
ONLINE MARKETPLACE APPLICATION

ASSIGNMENT-3

PROJECT REPORT

JAVA ANNOTATIONS, PROXY PATTERN, AUTHORIZATION PATTERN
REFLECTION, ROLE BASED ACCESS CONTROL

UNDER GUIDANCE OF

Dr. RYAN RYBARCZYK

BY

ADITHYA MORAMPUDI

Table of Contents

ASSIGNMENT 3 REPORT	3
Assignment#2 Feedback	3
Introduction	3
Application Flow	3
Software Design	3
Authorization Pattern	3
Proxy Pattern	4
Java Annotations	5
Reflection Pattern.....	5
RBAC	5
Class Diagram.....	6
Newly Created Classes.....	6
Sample Runs	7
Conclusion	9
References	9

ASSIGNMENT 3 REPORT

Assignment#2 Feedback

- No changes needed in Assignment#2

Introduction

In this Assignment, we are asked to implement Role based access control in the previously implemented partial Login functionality built using java RMI this assignment is done using java Annotations, Proxy, Authorization pattern and Reflection to get the Annotations on the methods during runtime. All these patterns are implemented on the server side of the application.

Application Flow

On top of the previously built application using Java RMI, we have built the Partial login process in the previous application, in this assignment I have implemented RBAC using java Annotations, to protect the application from unauthorized access to the methods which are not supposed to be accessed by the users who are unauthorized to access. For the login part, when the application starts, it first renders a page called Entry, this is the entry point to the application, as soon as the user selects an option in the Entry point, the request is sent to the Front controller pattern for Authorization or to give access rights to the user by checking whether he is a legitimate user or an intruder. Once the Front controller verifies that the user is a legitimate user, it then creates a session object based on the user role, if the user is an Admin, it builds the session of an Admin, if the user is a Customer, it builds the session of a customer. By getting these session objects after authentication, we can implement the RBAC using java annotations. Once the application gets the session related to the user. Whenever a user tries to access a method from the server, we use the reflection to find out if there is any annotation present on the method which the user is trying to access and it also checks the role of the user and based on these, it builds a proxy interface which can be accessible by the specific user role. This way our application is secure and specific functionality can only be accessed by specific users with specific roles.

Software Design

Authorization Pattern

Authorization Pattern helps us in leveraging the access rights in our application, The application which we are designing should limit its functionality to some users, if it allows everyone in the universe to access all the functionality then it is of little to no use. This is where the authorization pattern comes into picture. Authorization pattern helps in Assigning access rights to each client that can send requests to the security-sensitive subsystem and check these rights before executing

any requests on the subsystem, we can extend the application roles to any number using this pattern as new roles come into existence, without the need to refactor the entire application.

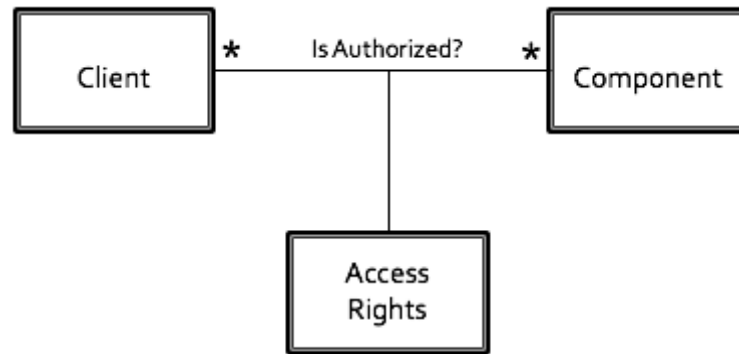


Fig [1] – Authorization Pattern

Between the client and the component that we are trying to access, there is a Access rights check, the client is allotted resource only if it has proper access rights.

Proxy Pattern

The proxy pattern provides our application with a proxy Interface, which is being created once a user requests any method, the user access this interface without being directly communicating with the original interface, this helps our application in many ways, first it provides more security to the application, by denying access to methods which the user doesn't have access to.

The proxy object can be used as substitute to another object, it provides access control to the application.

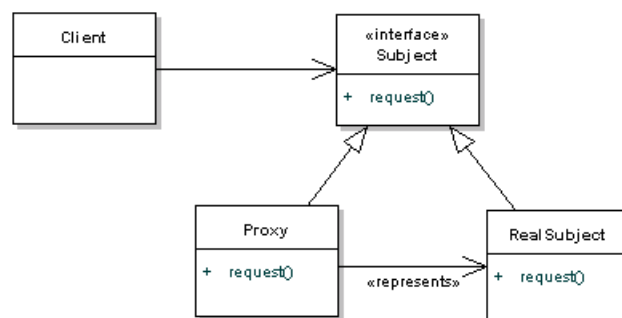


Fig [2]- Proxy Pattern

From Fig – 2, we can see that whenever a client wants to access the Real Subject he will be given access to a proxy object, which may or may not contain the whole functionality of the entire application, this way we can use this pattern in our requirement for Role based access control.

Java Annotations

In this application, I am using Java Annotations to achieve RBAC. Java Annotations helps create custom annotations required by us and use them. I have created a Interface named RequiresRole(String), this is used by the methods in the application to secure the application from unauthorized users, once any method is annotated with This annotation, then that method will only be accessible for users who are of this type.

Reflection Pattern

Using Java Annotations we have annotated our methods with the annotations, now in order to know which method is annotated with which role, we need to access the class to find out this during runtime, to do this we need reflection pattern, once the application is started, and receive a request from the client, application needs to determine, what is the role present on the method, Reflection helps us do exactly this. It can be used to inspect the class and find out what's in there during runtime, but only at the cost of an extra overhead.

RBAC

Used to set access rights to users based on the role assigned, minimizes the overhead by reducing the access rights given to individual users

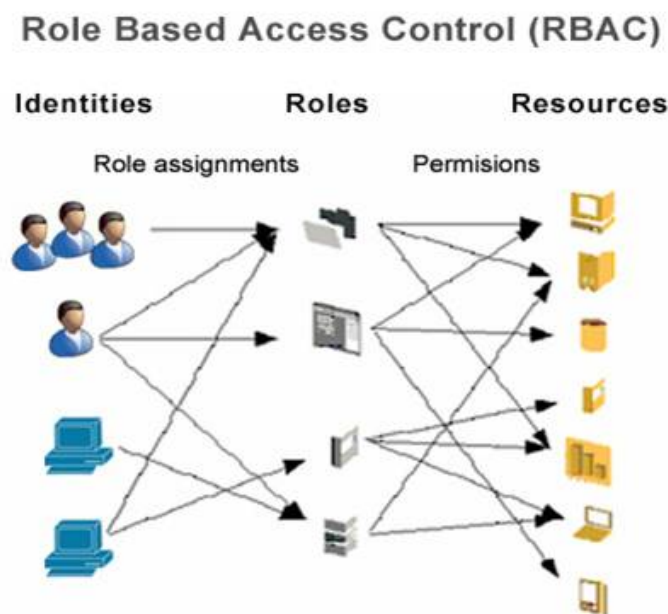


Fig [3]- RBAC

Class Diagram

class diagram, also there is a separate Image in the source directory(ClassDiagram.jpg)

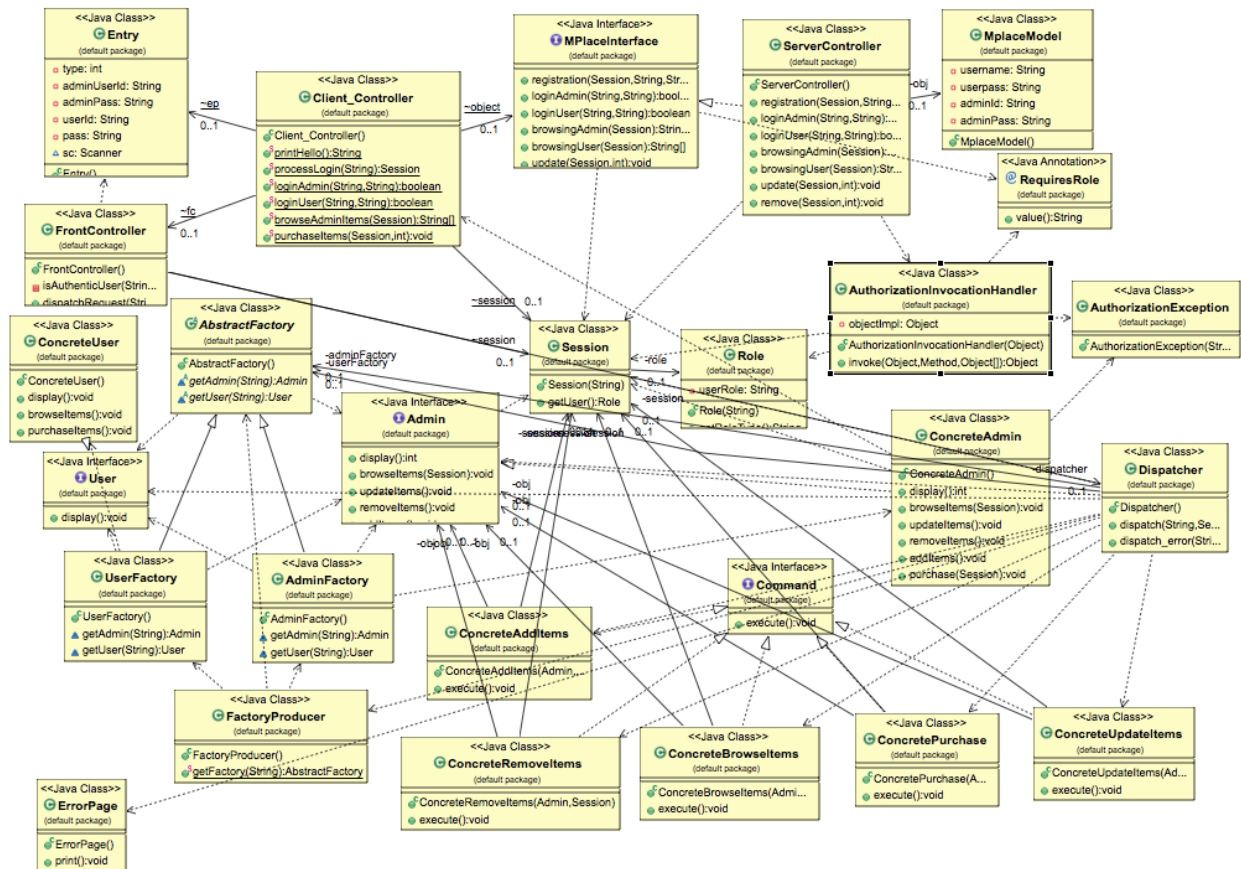


Fig [4]- Class Diagram

Newly Created Classes

- RequiresRole.java

This is an Interface which takes argument a String, which in my case is a String, this helps us define new roles wherever required.

- Session.java

This class helps us in creating a Session object of the user, this session can be used in accessing the methods present on the server.

- Role.java

This class helps in retrieving the role type of the user, it has a method named `getRoleType()`, this returns the role of the user, which is being used in the Reflection pattern.

- AuthorizationInvocationHandler.java

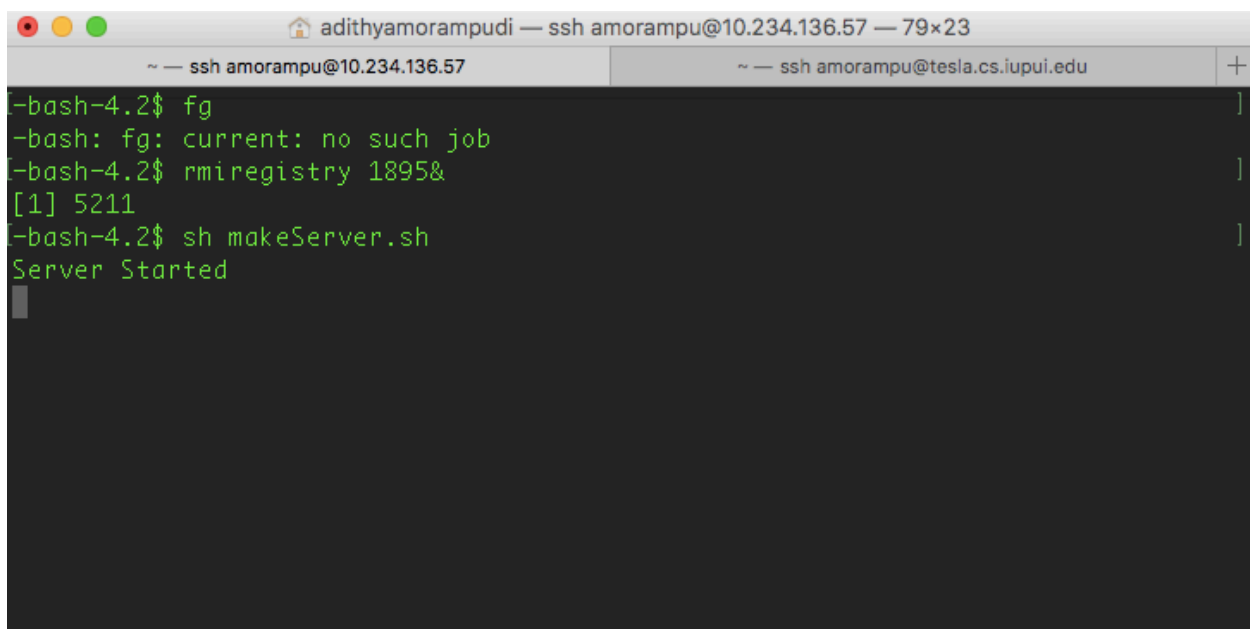
This class helps us to find out if there is a annotation on the required method and also checks if the user is of the type required by the annotation, if they both match then the user is granted access to the method, if not then a `Authorization Exception` is thrown.

- AuthorizationException.java

Created a custom exception class, this exception is thrown whenever the role required by the method doesn't match with the role of the user. This exception needs to be caught at runtime.

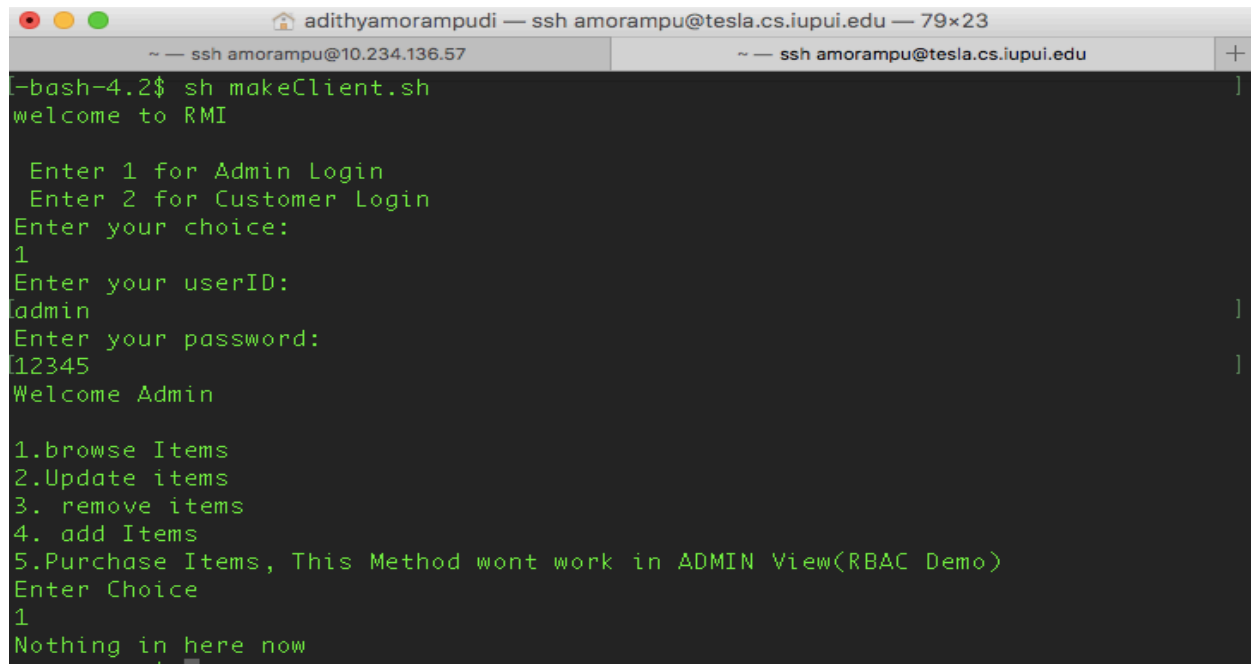
Sample Runs

Screen Shot to show the running Instance of the Server, please note, the server should always be run on 10.234.136.57 only.

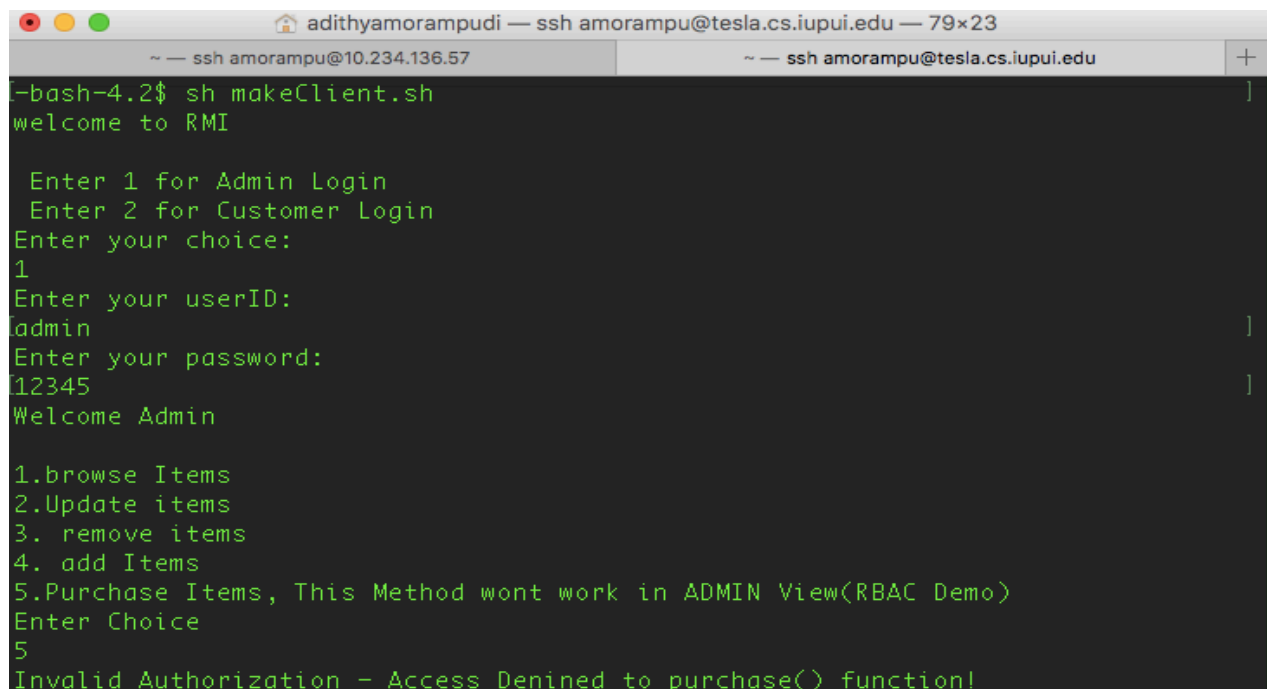


```
adithyamorampudi — ssh amorampu@10.234.136.57 — 79x23
~ — ssh amorampu@10.234.136.57      ~ — ssh amorampu@tesla.cs.iupui.edu  +
-bash-4.2$ fg                                ]
-bash: fg: current: no such job              ]
-bash-4.2$ rmiregistry 1895&                  ]
[1] 5211                                       ]
-bash-4.2$ sh makeServer.sh                   ]
Server Started                                ]
```

A running Instance of the client, which is being executing and sending requests, Showing RBAC for browse items.

A terminal window titled 'adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23'. The terminal shows the execution of 'makeClient.sh', which prompts for a choice between Admin Login (1) and Customer Login (2). The user enters '1' for Admin Login, provides 'admin' as the userID and '12345' as the password, and is welcomed as 'Admin'. A menu of options is displayed: 1.browse Items, 2.Update items, 3.remove items, 4.add Items, and 5.Purchase Items, This Method wont work in ADMIN View(RBAC Demo). The user enters '1' to browse items, and the terminal outputs 'Nothing in here now'.

Screen Shot to show RBAC when admin tries to purchase item from Admin account, for now I am assuming that admin doesn't have the right to make a purchase from his account. And I have implemented RBAC for all the methods on the server side, but for demonstration I am only showing 2 methods one which has access and the other whose access will be denied for Admin.

A terminal window titled 'adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23'. The terminal shows the execution of 'makeClient.sh', which prompts for a choice between Admin Login (1) and Customer Login (2). The user enters '1' for Admin Login, provides 'admin' as the userID and '12345' as the password, and is welcomed as 'Admin'. A menu of options is displayed: 1.browse Items, 2.Update items, 3.remove items, 4.add Items, and 5.Purchase Items, This Method wont work in ADMIN View(RBAC Demo). The user enters '5' to purchase items, and the terminal outputs 'Invalid Authorization - Access Denined to purchase() function!'.

Conclusion

After Implementing this assignment, I have understood what exactly RBAC is, how proxy works, how reflection works and how Authorization pattern works, I have also understood the use of custom annotations and learned how to declare custom Exceptions.

References

- [1]. https://en.wikipedia.org/wiki/Front_controller
- [2]. https://www.tutorialspoint.com/design_pattern/proxy_pattern.htm
- [3]. <https://www.geeksforgeeks.org/reflection-in-java/>
- [4]. Class slides
- [5]. Diagrams from class slides