MOVIE(id INTEGER IDENTITY, MovieName VARCHAR(32) NOT NULL, MoviePrice VARCHAR(32) NOT NULL, MovieTime VARCHAR(32) NOT NULL)");

