TOYOTA CAR REPAIR SYSTEM REPORT

158.751 Object Oriented Software Development: Theory & Practice

Team member: Ziyi Wang 18042783 | Rebecca Liu 17238744

Contents

[1 Problem statement 2](#_Toc529285446)

[1.1 Customer 2](#_Toc529285447)

[1.2 Helpdesk 3](#_Toc529285448)

[1.3 Manager 3](#_Toc529285449)

[1.4 Worker 4](#_Toc529285450)

[1.5 Job 4](#_Toc529285451)

[2 Future Enhancement 5](#_Toc529285452)

[3 Design and Implementation 6](#_Toc529285453)

[3.1 Model 6](#_Toc529285454)

[3.2 View 7](#_Toc529285455)

[3.3 DAL & BLL 8](#_Toc529285456)

[3.4 Class Diagram 10](#_Toc529285457)

[4 Data Structure 10](#_Toc529285458)

[5 Lessons learned 11](#_Toc529285459)

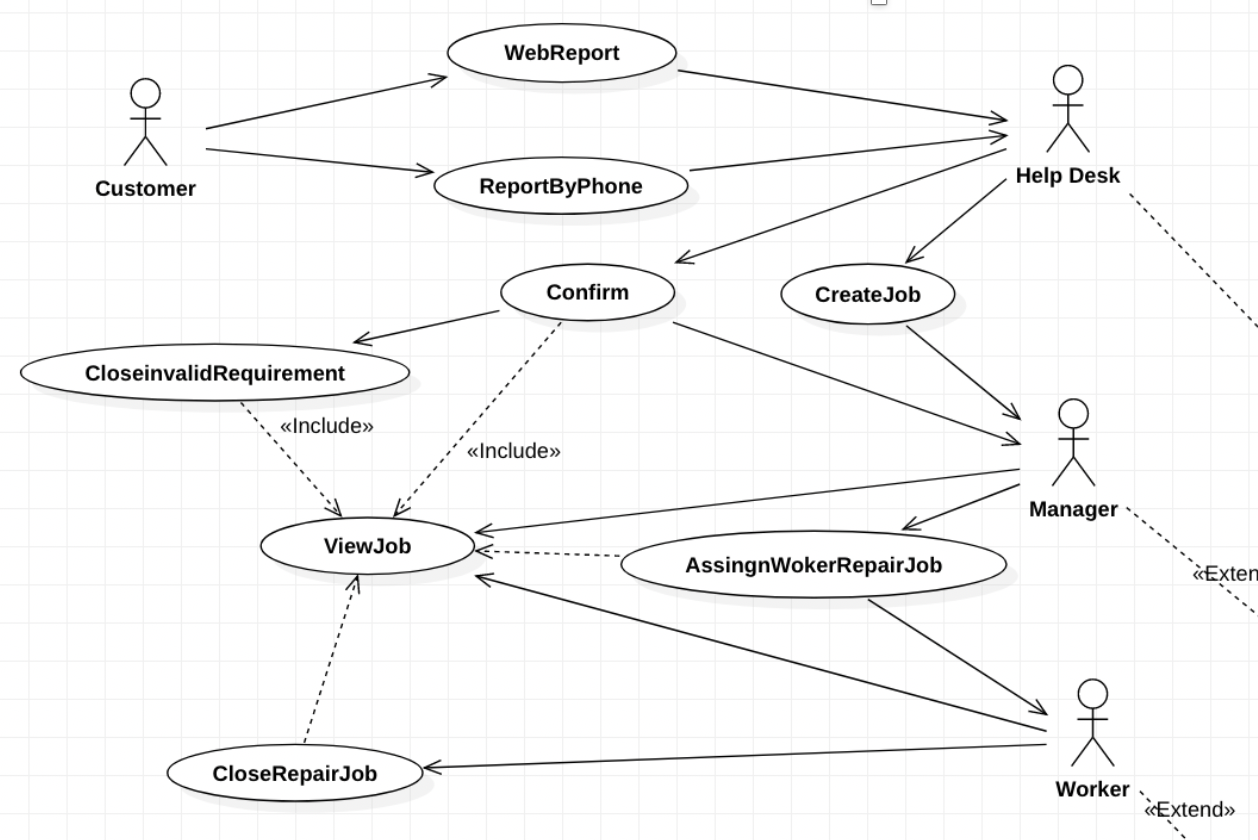
[Appendix A- Initialized Data 13](#_Toc529285460)

[Appendix B- UML Diagram 16](#_Toc529285461)

[Appendix C- Test Users 17](#_Toc529285462)

# Problem statement

Toyota Repairing Services provides a real time solution for the customers who have their car repairing requests. The customer can report their repairing requests online, and, see the reports about the results of their requests. The customer can also report the problems through the helpdesk of any outlet. After the confirmation of the request, a repair job can be assigned to a worker. The job can be closed by a worker after it is repaired. The following picture is the use case diagram of the system.



## Customer

We want the customers can freely choose the way to report their car problem and know the result of their requests in real time. So, we provide the following functions:

1. Customer can report the car repairing request online.
2. Customer can report the car repairing request through helpdesk by phone or email, etc.
3. Customer can see the result of the repairing request in real time through online service.

## Helpdesk

We want the helpdesk can record the repairing jobs fluently, and, know clearly about the jobs they need to confirm. So, we provided the following functions:

1. Helpdesk can add a job through the system if they receive the repairing job through outside the system.
2. Helpdesk can check the job details and confirm it if they saw an unconfirmed job reported by the customer through the system.
3. Helpdesk can update a job through the system.

## Manager

We want the manager can know clearly about how many jobs to be assigned, and, manage the jobs fluently. So, we provided the following functions:

1. Manager can assign/reassign jobs to the workers belong to him.
2. Manager can set the job’s priority.
3. Manager can see the report of a list of open/closed jobs assigned to a particular employee, a list of open/closed jobs, a list of jobs related to one particular car, a list of workers and the number of open jobs they have.

## Worker

We want the worker can know clearly about what they need to do, and, can show what they have done in real time in the system. So, we provided the following functions:

1. Worker can close (resolve) a job.
2. Worker can see a list of open/closed jobs assigned to him, a list of jobs related to one particular car.

## Job

The jobs are dealt in the following work flow.

# Future Enhancement

In this project, we just implemented the basic workflow of the Repair Service System. Some enhancements can be done in the future.

1. System administration

We can develop a department management function, in the future, to match the development of the organization. There will be more workers and roles.

Some more management parts can be added. For example, roles and functions configuration, system field’s configuration, and data access control, etc.

1. Sign up

Now the customers are the initialization data of the system. In the future, we can add the signup function to get more and more users involved more conveniently.

1. More flexible workflow settings & strict control

Now the workflow is fixed, as shown in 2.5. We can define different workflows for different kinds of jobs. In addition, the system lacks strict control of the job process. For example, if the job comes to resolving stage, the information of the job cannot by modified, etc.

1. Better UI design

The system’s UI is functional. We can design friendlier interface to make it more enjoyable to use.

1. More analysis reports

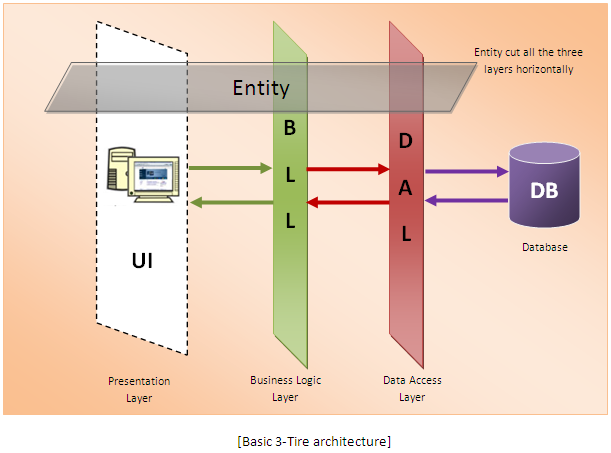
We can add more graphs for better process management and quality control when the business becomes more complex. For example, the cost control chart, burnout chart, etc.

1. Feedback

We can add feedback function for the user to evaluate the quality of the services.

# Design and Implementation

We use Three-tier architecture in our coding design. Our application is separated into the following three components. Basically 3-Tier include presentation layer, business logic layer and data access layer. In our project we design like UI layer, BLL layer DAL layer and Model layer.



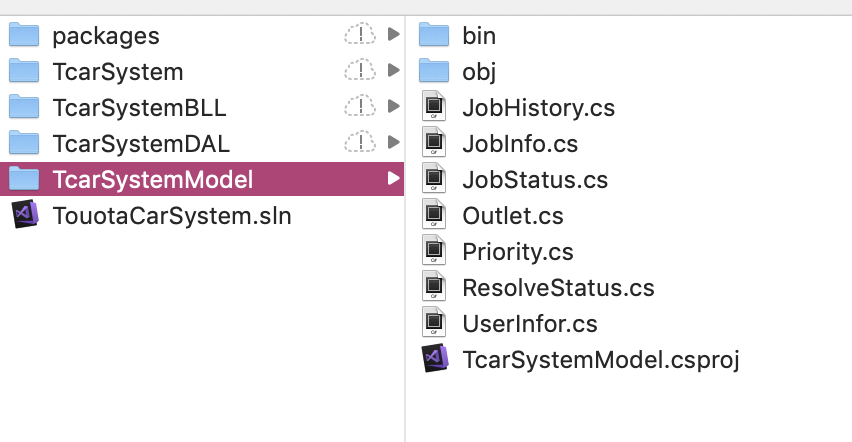
( https://stackoverflow.com/questions/13786549/how-to-implement-3-tiers-architecture-in-c-sharp )

## Model

Model defines all the data we use in this application. We have TcarSystemModel folder to put all of our model classes in. The major classes in this layer is JobInfo and UserInfor which are also the main tables in our database.

Some configuration items, like JobStatus, Priority, ResolveStatus, are defined as model classes which are enum-type. The rule here is that, if we do not have to change the contents of the data fields, we put them here rather than in the database.

Some model classes, like Outlet, are used for the foreign key in our application.

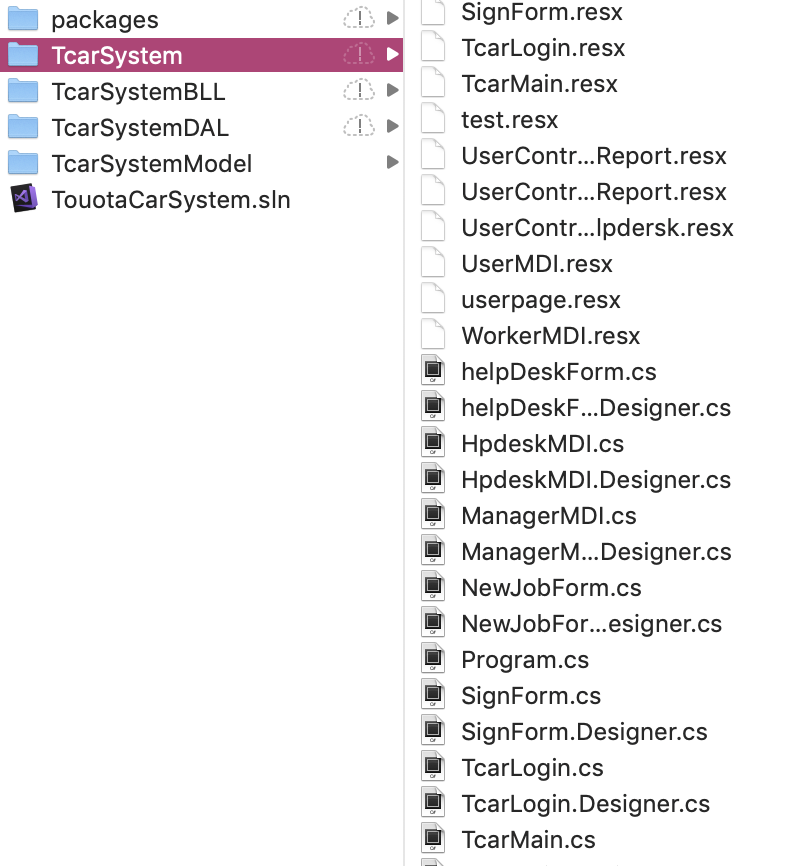


## UI

View includes all of our user interface class which is the TcarSystem folder in our application.

We put all of our window forms here. We separated the forms for different roles. For instance, we have the helpdesk form, worker form, manager form, and customer form. In this way, we can:

1. Reduce the complexity and the data traffic of putting all the things in one page. This can improve the performance of the page.
2. Generate loose coupling so that we can separate the tasks to different team player to finish. Besides, it is good for expanding different functions for different roles.
3. When we modify something, it will not influence other forms.



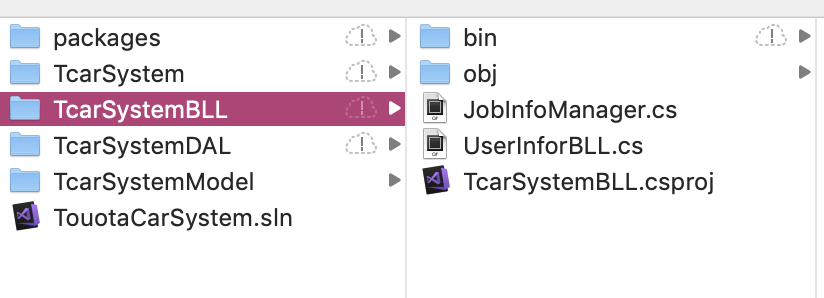
## DAL & BLL

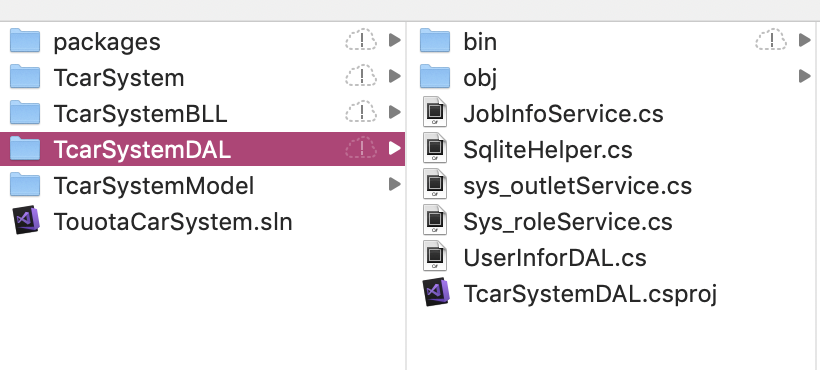
Data exchange layer is the bridge between the model and the UI. It contains all the business operations. We abstract all the actions in our applications to this layer using the Model to response the requests of the UI layer.

TcarSystemBLL as an entry point for the UI layer and the underlying data exchange, and TcarSystemDAL getting data from BLL layer, also this layer interact data with database convert the data to object.

In DAL folder, there are two major classes, JobInfoManager and UserInforDAL which contain the basic operations to the related database tables, for example, the adding, searching, updating, and deleting. SqliteHelper is the support functions of the database operation. The other class’ functions are based on this. In additions, we put several classed for using foreign keys, like sys\_OutletService and Sys-roleService which is used to relate the job table and the user table, outlet table.

In BLL folder which is closer to UI layer than DAL, there are the encapsulations for DAL classes, for example, JobInfrManager, UserInforBLL. These are used by the View forms directly.





The benefits of having BLL and DAL:

1. The lower layer has the multi-use functions which reduce the maintenance costs come from the modified database or model classes. And easily update the new technology stack.
2. Improve the code reusability, and, simplify the complexity of codes in higher layers. Also it be able to scale the application up and out, and high independence reliability be adds on the underlying servers or services.
3. The programing can be done from both sides which benefits group works in parallel.

## Class Diagram

Copy from visual studio

## Completed function

### Login page

|  |  |
| --- | --- |
| Location | Login |
| Function Name | User login |
| Function Instruction | Insert user name and password click submit will automatically login in correct user page by user identity. |
| Function Description | User insert user name and user password in login page click submit button, system will automatically verify user’s name, password and user identity in database open the corresponding user page at same time. |

### 3.5.2 Customer Page

|  |  |  |  |
| --- | --- | --- | --- |
| Location | Function Name | Function Instruction | Function Description |
| Customer Page | Customer Home Page | 1. After user login user page the user repair list will automatically on the home page. Customer can check the report list and click button to add report job. | This function is for user to check all the report repair jobs in Toyota car repair system. |
| Add Report page | Add report | 1. Car No: customer need write down their report car plate number  2. Outlet: select which outlet customer want to go  3. Type: customer need select repair job type which include Panel Beating, Electrical and Mechanical.  4. Job Description: customer can write down the problem about this car roughly. | Customer add their report on this page. All the report will sent to our database and the status will automatically be unconfirmed which need outlet helpdesk open this job status, this function can easy help Toyota outlets to inspect repeated job requirement. |
| Modify Report Page | Modify report | modify page is same as add report page when customer need to modify their unconfirmed report job just select report row in report list on home page, | This function only support unconfirmed job status. |

### Help Desk Page

|  |  |  |  |
| --- | --- | --- | --- |
| Location | Function Name | Function Instruction | Function Description |
| Home Page | Help Desk Home | 1. Help desk Pending work list.  2. Add job  3. Modify jobs  4. Search | Home page support help desk check pending work list, modify work, and search all the job in this help desk outlet. |
| Add Job | Add new works | Add new works which need Car No, Name, Type, Job Description and job status. | Help Desk create a new job will be automatically in open status. |
| Modify page | Modify Jobs | Modify job content by select on home page | Help desk can modify job on this page |
| Search page | Search all jobs | Select by job status and resolve type | Search all the job under this help desk. |

### Manager page

|  |  |  |  |
| --- | --- | --- | --- |
| Location | Function Name | Function Instruction | Function Description |
| Home page | Manager home page | 1. Manager pending job list  2. Assign job  3. Search | Home page support manager check pending work list, assign job to work, and search all the job in this manager belong outlet. |
| Assign job | Assign jobs to work | Select job on home page and assign jobs to worker or change worker in jobs | Assign job to worker in current outlets. |
| Search page | Manager search works | Select all the jobs in current outlet by job status, resolve and worker | Search total work list on this outlet. |

### Worker page

|  |  |  |  |
| --- | --- | --- | --- |
| Location | Function Name | Function Instruction | Function Description |
| Home page | Worker home page | 1. List of job which is assign to current worker by manager.  2. Close job form  3. Search job | Home page job list show all the job witch assign to current worker. |
| Close job page | Close job | When worker finish job need to close job on there | Only worker can close job on this page. |
| Search page | Search job | Search all the job which assign to current wokrer | Work search all the current job. |

# Data Structure

Test data - Create tables:

CREATE TABLE Outlets (  
    otID    INTEGER PRIMARY KEY NOT NULL,  
    otAddress TEXT NOT NULL,  
    otName    TEXT NOT NULL  
);  
  
CREATE TABLE user (  
    userId    INTEGER PRIMARY KEY NOT NULL,  
    user\_name    TEXT DEFAULT '',  
    carNo    TEXT DEFAULT '',  
    user\_password    TEXT NOT NULL,  
    identity    TEXT DEFAULT '',  
    Outlet    INTEGER,  
    FOREIGN KEY(Outlet) REFERENCES Outlet(otID)  
);  
  
CREATE TABLE jobs (  
    id    INTEGER PRIMARY KEY NOT NULL,  
    closedate TEXT DEFAULT '',  
    createdate    TEXT NOT NULL,  
    carNo    TEXT DEFAULT '',  
    desk    INTEGER,  
    manager    INTEGER,  
   worker    INTEGER,  
    jobDescription    TEXT DEFAULT '',  
    resolve    INTEGER,  
    priority INTEGER,  
    comment     TEXT DEFAULT '',  
    outlet INTEGER,  
    customer INTEGER,  
    jobStatus INTEGER,  
    opendate TEXT DEFAULT '',  
    assigndate TEXT DEFAULT '',  
    FOREIGN KEY(customer) REFERENCES user(userId),  
    FOREIGN KEY(worker) REFERENCES user(userId),  
    FOREIGN KEY(manager) REFERENCES user(userId),  
    FOREIGN KEY(desk) REFERENCES user(userId),  
    FOREIGN KEY(outlet) REFERENCES Outlets(otID)  
);

Other test data is in Appendix A.

# Lessons learned

Both of our team members do not have any developing experiences before. We have learned a lot from doing this project.

Firstly, we use our in-class knowledge of use case diagram to describe the system we need to do. We used system sequence diagram (SSD), use case diagram.

Secondly, we try to use 3-Tier architecture in our project. We have low coupling and high cohesion classes in the lower layers, such as Model classes, DAL classes, and BLL classes. In these layers, each class deal with all the business around it. For example, JobInfoManager which is in BLL folder has all the job-related functions, such as, adding, searching with certain conditions, updating, etc.

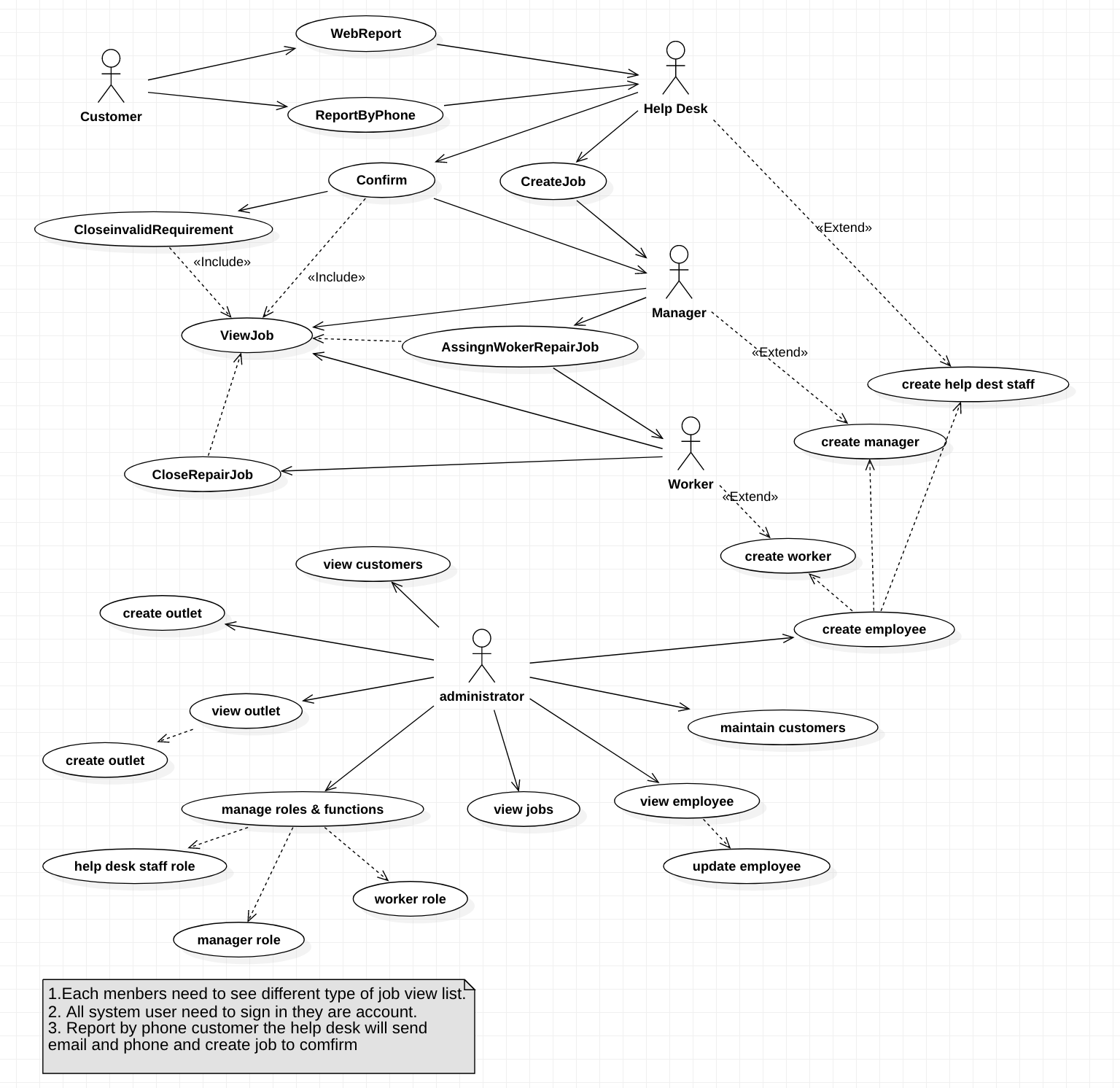
Thirdly, we practiced how to use Visual Studio. We used it to write C# application. Then, we use SQLite to store our data. We encountered some configuration problems, such as Sys.Data.SQLite version problem. We tried several versions and followed some forum contents to modify our project settings to solve such kind of problems.

Lastly, we also improved our first design of the classes in our implementation. The new class diagram is shown in 4.4.

In addition, in this group project, we encountered version control problems. We chose git to manage our code, to deal with the conflicts. We set up some rules like when to submit our codes and what kind of files we need to submit, etc. Through this, we understand better about the collaboration in a small developing team.

Appendix A- Initialized Data

# Appendix B- UML Diagram



# Appendix C- Test Users

|  |  |  |  |
| --- | --- | --- | --- |
| Role | Use Name | Password | Actions |
| Customer |  |  | Add a new repair job |
| Help Desk |  |  | Add a new repair job  Update a job  Confirm a job  Close an invalid job |
| Manager |  |  | Assign a job & set the priority |
| Worker |  |  | Close a job |