Contents

[Introduction to the System 2](#_Toc109399417)

[Architecture used in the system 2](#_Toc109399418)

[Layers with Code Examples 3](#_Toc109399419)

[Benefits of using the Layered Architecture 7](#_Toc109399420)

[Main Users of the system 7](#_Toc109399421)

[System Review with Instruction manual 9](#_Toc109399422)

[Searching 19](#_Toc109399423)

[Database Mapping 20](#_Toc109399424)

[Exception Handling 23](#_Toc109399425)

[Password Hashing 25](#_Toc109399426)

[Low Coupling and High Cohesion 27](#_Toc109399427)

[Transaction Management 28](#_Toc109399428)

## Introduction to the System

This system has been designed for the company called ‘E-shift’ which mainly focuses on shifting household items from point A to point B

## Architecture used in the system

As this involves numeral number of transactions and especially repetitive CURD (Create, Update, Read, and Delete) operations I’ve decided to use the Layered/Multitier Architecture.

Mainly the layered architecture can be represented be the following diagram



Figure 1

As mentioned above mainly 4 layers are used in the architecture. I have segregate the project mainly 3 parts as mentioned above

1. View (Presentation Layer)

* View
* Controller

1. Bo (Business Layer)

* BO

1. Dao (Database / Persistence Layer)

* Dao

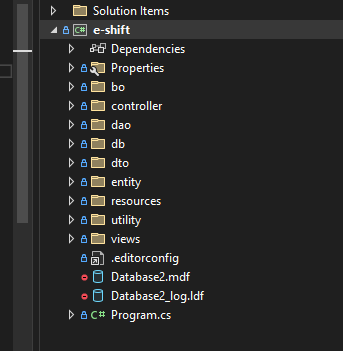


Figure 2

## Layers with Code Examples

**Business Layer (BO Layer)** – The business logic of the business is conducted here. Validation such as checking of duplicates, Generating unique ID’s are conducted here. The only purpose of this layer is to manipulate the business logic. Following is an example of how I used this layer.

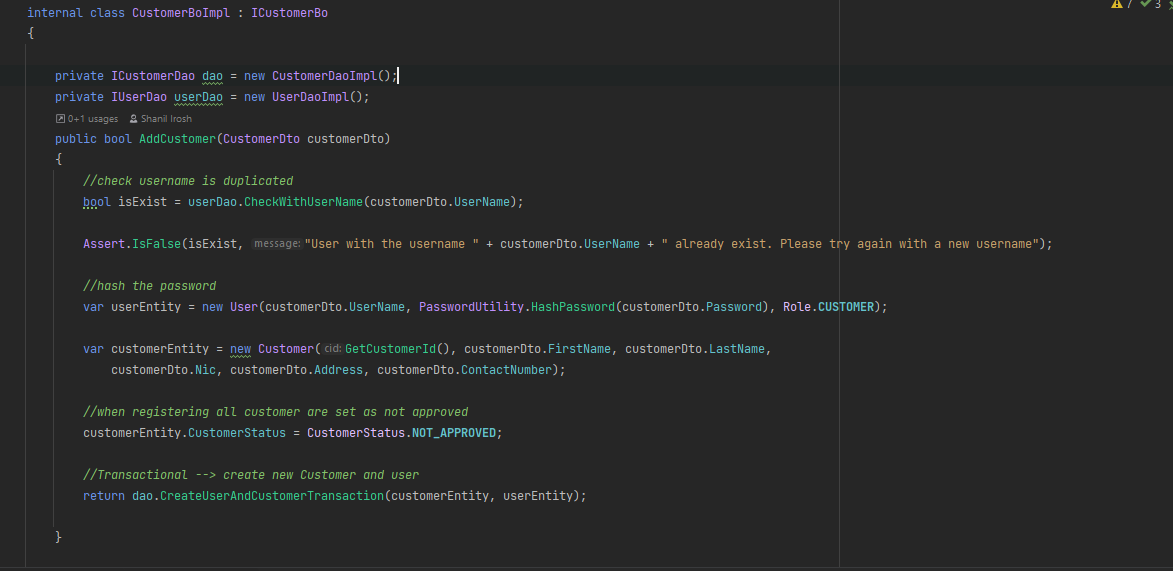


Figure 3

As shown above all the business related logics such as checking whether username is already there in the system, hashing of passwords, approval statuses are added in this layer. And if there are any mismatches then exceptions are thrown from this layer to the presentation layer and displayed.

**Presentation Layer** – Mainly responsible for retrieving the dataset from the client side. And the UI validations such as Empty textbox, submitting text values for numerical inputs is conducted by the layer. In this project the views sections and the controller section represents the presentation layer. This layer has no idea about the business data. And all the exceptions thrown from the other layers are captured here and displayed to the user. (As exceptions are thrown to the called method, from the database layer to presentation layer an exception is thrown without any grace)

Exception handing is mentioned in the Exception handling section

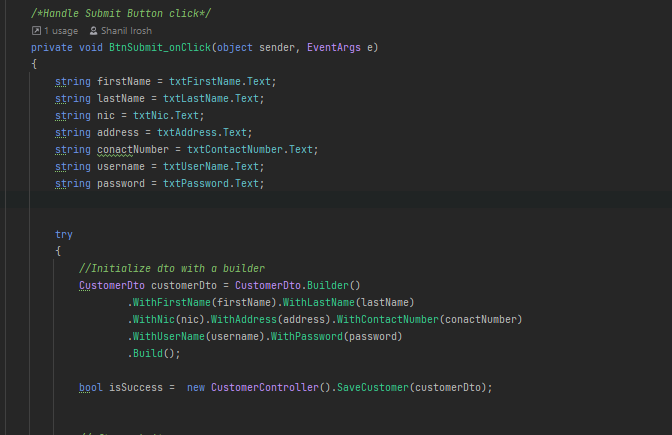


Figure 4

As in the above diagram in the presentation layer when we create the customerDto object all the fields are validated if an mandatory field is not filled by the user and exception will be thrown from the dto objected an it will be captured as mentioned in the below diagram

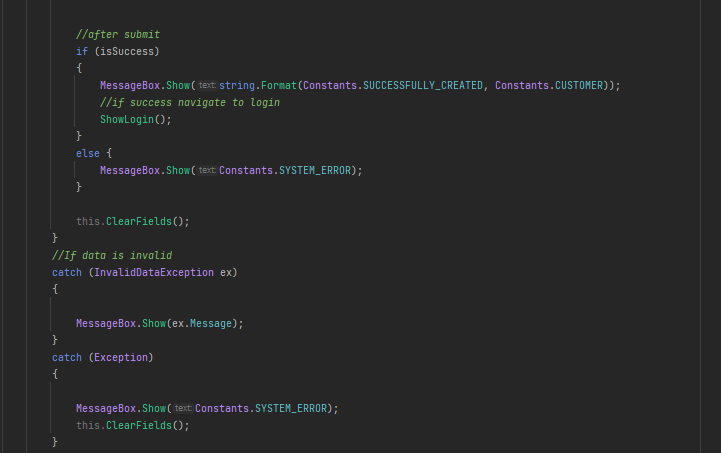


Figure 5

When creating the customerDto object if an exception is thrown it will go to the InvalidDataException catch block and that message is displayed to the user.

If an exception like NullPointer,SQLException,ClassCast,IndexOutOfBound occurred it will go to the exception block and a common “System Error” Message will be displaed to the user

**Database Layer (Dao Layer)** – Mainly the communication with the database happens here. This layer has no idea about the business logic. This layer just executes the database queries and retrieves the output. And transaction management activities are also conducted here

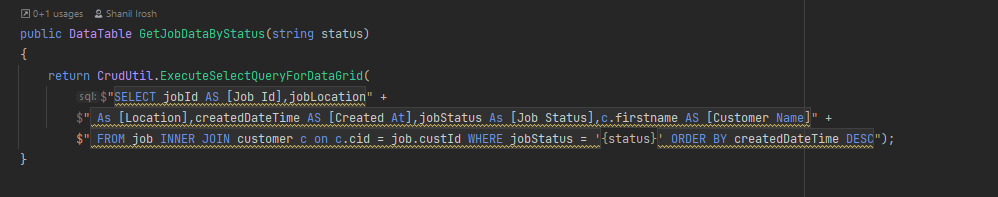


Figure 6

As most of the Tables like Item,Customer involves CRUD operations a common interface called CRUDDao is used with a generic an all the classes with CRUD is inherited from this interface.

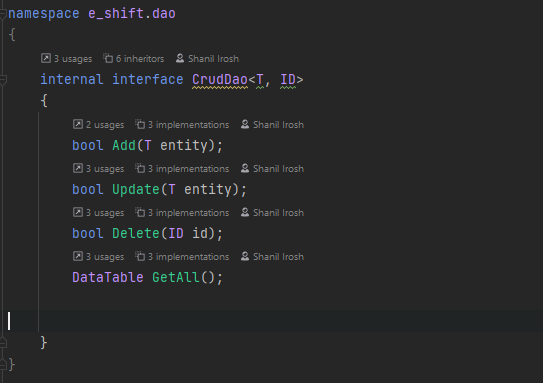


Figure 7

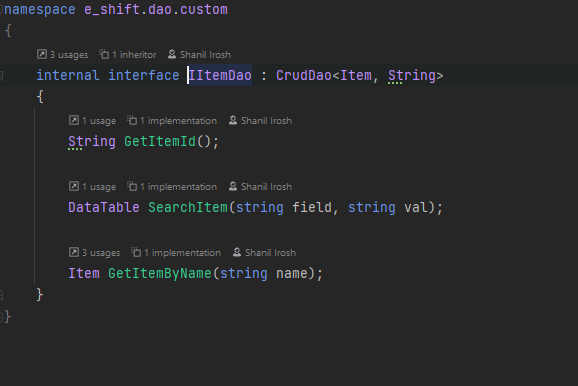


Figure 8

## Benefits of using the Layered Architecture

Really easy to maintain as all layers are segregated separately

Testing is very easy as separate components are maintained

Dependency is reduced as each layer is functioned separately. For an example a front end expert can simply just have to keep on working the presentation layer.

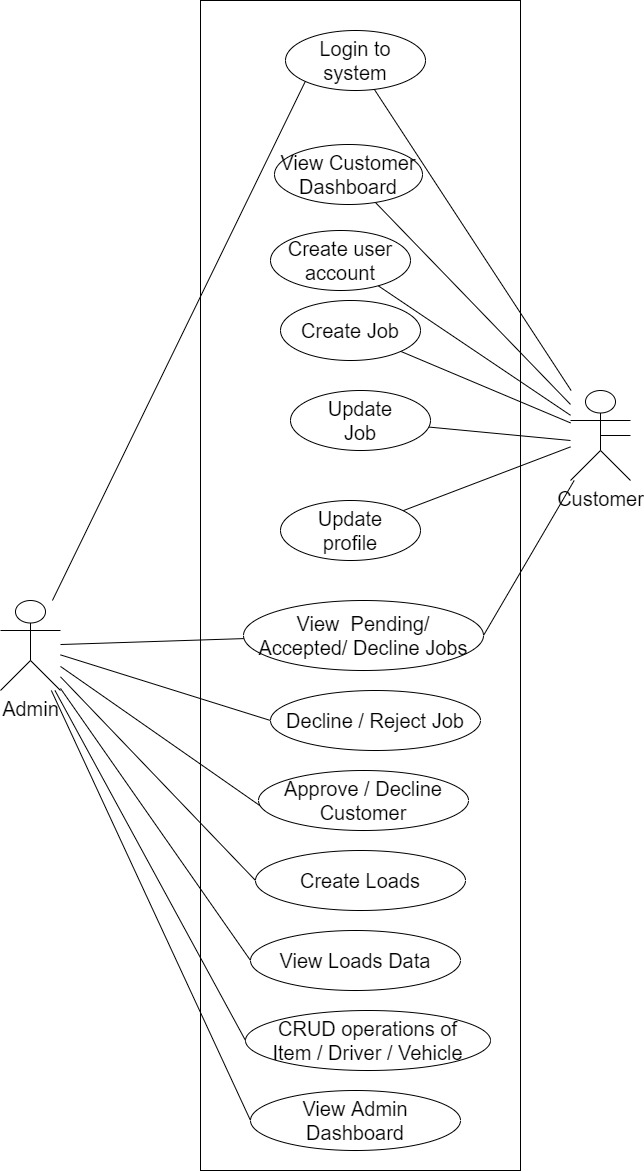
UI changes or UI framework switches can be done very easily as all the business logic and data layer is segregated. For an example, If the business want to develop a mobile application it can be done very simple as only the presentation layer is there to be changed.

## Main Users of the system

Mainly there are two types of users which mainly uses the system as follows

1. Admin User
2. Customer User

Roles of the users can be simply explained by a use case diagram as follows



As mentioned above mainly admin user is having the highest privileges to make changes in the system. An admin is capable of accepting or rejecting any sort of a job and also product item CRUD operations can be conducted by the admin.

Customer responsibility is really simple. Which is to create jobs based on their requirement.

## System Review with Instruction manual

**Run the application and login to the system**

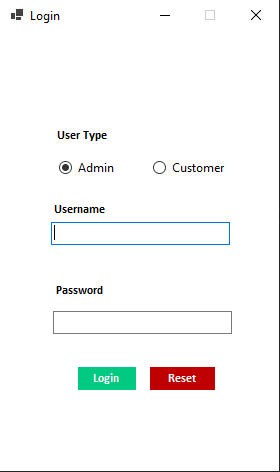


Figure 9

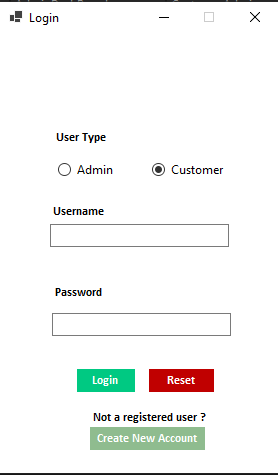


Figure 10

If the user is an admin simply user can login from the system through the admin section.

Note that only a Customer can be created from the system. Because if we allow a normal user to just register as an Admin all the admin privileges like maintaining parameter master data like items, drivers, vehicles will be allowed to them.

In order to create a new customer click the Create New Account under customer section as in the below diagram

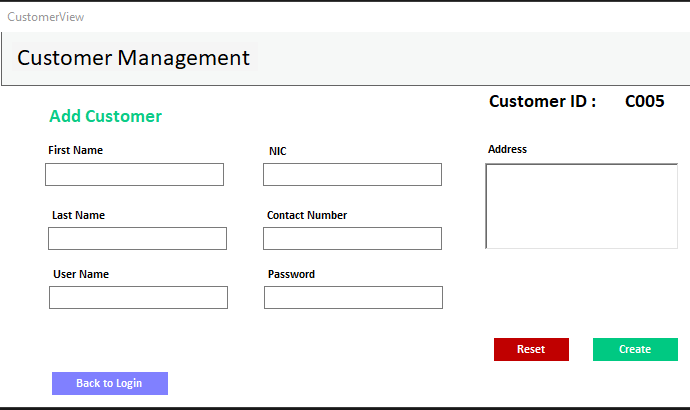


Figure 11

The username and the NIC should be unique and customer can get registered here with his/her detail.

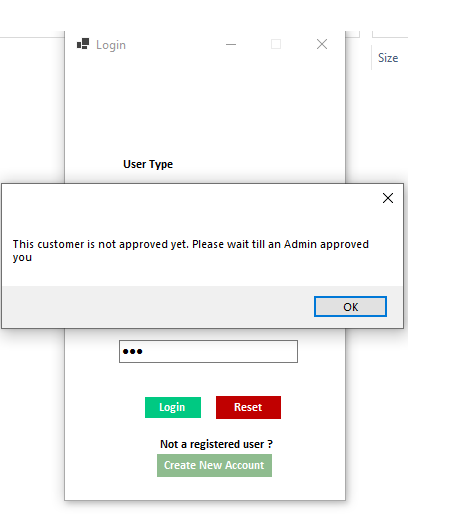


Figure 12

As above when a user get registered an Admin should approve it in order to customer to use it. After approval an Approved user can log in to the system and create necessary jobs as below

Customer screens are as follows

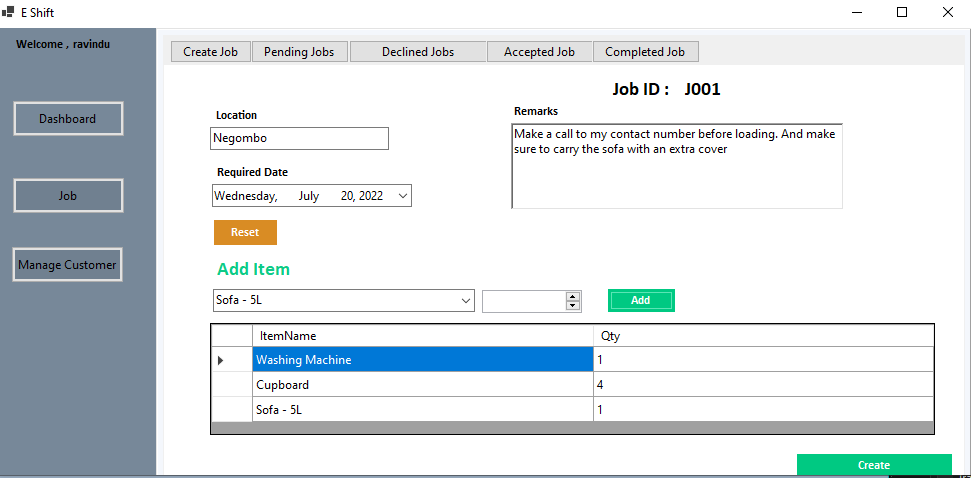


Figure 13

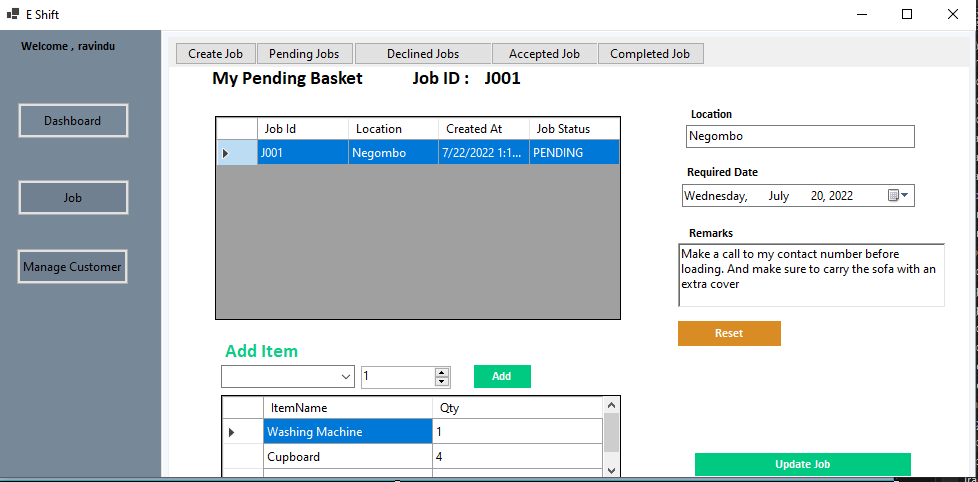


Figure 14

By pressing the pending jobs user can view his/her pending jobs and make updates before an Admin Accept the job

And he/she can view his completed jobs declined jobs subsequently.

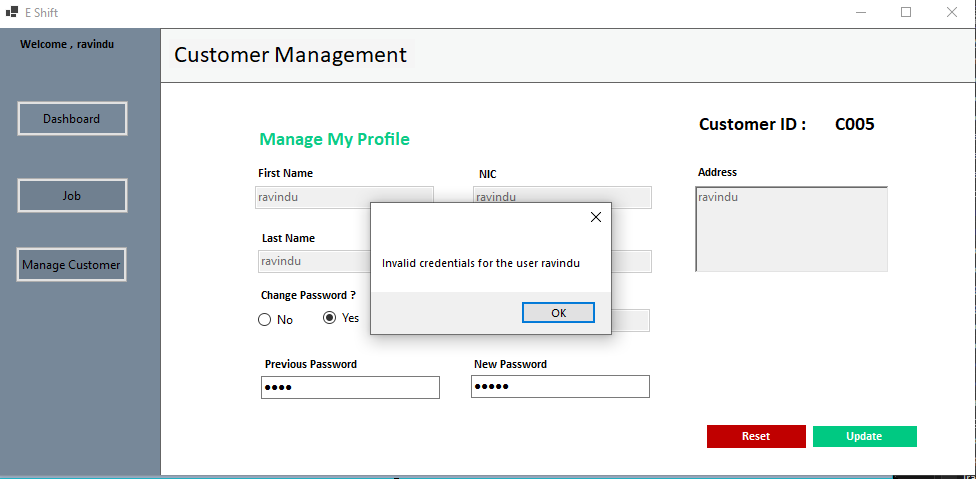


Figure 15

A customer can edit and update his/her information through manage customer section and change user’s password respectively

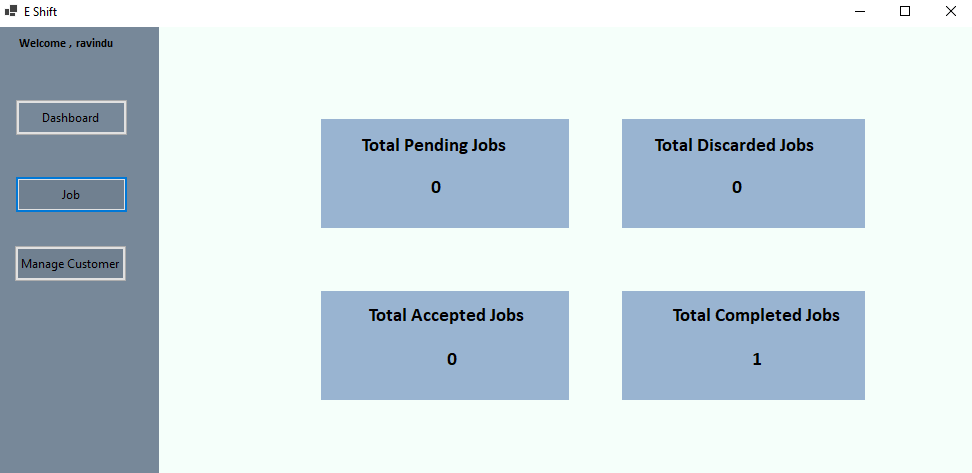


Figure 16

Above is the dashboard of the customer with the specific job count of that customer

Admin screens are as follows

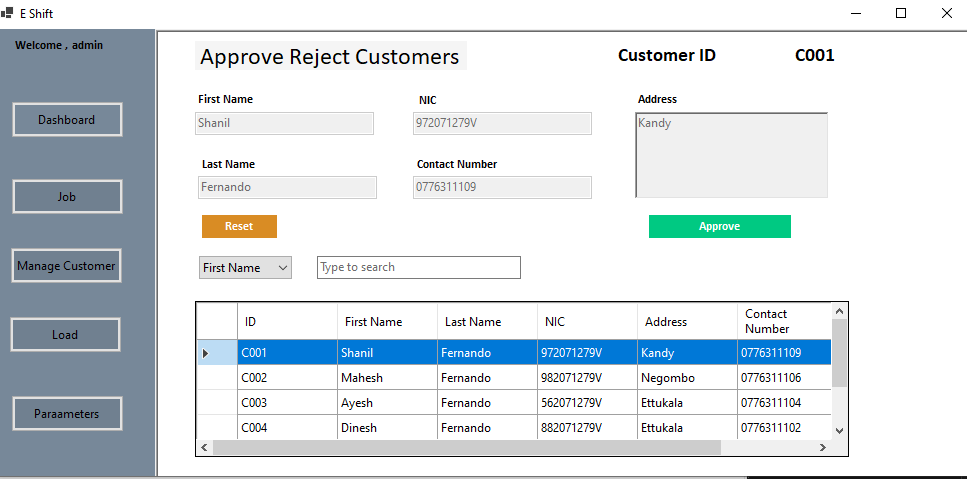


Figure 17

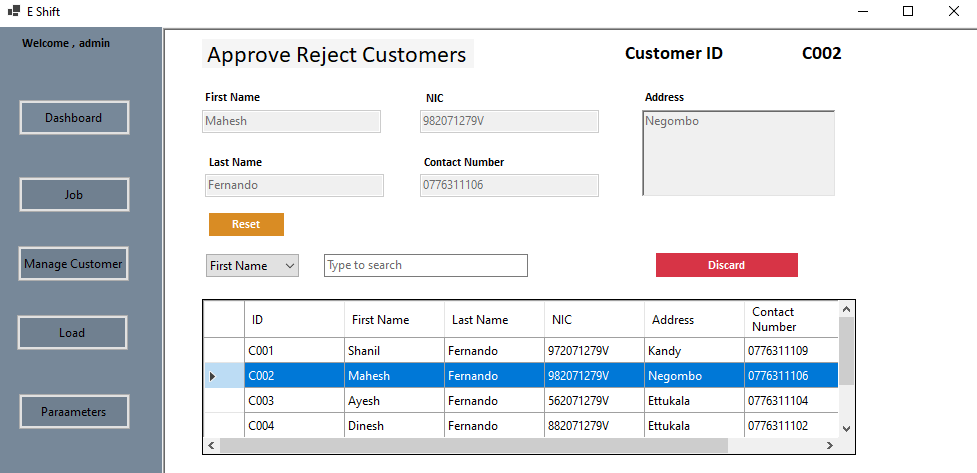


Figure 18

As above customer can approve or decline existing customer so that their logged in facility to the system is disabled

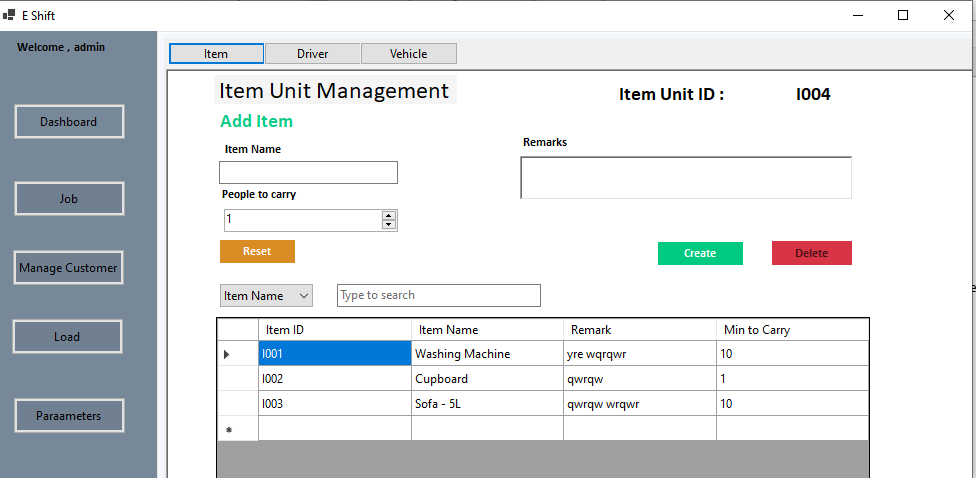


Figure 19

Parameter screen like items, driver, vehicle is applicable to admin in order to manage them. All CRUD operations are possible with these and a user can simple search a record by his/her wishing criteria

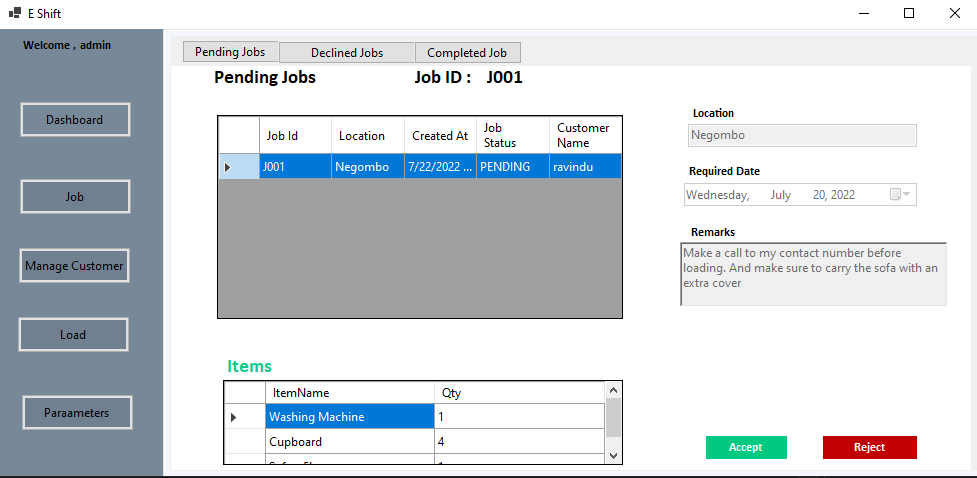


Figure 20

As above an admin can accept or reject a job based on his wish. And from the other tabs user can view all the declined jobs and the completed once

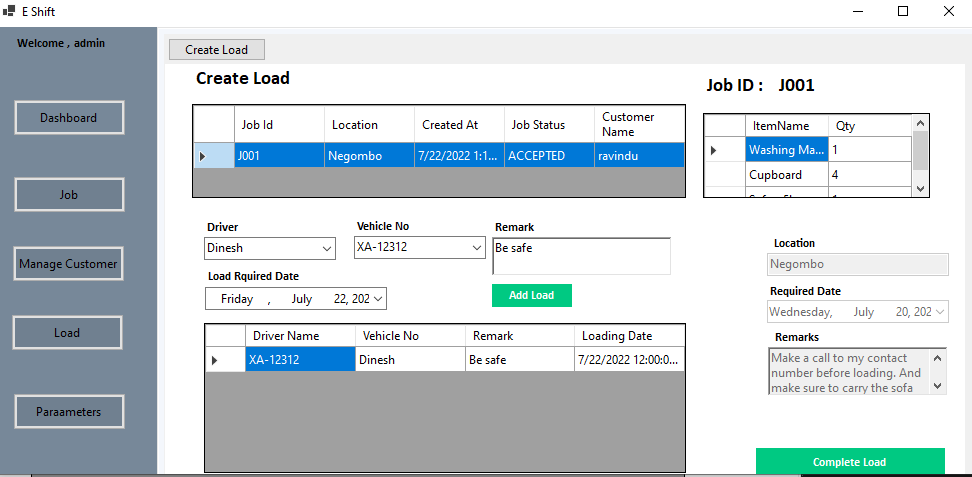


Figure 21

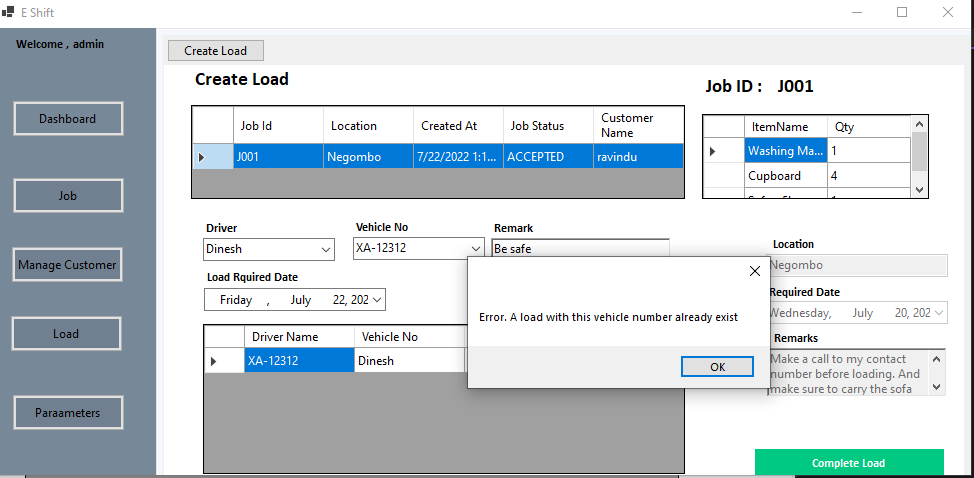


Figure 22

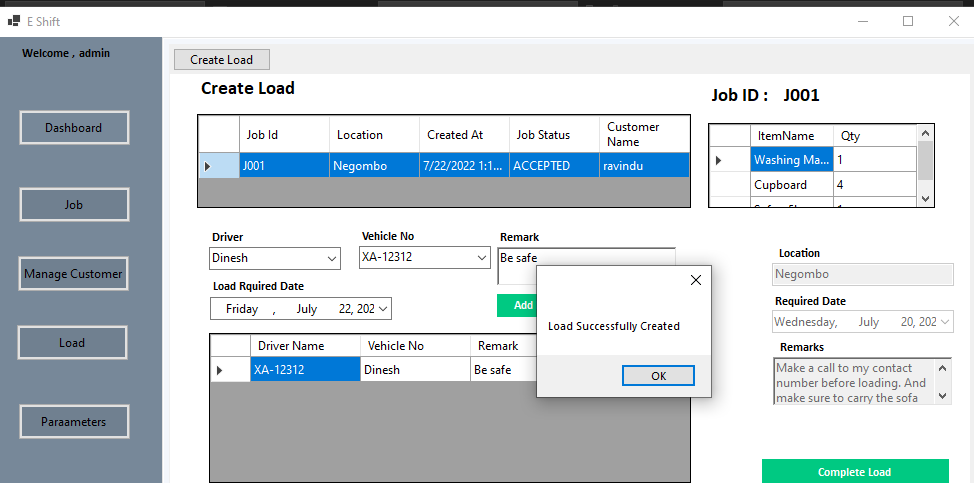


Figure 23

For the completed jobs user can enter loads by adding items. Same vehicle number records cannot be created for a specific job as above. In order create a load user should make sure to not duplicate the same vehicle twice.

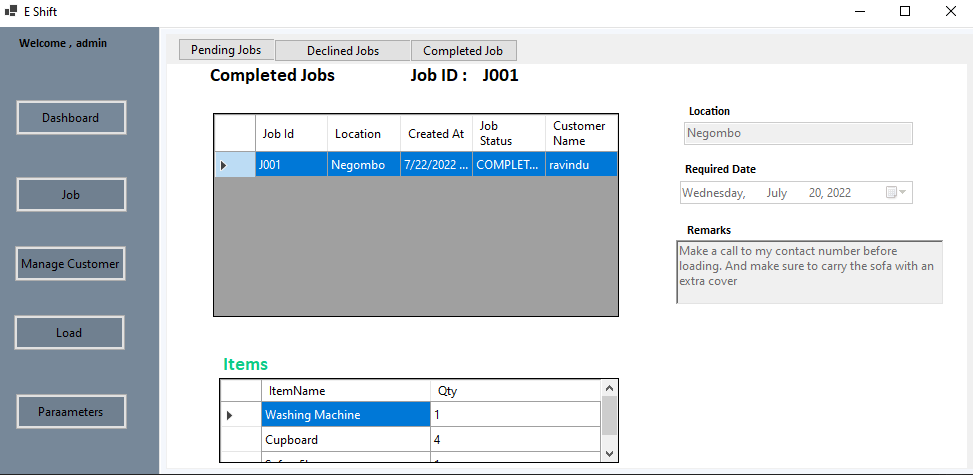


Figure 24



Figure 25

Through admin dashboard user can check the number of pending available for the admin in order to get a higher end idea

## Searching

For searching mainly I’ve used the like operator with the (%) wildcard. User has the opportunity of selecting the attribute that he want to search and conduct the search

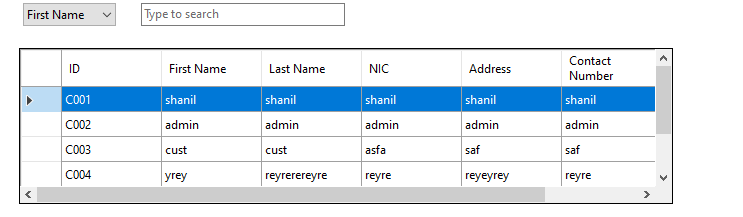


Figure 26

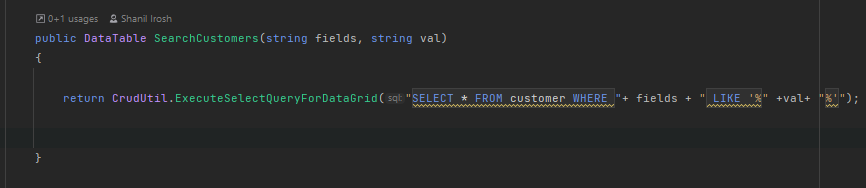


Figure 27

As above whatever data user needs to retrieve is fetched. How a like query with wild card operator is as follows

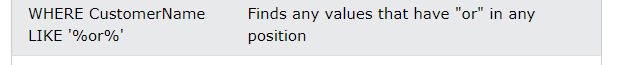


Figure 28

## Database Mapping

Mainly I have used 8 tables in order to design the database. Based on the usage they can be classified in to following two sub categories.

1. Master Data Tables

* Driver – Responsible to persist just driver data
* Vehicle - Responsible to persist just vehicle data
* Customer - Responsible to persist just vehicle data
* Job – Responsible for keeping non-recurring unique data for the job.
* Load – Responsible for keeping the Load data

1. Associate Tables

* Job Detail – Primary keys of items and job is merged and kept as primary key here to identify the items of the job

**UML Diagram of the Database**

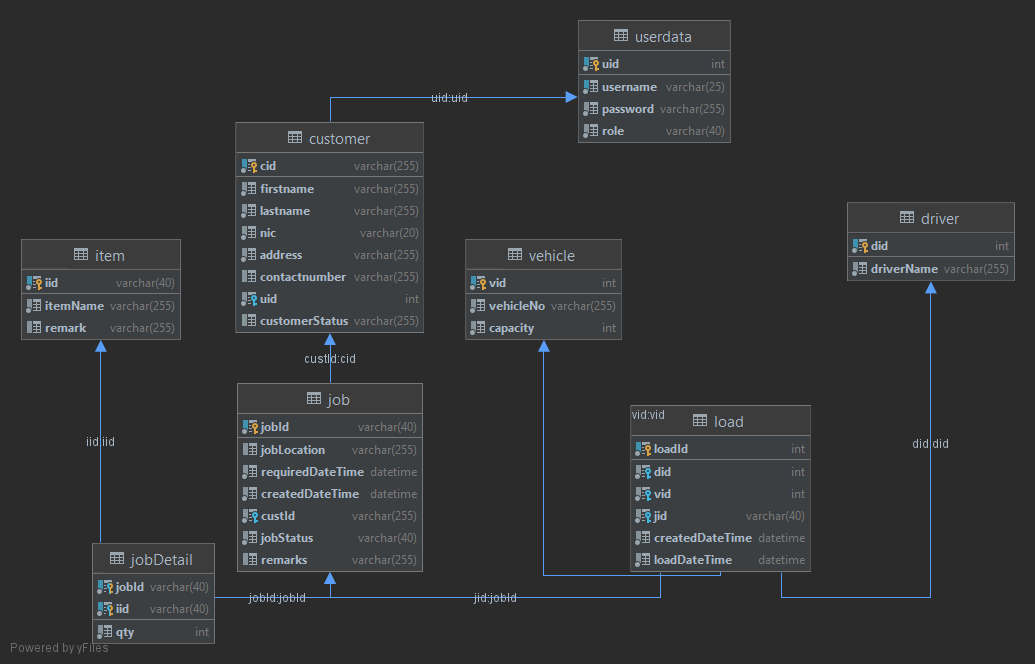


Figure 29

**ER Diagram of the Database**

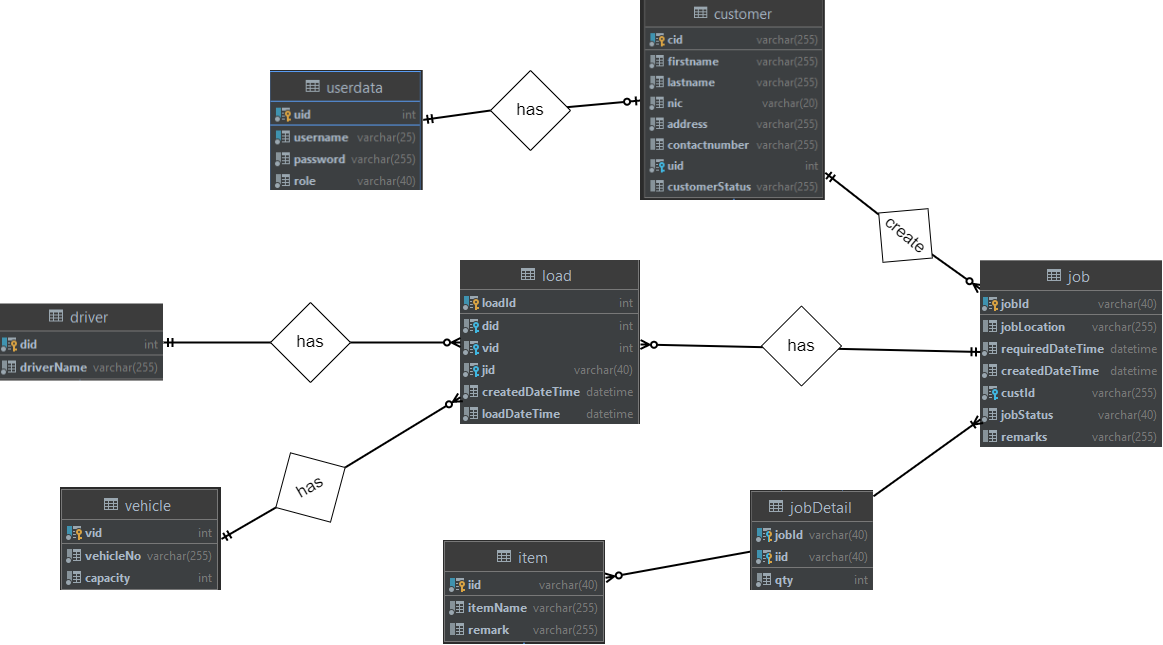


Figure 30

## Exception Handling

In this project mainly I have used “**Throw Everywhere Catch Later**” approach. In this architectures even if an exception occurred in the last layer it will be thrown till the initial layer (presentation) and the exception is handled there. In order to throw exceptions I have used a Utility class called **Assert.**

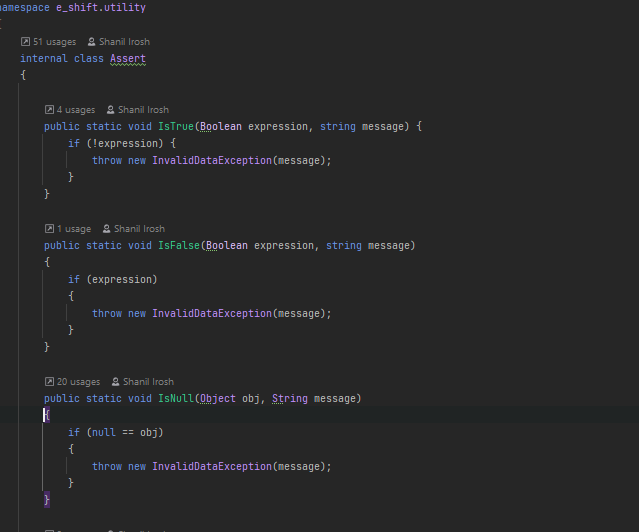
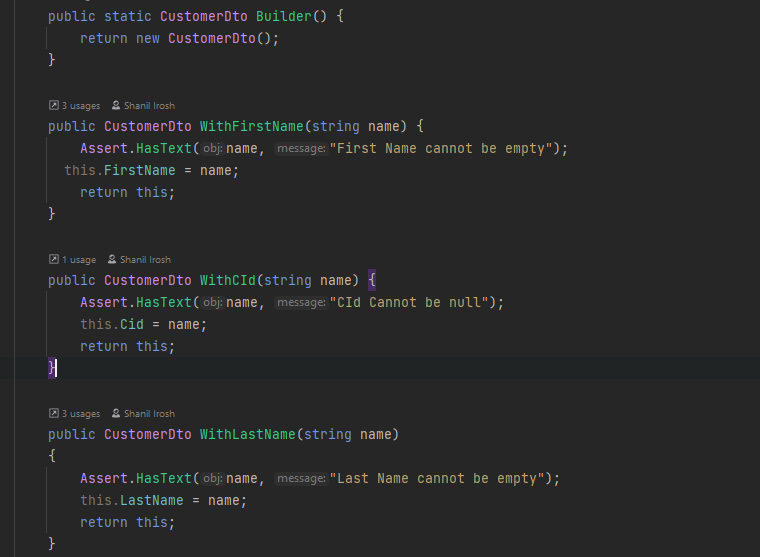
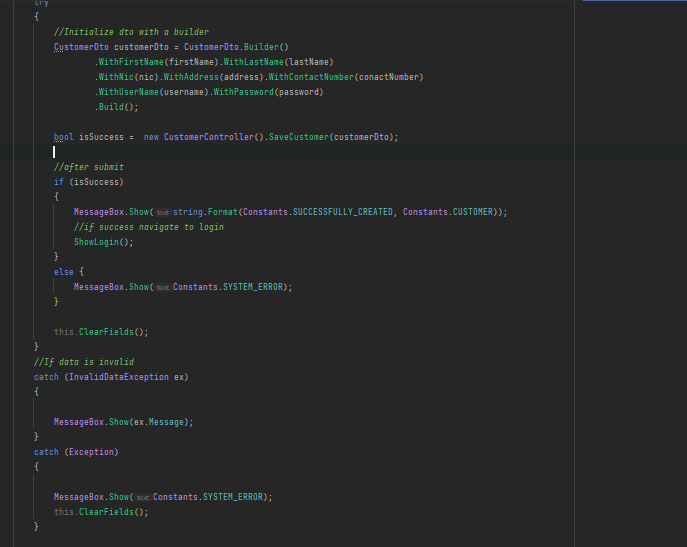


Figure 31

In order to generate business related exception I have used the exception class called **InvalidDataException.**

In the view layer even if a mandotory daata is not parsed from the UI I used this approach. To do that I’ve used the **fluent builder pattern**. In the view layer when I retrieve the data from UI I will create a DTO(Data transfer object). A Dto is built as follows.





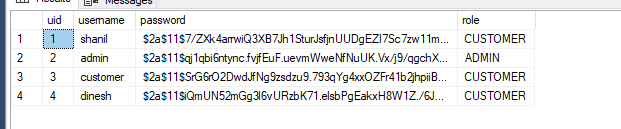
As shown above I have sorrunded it with a try catch and whenever our business logic related custom exception is occurred that messages is shown to the client as it is business related (InvalidDataException catch block).

But if a null pointer,SQL exception is ocured we cant show it to the client as it it irrelavant so in that kind of a scenario it will go the Exception catch block (As exception is the parent class of all exceptions) and there a Common, System Error message is shown to the client

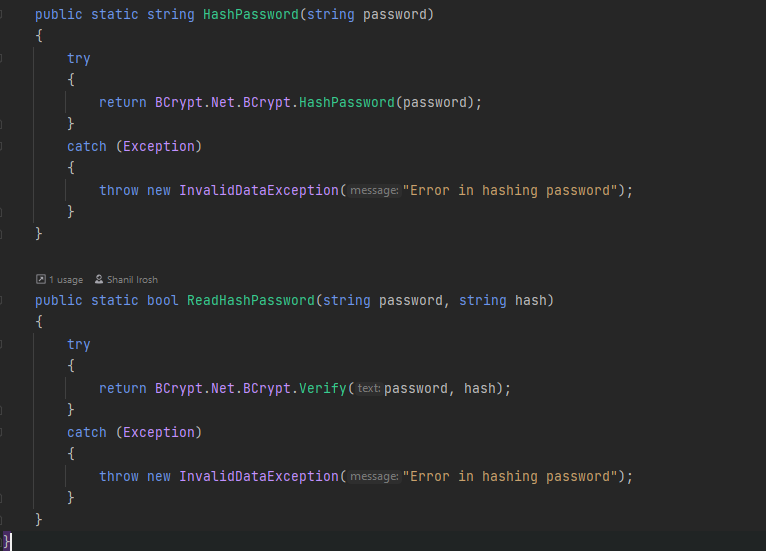
## Password Hashing

It is not a good practice to keep the passwords in the database as a text. Because if there is a person who has the access to the database he/she can impersonate and do unwanted activities in order to do that I have encrypted the passwords through a package called Bcrypt. All the passwords are stores in the database as a hash. And when logging in that hash is checked.

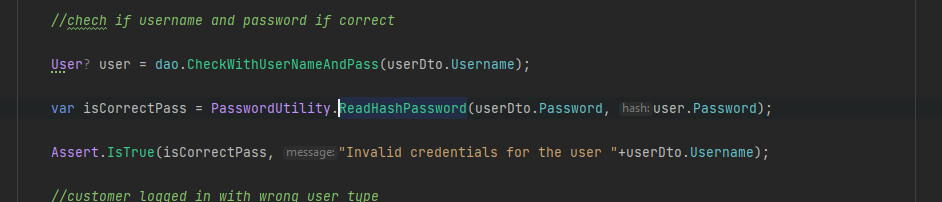
**User Data Table**



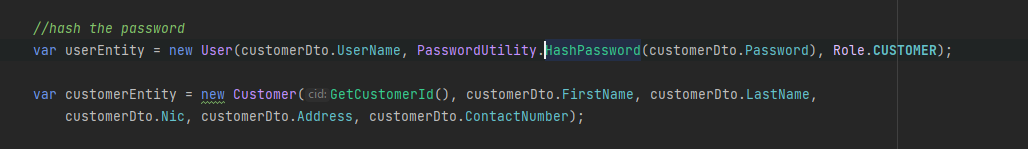
**Utility class used for hashing**



**Checking Whether the client entered password matches with the Database saved hash**



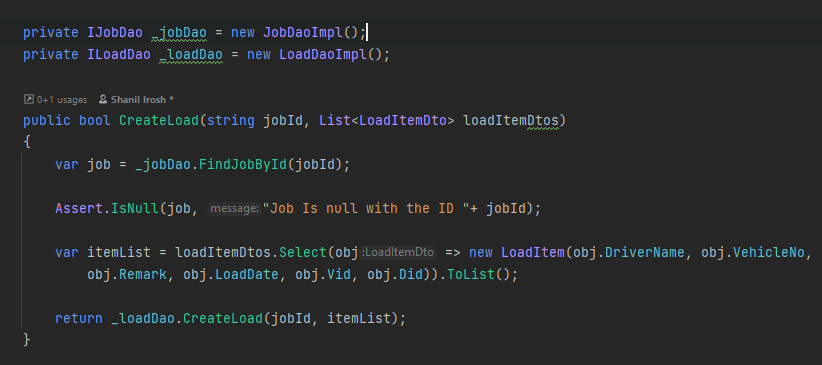
**Hashing the text**



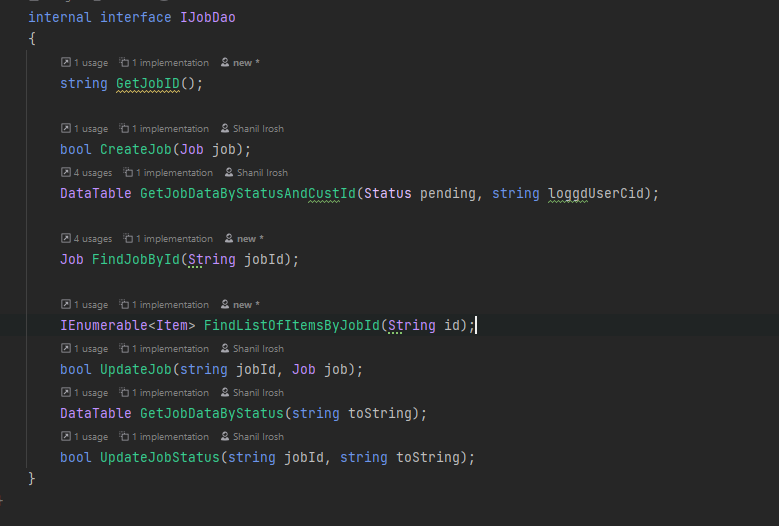
## Low Coupling and High Cohesion

Low coupling is among different classes/modules should be minimal dependency. In order to mitigate this I have used interfaces. And for each and every class the depending class has been **Injected (Dependancy Injection).** An interface works like a contract so that user must adhere the rules of the interface. We can’t change the dependencies as our wish as we use interfaces.

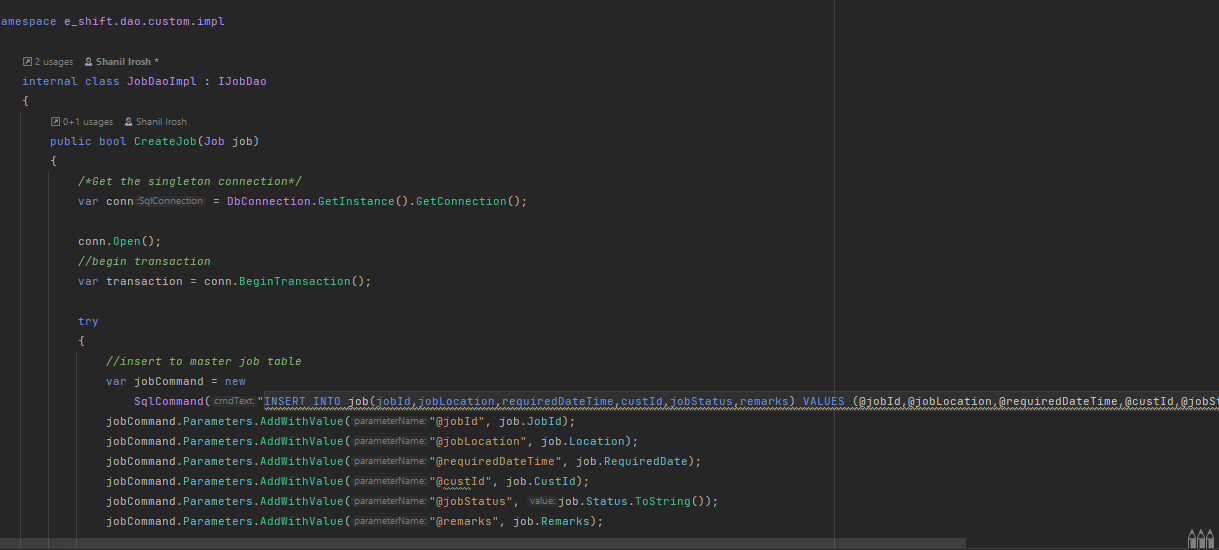
**Using as a Dependency Injection**



**Interface Used**



**Implementation Class**



## Transaction Management

When there are scenarios where two tables records get affected in a single request. And if one table get executed properly and other table didn’t get executed due to a specific reason there might be a problem. Because data is not matched with both the tables.

In this project when we create a Customer for the first time the userdata table and the customer table is getting effected in the same time. So if an exception occur both tables data should be rolled back

In order to solve this problem task management was used

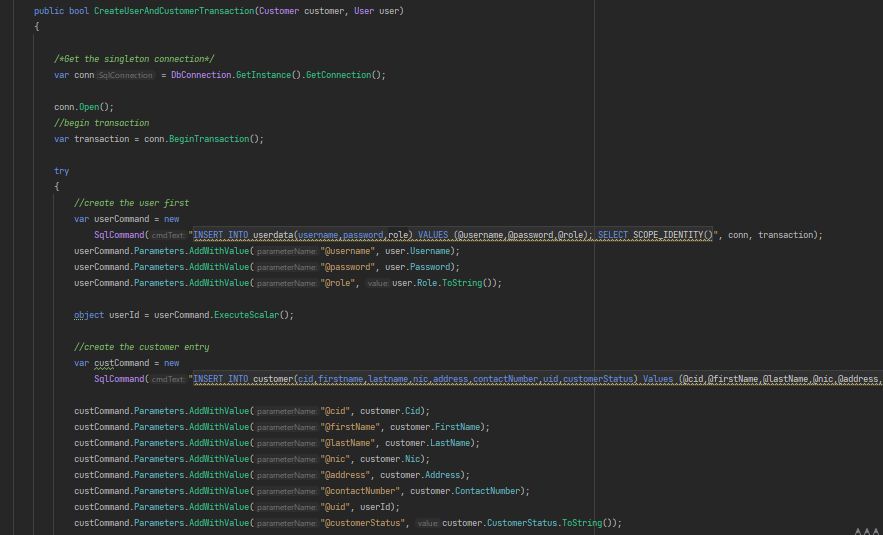


Figure 32

As above after getting the connection BeginTransaction method Is called to ensure transaction is started. Then the subsequent table data is conducted

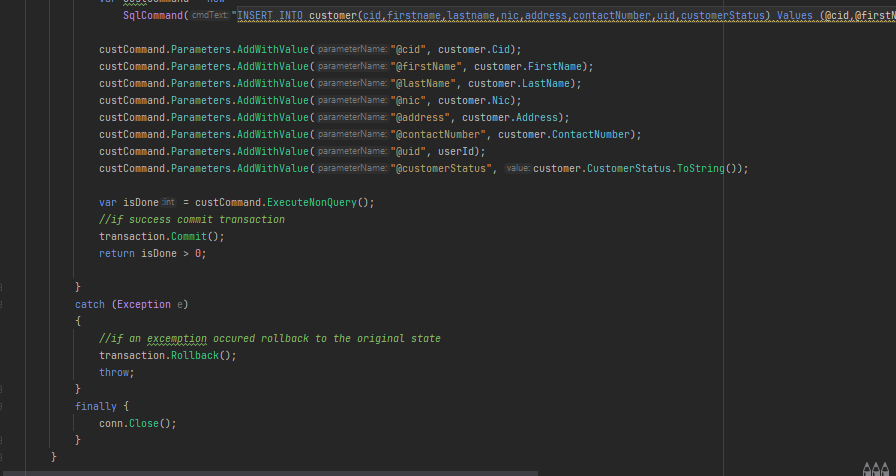


Figure 33

Then if all the database queries has been executed properly the transaction is committed and the data will be persisted in the database.

But if an exception occurred no data must be persisted in the database. So the transaction is rolled back by calling the Rollback() method. So no data is persisted in the db. And in the final block connection is closed no matter an exception occurred or not as finally block is called always before leaving a method.

## References Used

<https://stackoverflow.com/>

<https://www.c-sharpcorner.com/article/crud-operation-in-c-sharp-windows-application-using-store-procedure/>

## Github repository