A REPORT

ON

LAUNDROMAT MANAGEMENT SYSTEM

BY

K DHANUSH	2020B2A70709P
NACHIKET KANDARI	2021A7PS2691P
SANJU S	2021A7PS2537P
SIDARTHA SANKARA PATI	2020B2A70687P

AT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI, PILANI CAMPUS

DECEMBER 2022

Anti-Plagiarism Statement

1.) We know that plagiarism means taking and using the ideas, writings, works or inventions of another as if they were one's own. We know that plagiarism not only includes verbatim copying, but also the extensive use of another person's ideas without proper acknowledgement . We know that plagiarism covers this sort of use of material found in textual sources and from the Internet.

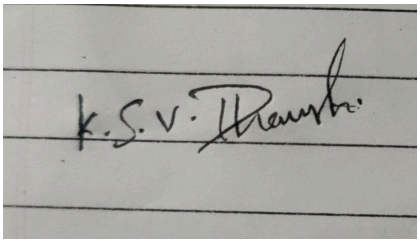
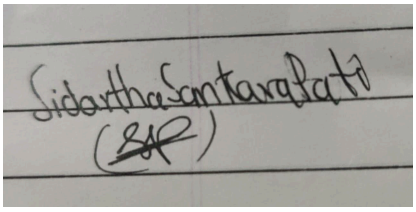
2.) We acknowledge and understand that plagiarism is wrong.

3.) All the codes written throughout this project have been written by us. All the work done in this project is our original work.

4.) We have not allowed, nor will we in the future allow, anyone to copy my work with the intention of passing it off as their own work.

5.) We also understand that if we are found to have violated this policy, we will face the consequences accordingly.

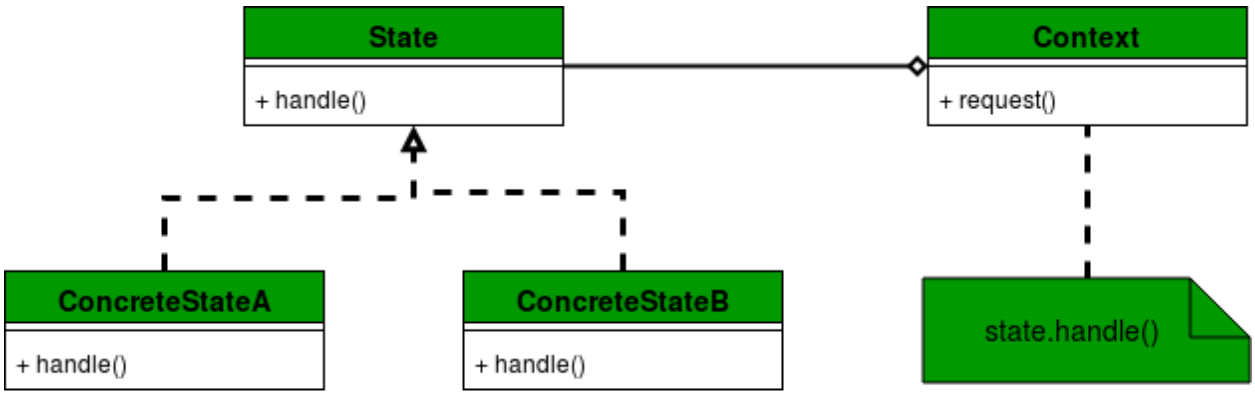
Contribution Table

NAME	CONTRIBUTION	SIGN
KOMMULA SAI VENKATA DHANUSH - 2020B2A7PS0709P	Made UML diagrams. Implemented a few functions. Tested working of program. Fixed the bugs that popped up while testing.	
NACHIKET SINGH JAGMOHAN SINGH KANDARI - 2021A7PS2691P	Made initial UML diagrams. Made classes other than the main class. Implemented multi-threading. Studied solid principles and tried to implement it wherever possible. Implemented a few functions. Contributed in making the document.	
SIDARTHA SANKARA PATI - 2020B2A7PS0687P	Added storage features. Implemented a few functions. Researched on the required design pattern. Made necessary classes other than the main class.	
SANJU S - 2021A7PS2537P	Made the main class of the program. Implemented a few functions. Did file-handling work. Contributed in making documents. Overall coordination of the team via scheduling meets, creation of whatsapp group,etc..	

Design Pattern Used

At the time of creating this project we were not aware (as design patterns had not yet been taught) about many of the design patterns which could have been applied and used to make this project more efficient. We have not made any design pattern for this project as such, it could be however considered as a mixture of some design patterns.

If we had to do this project again, we would ideally use a state design pattern. State pattern is a behavioral design pattern. In this pattern the object has a state and can change along with different behaviors and actions. If we have to change the behavior of an object based on its state, we can have a state variable in the Object and use an if-else condition block to perform different actions based on the state. The context is an interface to the client to interact. It maintains references to concrete state objects which may be used to define the current state of the object. State is an interface for declaring what each concrete state should do.



We would create an object called wash which would change into various states upon completing/encountering certain actions. For example, we could have a wash object in 4 states, namely dropped, washing, washed and out-for-delivery. All these

states would have their individual concrete implementations with specific methods and variables in each of these states. We could change the state of each wash upon encountering an action (input from client), for example, if the clothes are picked up from a hostel we can change the state from dropped off to washing, or if the clothes have been washed, we can change the state from washing to washed. These simple changes in states would easily serve our purpose and many mentioned functionalities could be implemented with ease. We could easily implement all queries related to each wash and all washes in general. It would also be very easy to keep track of each wash object (using the state which it's currently in).

This pattern would implement polymorphic behavior and it would be very simple to either add or delete states and additional functionalities.

Code Analysis with respect to S.O.L.I.D. Principles

Single Responsibility Principle (SRP) - A class should have one and only one reason to change, meaning that a class should only have one job.

With reference to the single responsibility principle, our project has implemented this principle in our classes as our User, Wash, Washplan classes are created to serve a single purpose, for example :- the user class contains the user details and only one method to add a user (it performs only one job). The class wash has no methods and hence it is only responsible for creating a wash.

Open-Closed Principle (OCP) - Objects or entities should be open for extension, but closed for modification.

With reference to the open-closed principle, generally we have to use abstract classes, interfaces and inheritance in order to use the open-closed principle and since we haven't used either of these we do not check on the open-closed principle.

Liskov Substitution Principle (LSP) - Every subclass/derived class should be substitutable for their base/parent class.

We did not use inheritance in our project and hence we cannot comment on the implementation of the Liskov substitution principle.

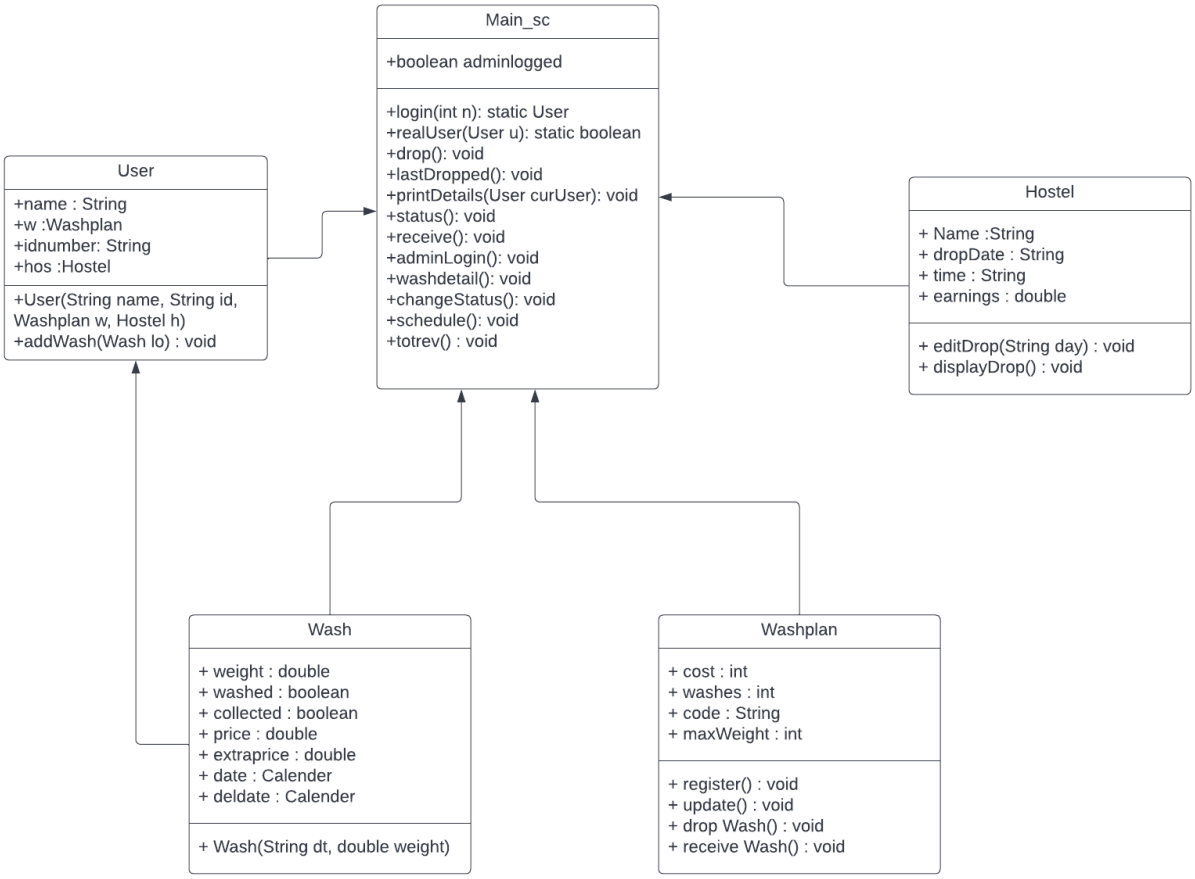
Interface Segregation Principle (ISP) - The principle states that the larger interfaces split into smaller ones. Because the implementation classes use only the methods that are required. We should not force the client to use the methods that they do not want to use.

In our project, we did not implement interfaces and hence we cannot comment on the interface segregation principle.

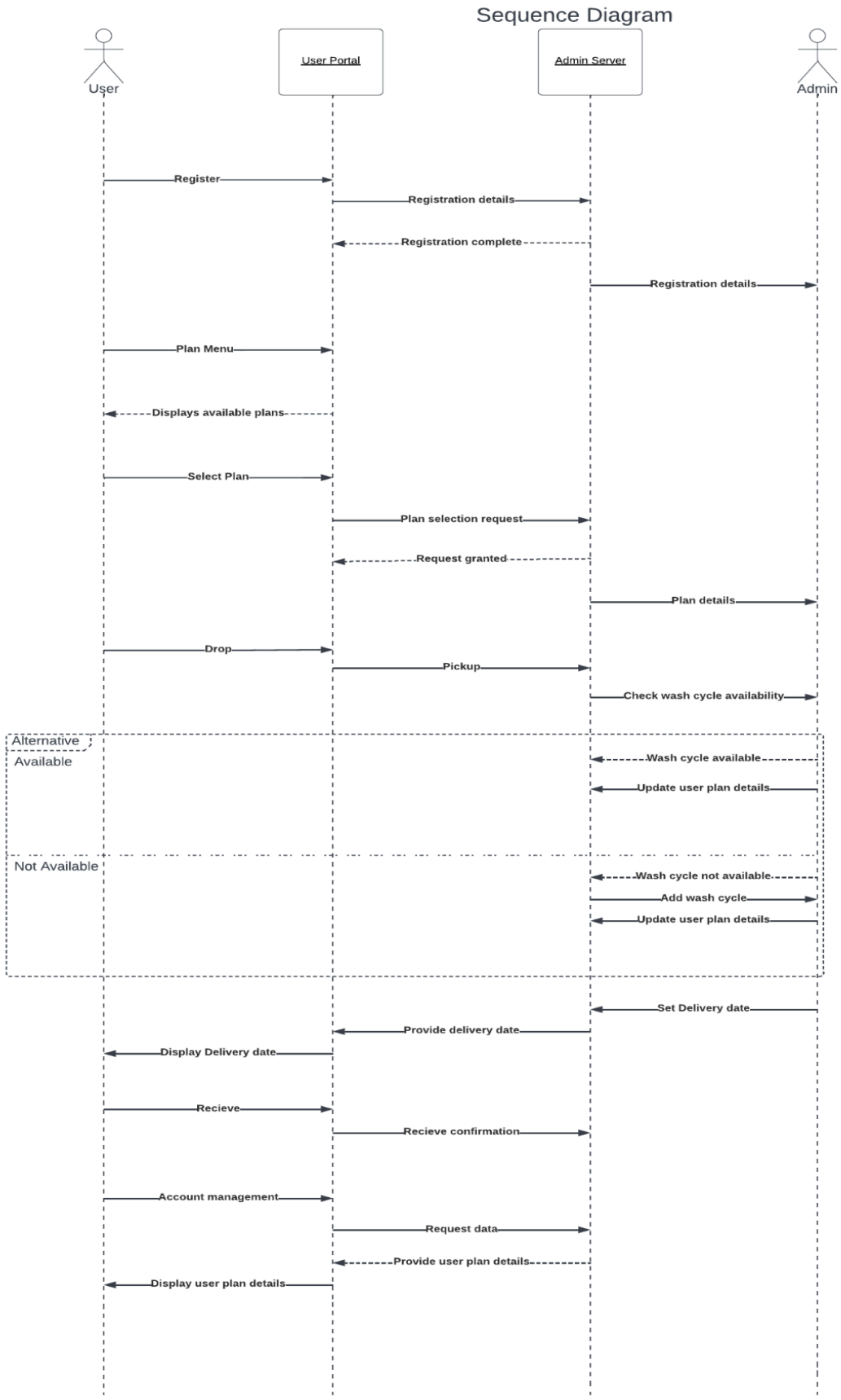
Dependency Inversion Principle (DIP) - The principle states that we must use abstraction (abstract classes and interfaces) instead of concrete implementations. High-level modules should not depend on the low-level module but both should depend on the abstraction.

Since we did not implement any interfaces in our project, we fail to check the dependency inversion principle which prefers the usage of abstract classes and interfaces over classes.

UML Diagrams



Use Case Diagram


Sequence Diagram
User
User Portal
Admin Server
Admin
Register
Registration details
Registration complete
Registration details
Plan Menu
Displays available plans
Select Plan
Plan selection request
Request granted
Plan details
Drop
Pickup
Check wash cycle availability
Alternative
Available
Wash cycle available
Update user plan details
Not Available
Wash cycle not available
Add wash cycle
Update user plan details
Set Delivery date
Provide delivery date
Display Delivery date
Recieve
Recieve confirmation
Account management
Request data
Provide user plan details
Display user plan details

Video Drive Link

https://drive.google.com/drive/folders/1Yud4VPlVoejt95qo35-PaZMFtcNd801X