ONLINE MARKETPLACE

Assignment 1

Samira Khorshidi

TABLE OF CONTENTS

ASSIGNMENT
DISCUSSION

The purpose of this practice is designing Domain Model of online Marketplace using Java RMI and MVC design pattern.

JAVA RMI

RMI stands for Remote Method Invocation. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

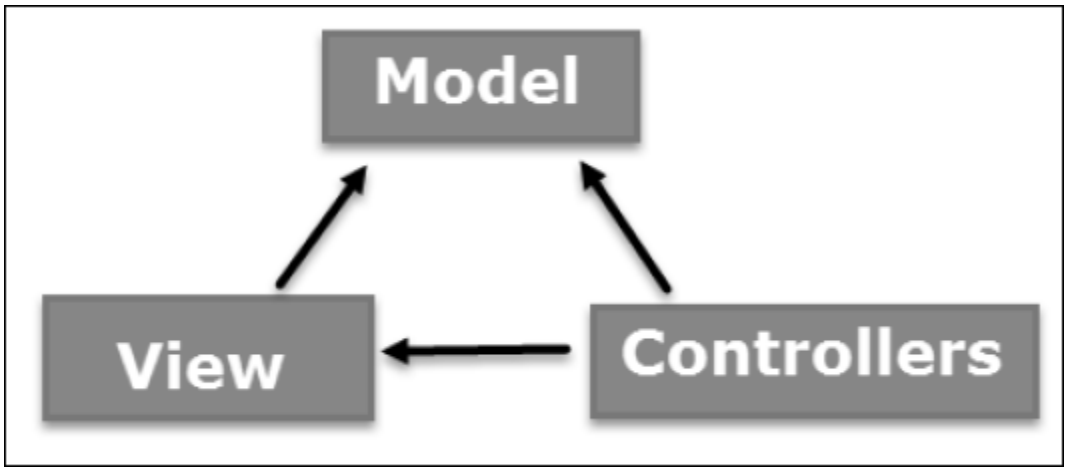
RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package `java.rmi`.

MVC PATTERN

The **Model-View-Controller (MVC)** is an architectural pattern that separates an application into three main logical components: the **model**, the view, and the controller.

Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

Generally, MVC pattern can be represented as following figure:



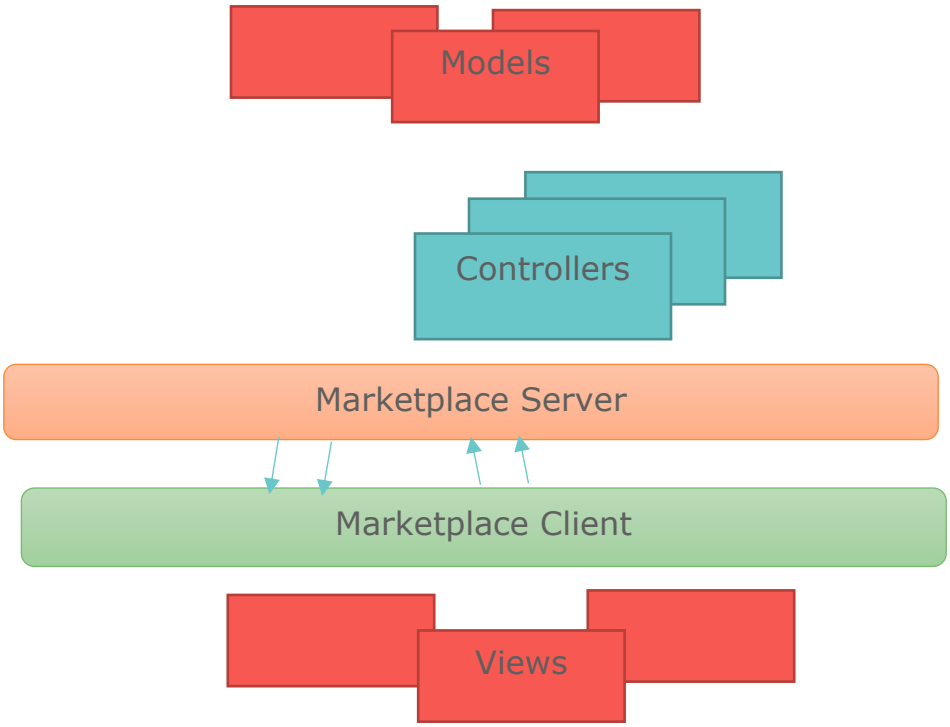
Which The Model component corresponds to the data-related, view corresponds to the User Interface and Controller acts as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.

USING MVC PATTERN IN RMI

In order to implement a distributed system using RMI and MVC pattern, we decided to emphasize on the nature of Model component in MVC and its relationship with Controller components.

On the other hand, RMI let us to call remote methods, so we come up with an architecture that prohibit View component in MVC to have access to the Model directly and keep our data safe.

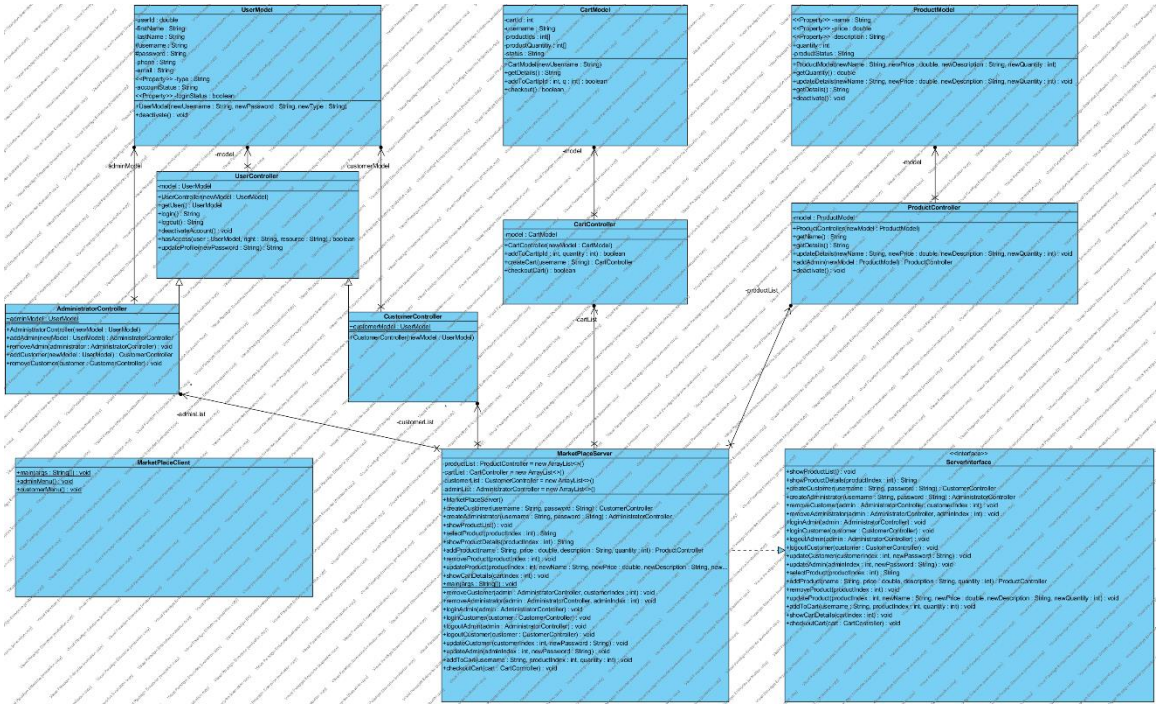
Thus, using Java RMI, we defined a Interface between View and Controllers. The general architecture of our system is as following figure:


Models
Controllers
Marketplace Server
Marketplace Client
Views

FIGURE,
SAMPLE CODES

In following we provide you a Class Diagram of our system along with some sample codes.

UML DIAGRAM



The original UML diagram has been provided in Documentation directory. As you see, in our proposed architecture, Model Components and Controller Components are in server side to obtain high cohesion system design and keep business logics and data separately related to view and Client component of our system. In this system, business logics and models can change frequently while view components can remain safe in term of affected by these changes.

SAMPLE CODES

In following we provide a sample code of our models, controllers and server Interface to the client. Models are used by Controllers and Server Interface acts as an Interface between Clients which are views and Controllers:

- *Model:*

```
package onlineMarket;

public class UserModel {

	private double userId;
	private String firstName;
	private String lastName;
	protected String username;
	protected String password;
	private String phone;
	private String email;
	private String type; // admin, customer
	private String accountStatus; // active, deactive
	private boolean loginStatus;

	public UserModel(String newUsername, String newPassword, String newType)
{
		username = newPassword;
		password = newUsername;
		type = newType;
		loginStatus = true;
		accountStatus = "active";
	}

	/**
	 * Responsible to login and logout the user
	 *
	 * @param status
	 */
	public void setLoginStatus(boolean status) {
		loginStatus = status;
	}
```

- *Controller:*

```
public class UserController {
	private UserModel model;

	// Model-View Glue
	public UserController(UserModel newModel) {
		this.model = newModel;
	}

	public UserModel getUser() {
		return model;
	}

	/**
	 * Responsible for login user
	 *
	 * @return user's type: admin or customer
	 */
	public String login() {
		model.setLoginStatus(true);
		return model.getType();
	}

	/**
	 * Responsible for logout the User
	 *
	 * @return Success message
	 */
	public String logout() {
		model.setLoginStatus(false);
		return "Logout successfully";
	}
```

- *Remote Method:*

```
public synchronized void loginCustomer(CustomerController customer) throws
RemoteException {
		System.out.println("Customer Login");
		customer.login();
	}
```

SAMPLE RUNS

In this section you'll see screenshots of sample runs. The system is minimal, so there is no GUI, instead every interaction happen using command prompt.

SCREENSHOTS

DISCUSSION

The proposed system has not complete functionalities, it means most of the system responses are using mockup data. Also, the current system need some modification to support multiple client interaction which will be implemented in the future. Thus, this system is a prototype of our online marketplace.

CONCLUTION
REFERENCES

CONCLUTION

In conclusions, as far as I see, Model-View-Controller (MVC) is a strong design pattern that can be used in application development, but using this design pattern and RMI can be a very confusing if you don't understand which component should goes where.

Moreover, Model-View-Controller is essentially a design pattern that separates the different aspects of a piece of Software and this separation promotes code reusability and a more structured application architecture.

REFERENCES

- *https://www.tutorialspoint.com/java_rmi/index.htm*
- *https://struts.apache.org/*
- *https://www.tutorialspoint.com/mvc_framework/index.htm*
- *https://www.tutorialspoint.com/design_pattern/index.htm*
- *https://stackoverflow.com/questions/35132852/java-rmi-mvc-pattern*

- *https://docs.oracle.com/javase/tutorial/java/concepts/package.html*
- *https://www.tutorialspoint.com/uml/*