Source code Documentation on Java spring boot application.

Project Abstract and Problem Statement:

The Application that we have developed is used to find the different stocks based on user selection. Also user can insert the new stock item's into the system**.** The user can also update their stock data.

Project Source files (Packages and it's files) :

com.springrest.springrest Package : This is the main package which had main java file which will run our Application. It contains one class file**.**

1) **SpringrestApplication.java :** This is Java file which is used to run our Application

com.springrest.springrest.controller : This Package is having only one java file and this file is used to control the entire flow.

1) **MyController.java** : This is the Controller file , Which is used to control the whole flow , where we can maintain and our operations would be handled.

com.springrest.springrest.Db : This Packed is usually created to manage DB related activities**.**

1) **Dao.java :** This java file is used to manages DB related Activities .

com.springrest.springrest.services : This Package consists of java files which contains classes and interfaces which are used to perform activities that are requested by controller

1) **stockService.java :** It is java file which contains interface where all the required method that were required by controller were defined**.**

2) **stockDetails.java :** It is a java file which implemented all the abstract method that were defined in the stockService.java interface. In this class I have handled all stock related coding

3) **UserDetails.java :** It is a java file which contains an interface which used to define user related activities

4) **userDetailsClass.java :** It is java file which implemented UserDetails.java interface and performed all the required activities

com.springrest.springrest.stock : In this package , I have defined entity related Data which had one java file which were related to stocks.

1) **Stock.java :** It consists of entity class which is used to manage stock data.

Code Explanation :

File name : SpringrestApplication.java

```
package com.springrest.springrest;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication

@SpringBootApplication

public class SpringrestApplication {

	public static void main(String[] args) {

		SpringApplication.run(SpringrestApplication.class, args);

	}

}
```

Explanation : This is the Main class of our SpringBootApplication which is used to run our project.

Here we have imported spring boot framework and we have defined a class and in it's main method we have called a run method which is used to run the project.

File name : Dao.java

```
package com.springrest.springrest.Db;

import org.springframework.data.jpa.repository.JpaRepository;

import com.springrest.springrest.stock.Stock;

public interface Dao extends JpaRepository <Stock , Long> {

}
```

Explanation : This is the interface which is used to Manage DB related Activities.

Here we have imported JpaRepository and we have defined a interface with two parameters (Entity class , Entity class table primary key data type) and it extended JpaRepository interface by which it get inherited some default methods.

Controller Class :

File name : myController.java

Package and imported frameworks and other classes.

```
package com.springrest.springrest.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.springrest.springrest.services.stockService;
import com.springrest.springrest.stock.Stock;
import java.sql.SQLException;
import java.util.*;

@RestController  // Representational State Transfer used to write Rest API's

public class myController {
 @Autowired  // For creating objects by it's own
 private stockService stockservice;

 /**
  * interface and object where Service was defined.
  */

 @GetMapping("/home")
 public String home()
 {
  return "Welcome to BYSOS";
 }
```

Method :

```
@GetMapping(path = "/stocks/{id}")
 public Stock getStockByCategoryId(@PathVariable String id)
 {
  return this.stockservice.getStockByCategoryId(Long.parseLong(id));
 }
```

Explanation:

```
/**
 * @GetMapping is used to retrive the data from dataBase
 * path = localhost:8080/"/stocks/{id} ,  method : GET
 *  stocks/{id}-- {} is used to define dynamic variable id is the dynamic variable here ex : stocks/101
 * getStocks();
 * This method  is used display the User required stock item by category id .
```

```
 * this method will call  interface to get the required stock item by matching with user input and in return method will return a stock that user has asked for.
 * if categoryid mentioned in the path was not available , It will return Null.
 */
```

Method :

```
@GetMapping(path = "/categoryid/{categoryid}/login/{loginid}")
 public Stock getStockByCategoryId(@PathVariable String categoryid, @PathVariable String loginid) throws NumberFormatException, SQLException {
  return this.stockservice.getStockByCategoryIdByValidatingUser(Long.parseLong(categoryid), loginid);
 }
```

Explanation:

```
 /**
  * @GetMapping is used to retrive the data from dataBase
  * path = localhost:8080//categoryid/{categoryid}/login/{loginid} ,  method : GET
  * /categoryid/{categoryid}/login/{loginid} -- {} is used to define dynamic variable , ex : categoryid/101/login/anil
  * getStocks();
  * This method  is used display the User required stock item by category id and by  validating the user.
  * this method will call  interface to get user required stock and in return method will return a list of stock that user has asked for.
  * if user mentioned in the path was not available , it will return Null also applicable for Categoryid
  */
```

Method:

```
 @GetMapping(path = "/stocks")
 public List < Stock > getStocks() {
  return this.stockservice.getStocks();
 }
```

Explanation:

```
 /**
  * @GetMapping is used to retrive the data from dataBase
  * path = localhost:8080/stocks , method : GET
  * getStocks();
  * This method  is used display all the stored stocks
  * this method will call  interface to get all the stocks and in return method will return a list of stocks
  *
  */
```

Method :

```
@PostMapping(path = "/stocks", consumes = "application/json")
 public Stock addStocks(@RequestBody Stock stock) {
```

```
   return this.stockservice.addStock(stock);
  }
```

Explanation:

```
  /**
   * @PostMapping is used to insert the data into the dataBase
   * path = localhost:8080/stocks , Method : POST
   * consumes defines the type of input that we are providing , here in our case i am sending json
   * addStocks();
   * This method  is used insert a new entry into the database
   * this method will call  interface to insert data into the dable in return method will return the item that we have
inserted.
   */
```

Method :

```
  @PutMapping(path = "/stocks", consumes = "application/json")
  public ResponseEntity <HttpStatus>  updateStocks(@RequestBody Stock stock) {
            try
            {
                      this.stockservice.updateStocks(stock);
                      return new ResponseEntity<>(HttpStatus.ACCEPTED);
            }
            catch (Exception e) {
                      return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);

            }
```

Explanation:

```
            /**
             * @PutMapping is used to update the data in the dataBase
             * path = localhost:8080/stocks , method : PUT
             * consumes defines the type of input that we are providing , here in our case i am sending json
             * updateStocks();
             * This method  is used update a new  in the database , if that entry is not present then it will insert that
into the table
             * this method will call  interface to update data in the table.
             */
  }
}
```

End of myController.java

Interfaces :

File name : stockService.java

```
package com.springrest.springrest.services;
import java.sql.SQLException;
import java.util.List;
import com.springrest.springrest.stock.Stock;

public interface stockService {
	public Stock getStockByCategoryId(long CategoryId);
	public Stock getStockByCategoryIdByValidatingUser(long CategoryId , String id) throws SQLException;
	public Stock addStock(Stock stock);
	public List<Stock> getStocks();
	public void updateStocks(Stock stock);

}
```

End of stockService.java

Explanation :

Here I have defined an interface stockService for to achieve abstraction.
These interface contains five abstract methods for adding , updating , retrieving the stocks which were further implemented by child class.

File name : UserDetails.java

```
package com.springrest.springrest.services;
import java.sql.SQLException;

public interface UserDetails

{
	public  boolean validateUser (String UserName) throws SQLException;

}
```

Explanation :

Here I have defined an interface UserDetails for to achieve abstraction.
These interface contains one abstract method for validating the user by checking it from DB, Which were further implemented by child class.

End of userDetails.java

Service Class :

```
package com.springrest.springrest.services;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.springrest.springrest.Db.Dao;
import com.springrest.springrest.stock.Stock;
import java.util.List;

@Service

/**
 * @service is used in Defining this is the service class
 * */

public class stocksDetails implements stockService {

  @Autowired

  private Dao dao;  // Dao object used to connect with JPA
```

Method :

```
  public List < Stock > getStocks()
  {
    return dao.findAll();
  }
```

Explanation :

```
/**
 * getStocks() will take no parameters and it is used to return all the stock items from the stock table
 * it can be achieved by findAll() method.
 * this method will return all the stocks in the form of list
 */
```

Method :

```
@Override
  public Stock addStock(Stock stock)
  {
    dao.save(stock);
    return stock;
  }
```

Explanation:

```
  /**
```

```
  * addStock() will take stock object as it's parameters and it is used to insert the stock item into the stock table
  * we can insert an item into  a table by save() method.
  * this method will return inserted stocks item to get it checked by user.
  */
```

Method :

```
public Stock getStockByCategoryId(long CategoryId)
 {
  Stock tempstock = null;
  List < Stock > str = dao.findAll();
  for (Stock s: str) {
   if (s.getCategoryId() == CategoryId)
   {
    System.out.println("Valid Category Id , displaying the Data");
    return s;
    }}
  System.out.println("Invalid Category Id , No Data to be displayed");
  return tempstock;
 }
```

Explanation:

```
 /**
  * getStockByCategoryId() this method will take category id as it's parameter and it is used to find stock item by
it's categoryid
  * If there is any entries related to that category id then  it will return the stock data  , else it will Null
  * to check for our required stock item , i am iterating all the stock items , once we got out required item then
program will terminate
  * it will return stock item, if no matches then it will return Null
  */
```

Method:

```
@Override
 public Stock getStockByCategoryIdByValidatingUser(long CategoryId, String id)
 {
  userDetailsClass Userdetails = new userDetailsClass();
  boolean flag = false;
  Stock tempstock = null;
  try {
   flag = Userdetails.validateUser(id); // calling method to check user is valid or not
  } catch (Exception e) {}
  if (flag) {
   System.out.println("Valid user...Checking for Stock Data");
   tempstock = getStockByCategoryId(CategoryId);  // calling method to get Stock By CategoryId
  } else {
   System.out.println("Invalid User.., Please login with valid user credentials");
  }
  return tempstock;
 }
```

Explanation :

```
 /**
  * getStockByCategoryIdByValidatingUser() this method will take category id , userid as it's parameter
  * This method is used to validate the user , if user is invalid then program will return Null else it will check for the stock item.
  * if the user is invalid then it program will return NULL
  * Once User got validated , If there is any entries related to that category id then it will return the stock data,else it will Null
  * to check for our required stock item , i am iterating all the stock items , once we got out required item then program will terminate
  * it will return stock item, if no matches then it will return Null
  */
```

Method :

```
 @Override
 public void updateStocks(Stock stock)
 {
  dao.save(stock);
 }
```

Explanation :

```
 /**
  * updateStocks() will take stock object as it's parameters and it is used to update the stock item in the stock table
  * we can update an item in the table by save() method , if there are no item related to primary key then it will insert into table.
  * this method will return updated stock item to get it checked by user.
  */
}
```

End of stockDetails.java

Filename : validateUser.java

```
package com.springrest.springrest.services;
import java.sql.*;
public class userDetailsClass implements UserDetails {
 public boolean validateUser(String UserName) throws SQLException {

  /**
   * DataBase Credentials to login into table , Another way of logging into DB
   */

   final String URL = "jdbc:mysql://localhost:3306/stocks";
   final String userid = "anil";
   final String Password = "anilChow@6";
   String query = "SELECT * FROM USERS WHERE USERID = '" + UserName + "';";


   try (Connection conn = DriverManager.getConnection(URL, userid, Password); Statement stmt =
conn.createStatement(); ResultSet rs = stmt.executeQuery(query);)
   {
    if (rs.next())
    {
     return true;
    }
   }
   catch (Exception e) {}
   return false;
 }
```

Explanation : Try block will connect to the DB if connection is valid then if will allow us to perform other activities else it will throw exception.

```
 /**
  * validateUser() this method will take user id as input and it is declaring with SQLExceptions as it is dealing with
DB Connection
  * The motive of this method is to check the user is maintained in the DB or not
  * if the user is maintained then it will return true else it will return false
  *
  */
```

End of validateUser.java

Entity Class :

```
package com.springrest.springrest.stock;
import javax.persistence.Entity;
import javax.persistence.Id;
@Entity

/**
 * @entity is like defining this class is the entity class
 * @author PokaVenkataAnilKumar
 *
 */
public class Stock {

@Id
private long CategoryId;
/**
 * @ID is used to define the variable that is coming after that is a primary key of that table
 */
private String StockName;
private double Price;

public Stock(long categoryId, String stockName, double price) {
		super();
		CategoryId = categoryId;
		StockName = stockName;
		Price = price;
}

/**
 * Parametirized constructor
 */

public Stock() {
		super();
}

/**
 * Default constructor
 */

public long getCategoryId() {
		return CategoryId;
}
public void setCategoryId(long categoryId) {
		CategoryId = categoryId;
}
public String getStockName() {
```

```
		return StockName;
	}
	public void setStockName(String stockName) {
		StockName = stockName;
	}
	public double getPrice() {
		return Price;
	}
	public void setPrice(double price) {
		Price = price;
	}

	/**
	 * setter and getter for all the variable used to retrieve or update the data.
	 */

	@Override
	public String toString() {
		return "Stock [CategoryId=" + CategoryId + ", StockName=" + StockName + ", Price="
				+ Price + ", getCategoryId()=" + getCategoryId() + ", getStockName()=" + getStockName()
	+ ", getPrice()=" + getPrice() + ", getClass()=" + getClass()
				+ ", hashCode()=" + hashCode() + ", toString()=" + super.toString() + "]";
	}

	}
```

End of Stock.java

Source Files :


SpringrestApplication.java


myController.java


Dao.java

Service files


stocksDetails.java


stockService.java


UserDetails.java


userDetailsClass.java

Entity files


Stock.java


Users.java

Other resource files


application.properties


pom.xml

Tools Used :

STS -- To develop source code

MySQL – It is used to store the Data (Local Machine)

Post Man – It is used to test the API’s

Test Cases :

Testcases.pdf

Prepared By :

P.v Anil Kumar

9877449284