

OBJECT-ORIENTED PROGRAMMING

GROUP-26

Project 6 - Booking Management System

Videos-

https://drive.google.com/drive/folders/1lwzETY_1u3cywrEGGZOiSfvPdn1zlMYN

MANTRI SRI KRUTHI

2021A7PS2006P

ABIR ANIKET ABHYANKAR

2021A7PS0523P

NAVNEET SINGLA

2021A7PS1450P

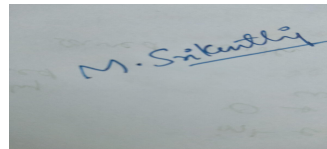
HARPREET SINGH ANAND

2021A7PS2416P

ANTI-PLAGIARISM STATEMENT

We hereby declare that this piece of work is the result of our own ideas and writing. No part has been copied from the internet or from works already presented in the academic field by other students.

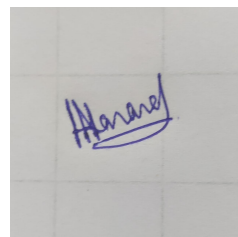
Mantri Sri Kruthi



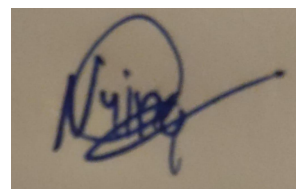
Abir Aniket Abhyankar



Harpreet Singh Anand



Navneet Singla



Contribution Table

NAME OF THE STUDENT	CONTRIBUTION OF THE STUDENT
MANTRI SRI KRUTHI	(i) Helped in Making the Project Report.
ABIR ANIKET ABHYANKAR	(i) Wrote The Code for Entire Project, Start to End. (ii) Helped in Making the Project Report (Design Pattern and SOLID Principles Sections). (iii) Made Final UML Diagram and helped formulate initial Phase 1 UML Diagrams as well. (iv) Testing and Debugging
NAVNEET SINGLA	(i) Formulating and Making the Phase 1 UML, Use-Case and Sequence Diagrams and making Final Diagrams as well. (ii) Helped in Making the Project Report.
HARPREET SINGH ANAND	(i) Formulating and Making the Phase 1 UML, Use-Case and Sequence Diagrams and making Final Diagrams as well. (ii) Helped in Making the Project Report. (iii) Testing and Debugging

Design Pattern

- We have not used any specific design patterns while developing the project.
- Our code does bear striking resemblance to the Observer Design Pattern.
- In our application for both the User and the Admin Classes, we have implemented dynamic updation of display data, and these changes to our stored data are reflected on every window after a refresh. Thus the retrieval, updation and storage of data is greatly demonstrated.
- The Appls Class which uses Several Functions of the Application Class and allows the Admin to view Bookings Pending for Approval, retrieves data every time page is refreshed. So the Admin can continuously view applications even if they are being when he is viewing pending applications. This is implemented by the Application.ApplInfo() Static Method.
- We have also used a Number of Static Methods in the Project, as we feel they are very much suited to retrieve data when object creation is not required.
- We have also accidentally incorporated some part of a Command Design Pattern from the User Side, as a User Object invokes all the functions which involve updating and retrieving data. We have not implemented the same from the Admin End and have used Static methods to perform Admin Tasks.

Analysing Code w.r.t S.O.L.I.D Principles

1. The Single Responsibility Principle

- The Single Responsibility Principle states that a class should do one thing and therefore it should have only a single reason to change.
- We have implemented this principle in our application. Each Class has been assigned a particular important role.
- The GUI Classes help in implementing the GUI part of the Program, these are .java files which are accompanied with a .form file. All these files help in easy retrieval of commands from Users and Admins.
- The User and Admin Classes provide functionality for Users and Admins. Both extends the Login Class, which is used to Register and Login Individuals. These classes call upon methods from the Application, Room and Date Class.
- The Application Class provides all functions which pertain to the Bookings Made by the Users and Approved/Disapproved by Admins. Everything related to Bookings is implemented by the Application Class.
- The Room Class, provides methods to check Validity of rooms in terms of Number of Students and helps in booking the appropriate room.
- The Date Class helps Marking the Dates for Each Booking.

2. Open-Closed Principle

- The Open-Closed Principle requires that classes should be open for extension and closed to modification.
- This principle is showcased very well in the project. All Classes have Private variables which are closed for modification, and any new functionality can be added without modifying the original code, as was done throughout the project.
- The only thing one might want to change are the locations of the files which store data.

3. Liskov Substitution Principle

- The Liskov Substitution Principle states that subclasses should be substitutable for their base classes.
- We don't think this principle is displayed that well in the project as we only have a Login Class which extends both User and Admin classes and JFrame extending our GUI Classes.
- We feel that if User and Admin Objects are replaced by Login Class, we may face some errors. Thus this principle is applicable to our extent on a very small scale.

4. Interface Segregation Principle

- The principle states that many client-specific interfaces are better than one general-purpose interface. Clients should not be forced to implement a function they do not need.
- We have only used the Runnable Interface during development, and I believe that in accordance with the Open-Closed principle, all our functionalities can be fulfilled innately. Thus extending our project won't require implementing unwanted functions.

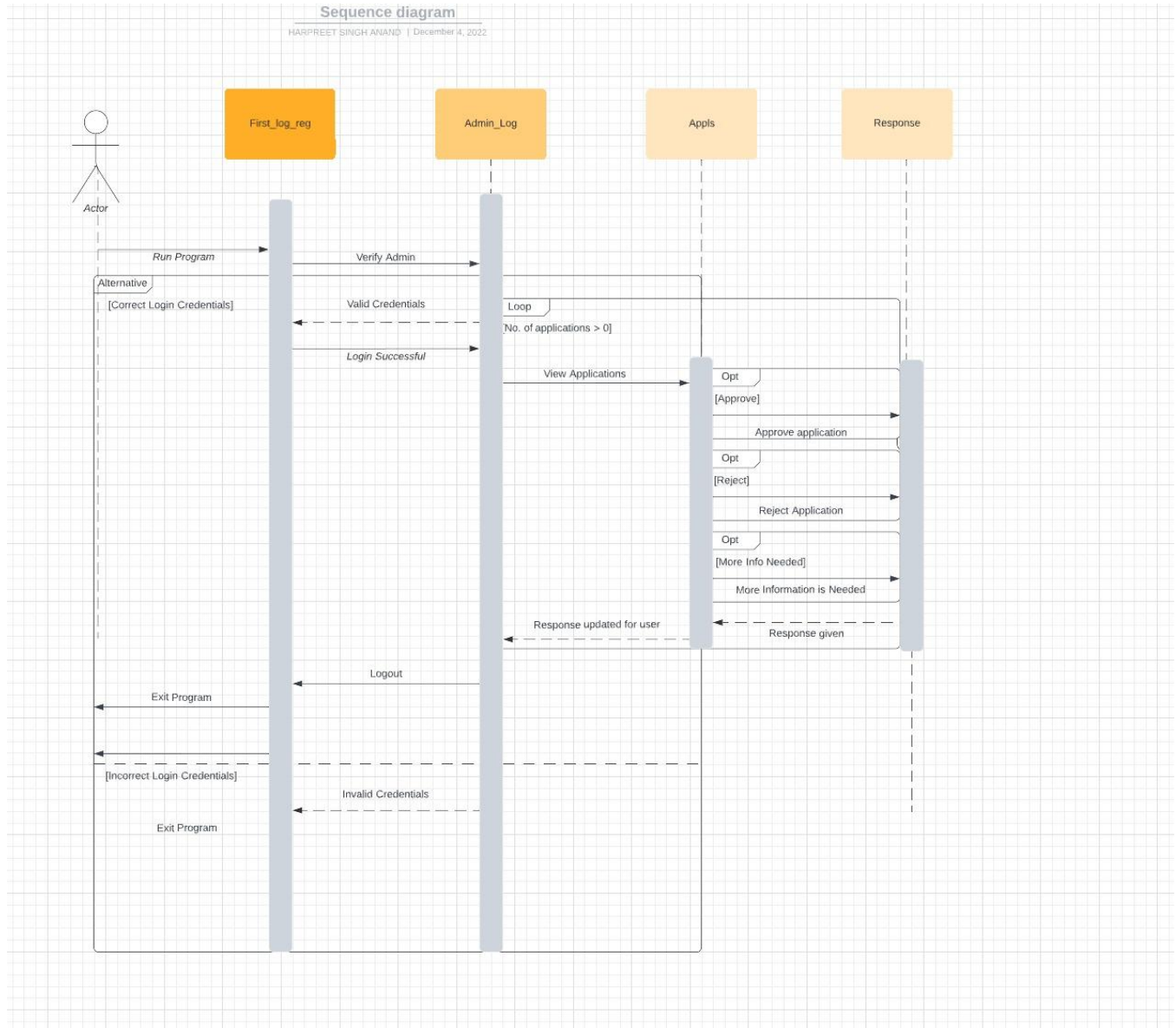
5. Dependency Inversion Principle

- The Dependency Inversion principle states that our classes should depend upon interfaces or abstract classes instead of concrete classes and functions.
- This principle is not applicable to our application, as we have made Concrete Classes for every functionality. Except for the Runnable Interface, we have not used any Abstract Classes and Interfaces.

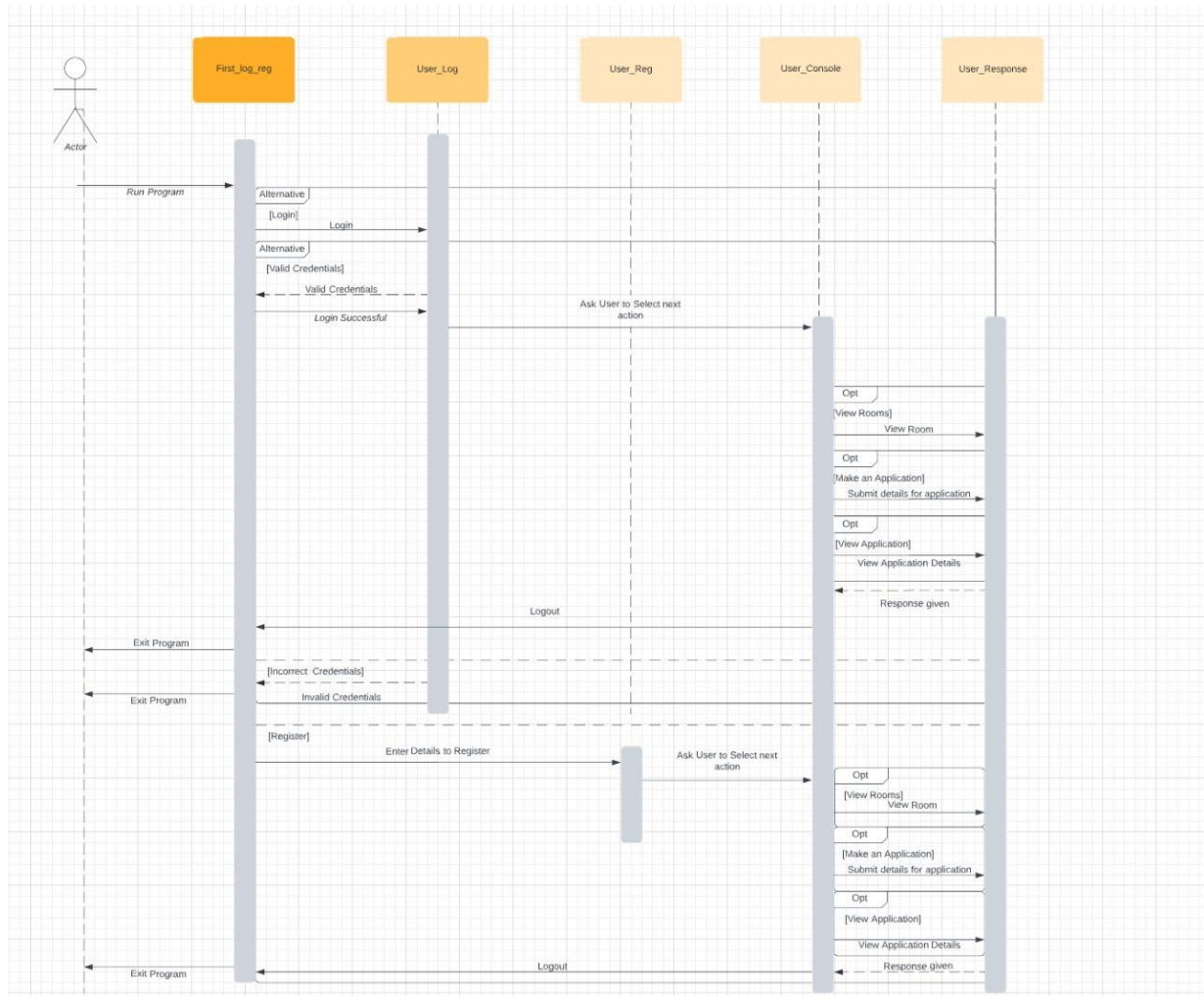
[illegible]

UML Sequence Diagram

(i) Admin Mode



(ii) User Mode



UML Use-Case Diagram

