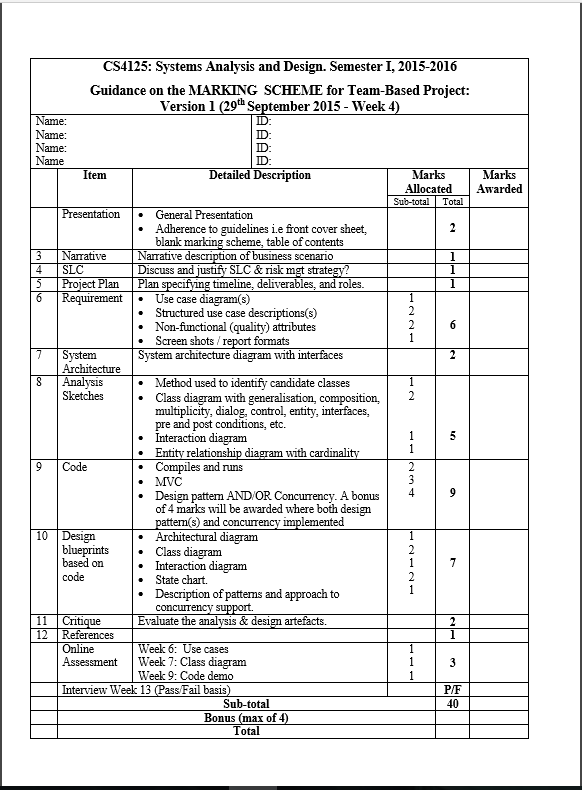
|  |
| --- |
| **CS4125 Systems Analysis & Design** |
| Console Collaboration |
|  |

|  |
| --- |
|  |



# Table of Contents

[Table of Contents 2](#_Toc436654558)

[Description 3](#_Toc436654559)

[Software Lifecycle Model 4](#_Toc436654560)

[Waterfall model: 4](#_Toc436654561)

[V-model: 4](#_Toc436654562)

[Agile Software Development: 4](#_Toc436654563)

[Project Plan and Allocation of Roles 6](#_Toc436654564)

[REQUIREMENTS 7](#_Toc436654565)

[Use Case Diagrams 7](#_Toc436654566)

[Use Case Description: 9](#_Toc436654567)

[Non-functional Requirements 9](#_Toc436654568)

[Screen Shots 11](#_Toc436654569)

[Observer pattern for multiple watched games 13](#_Toc436654570)

[System Architecture 14](#_Toc436654571)

[Class Diagram 15](#_Toc436654572)

[Interaction Diagram 16](#_Toc436654573)

[Entity Relationship 17](#_Toc436654574)

[MVC 18](#_Toc436654575)

[Architectural Diagram 19](#_Toc436654576)

[Class Diagram (Associations) 20](#_Toc436654577)

[Class Diagram (Dependencies) 21](#_Toc436654578)

[Class Diagram (Inheritance) 22](#_Toc436654579)

[State Chart 23](#_Toc436654581)

[Descriptions of Patterns 24](#_Toc436654582)

[Factory 24](#_Toc436654583)

[Observer 24](#_Toc436654584)

[Critique 25](#_Toc436654585)

[References 26](#_Toc436654586)

# Description

**Overview**

As the gaming market continues to grow yearly and the graduation of games being sold as hardcopies to being downloaded from the user’s home (softcopy) one thing is clear, these softcopies are here to stay. While there are pros and cons for each I believe that in the future softcopies will emerge victorious just as disc triumphed over cassette. It is hard to find many benefits with hardcopies especially now on the next generation consoles where even if you buy a hardcopy of a game you will still need to install it in order to play the game. It is only for segmental value or perhaps special editions of games where people are buying hardcopies.

On the other hand softcopies offer a broad number of advantages. Firstly most gaming companies are allowing their games to be downloadable before release so that on the official launch of the game the game will be ready and the player can play immediately. Another benefit of softcopies is that while discs can be scratched a softcopy of the game is linked to your profile so it will always be your game. One of the biggest factors that promote the purchase of softcopies is that companies whose games are sold through software middle man businesses such as steam often have games on sale or reduce the cost of a game for a week or two.

While steam is a pc middle man business and has become a huge success in the pc community there is no real business in the console community. While each console has its own software marketplace the deals and sales that make steam so successful are lacking to be found on these marketplaces. This is where console collaboration comes into effect.

**What is console collaboration?**

Console collaboration is a console cross platform marketplace which will allow players to buy games on their console of choice with more completive prices than the consoles official marketplace. Console collaboration will act like a steam based marketplace and will have frequent sales and special offers to entice members to buy more games.

# Software Lifecycle Model

While discussing which software lifecycle model to use for our project we looked at three different types first the waterfall model, second the V-model, & third agile development. We decided to choose the agile development as both the waterfall and V-model had some flaws that made them unsuitable for our project.

# Waterfall model:

\* If we were to use the waterfall model our project would have to follow a sequential flow i.e. cannot make changes later on as it could lead to confusion later on.

\* Since all phases are dependent on each other any change on a previous phase can cause big problems for subsequent phases.

\* The waterfall model has an emphasis on up front requirements & design, but since most customers only figure out what they want through interaction with the designers, it is unrealistic and unsuitable.

# V-model:

\* Software is developed during the implementation phase, so no early prototypes of the software are produced

\* Very rigid and inflexible.

\* If any changes happen in midway, then the test documents along with requirement documents has to be updated

# Agile Software Development:

Agile development uses 12 guiding principles that are designed to satisfy the customer through early and continuous delivery of valuable software. Principles include:

\* Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

\* Working software is the primary measure of progress.

\* The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

\* The best architectures, requirements, and designs emerge from self-organizing teams.

\* At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

With all the criticisms for the waterfall & V-model, we decide to use the approach that promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to change. This means that by using agile software, if the customer decided to add a feature later on it would be possible.

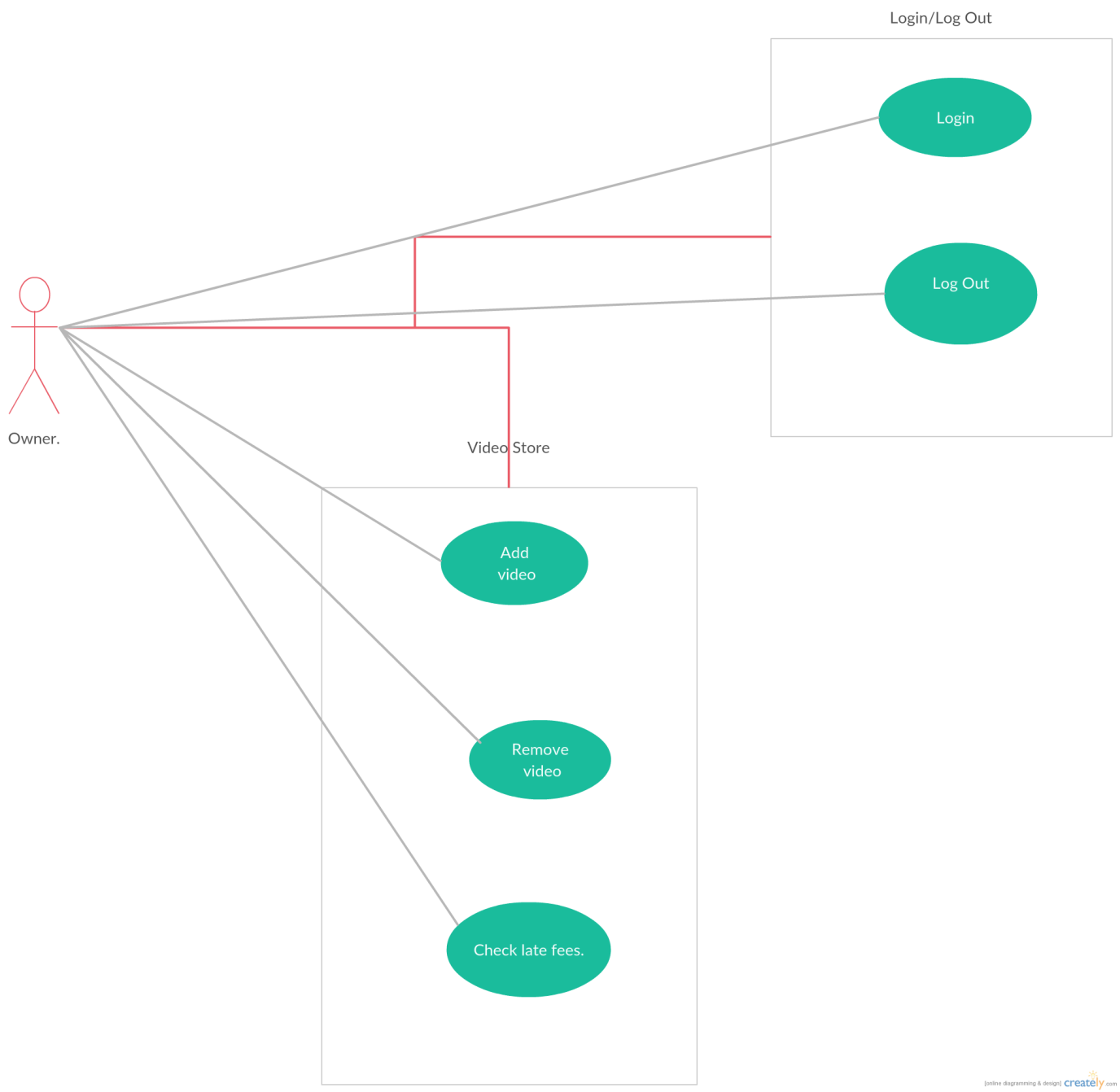
Agile software development focuses on keeping code simple, testing often, and delivering functional bits of the application as soon as they're ready. The goal of agile software development is to build upon small client-approved parts as the project progresses, as opposed to delivering one large application at the end of the project.

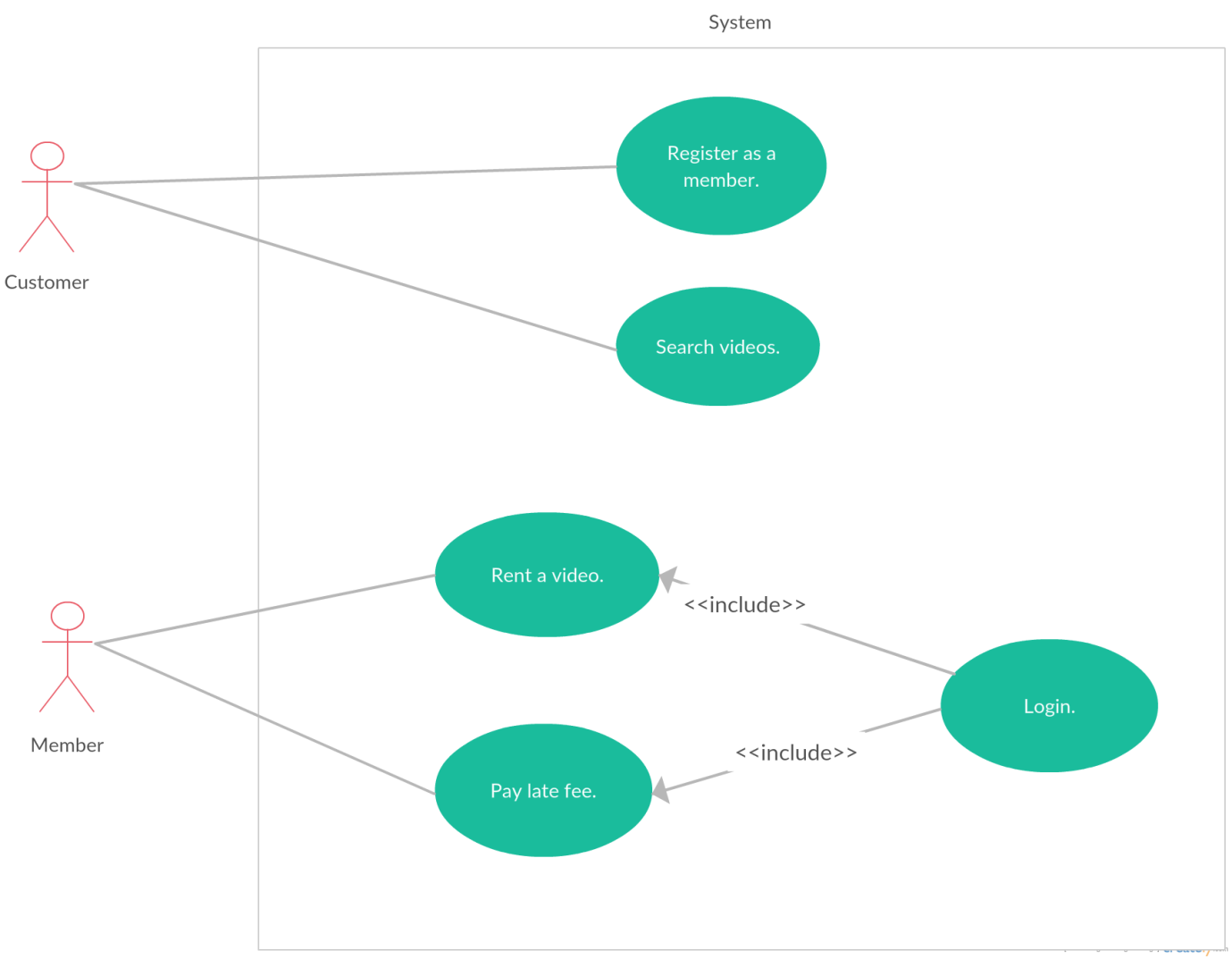
# Project Plan and Allocation of Roles

|  |  |  |  |
| --- | --- | --- | --- |
| Deliverable | Description |  | Week |
| Presentation |  |  |  |
| Narrative | Narritive description of business | Pat | 4 |
| Software LifeCycle | Discussion on software model used | Shane | 5 |
| Project Plan | Specifying times and roles | Group | 4 |
| Requirments | Use Case Diagram  Use Case Description  NFR’s  Screen Shots | Shane | 7  6  8  12 |
| System Architecture | Architecture diagram with interfaces | Pat | 7 |
| Analysis Sketches | Identify Classes  Class Diagram  Interaction Diagram  Entity Relationship | Group | 6  7  7  8 |
| Code | Implementation of code | Group | 8-12 |
| Design | Architectural Diagram  Class Diagram  Interaction Diagram  State Chart  Patterns Description | Shane | 12  12  12  12  12 |
| Critique | Critique on quality of analysis and design | Group | 12 |
| References | Sources use for information | Group | 4-12 |

# REQUIREMENTS

# Use Case Diagrams





# Use Case Description:

# Non-functional Requirements

|  |  |
| --- | --- |
| Use Case:01 | Login |
| Actor Action | System Response |
| 1. User enters Information (username & password) & clicks login | 1. System validates user log in information.   System displays user’s main screen |
| Alternate Action | |
| 1. User enters Information (username & password) & clicks login | 1. System does not recognize username or password. Display error message. |

Non-functional Requirement: Security

The system should encrypt the user’s info and store it securely.

|  |  |
| --- | --- |
| Use Case:02 | Log Out |
| Actor Action |  |
| 1. User clicks log out | 1. System displays log out screen. |

Non-functional Requirement: Security

The system should end the session duration and log out the user.

|  |  |
| --- | --- |
| Use Case: 03 | Add Games |
| Actor Action: | System Response |
| 1. User adds games to store catalogue. | 1. System is updated to show new games have been added. |
| Alternate Action | |
| 1. User adds games to store catalogue | 1. System already has the game in its catalogue. Display Error message |

Non-functional Requirement: Reliability

The system should update its content and display to the user.

|  |  |
| --- | --- |
| Use Case: 04 | Remove Games |
| Actor Action: | System Response: |
| 1. User removes games from store catalogue. | 1. System is updated to show games have been removed. |
| Alternate Action: | |
| 1. User removes games from store catalogue. | 1. System cannot find game by that name. Display Error message. |

Non-functional Requirement: Reliability

The system should update its content and display to the user.

|  |  |
| --- | --- |
| Use Case: 05 | Check Late Fees |
| Actor Action: | System Response: |
| 1. User selects option from main screen. | 1. System displays info of all customers who owe money. |

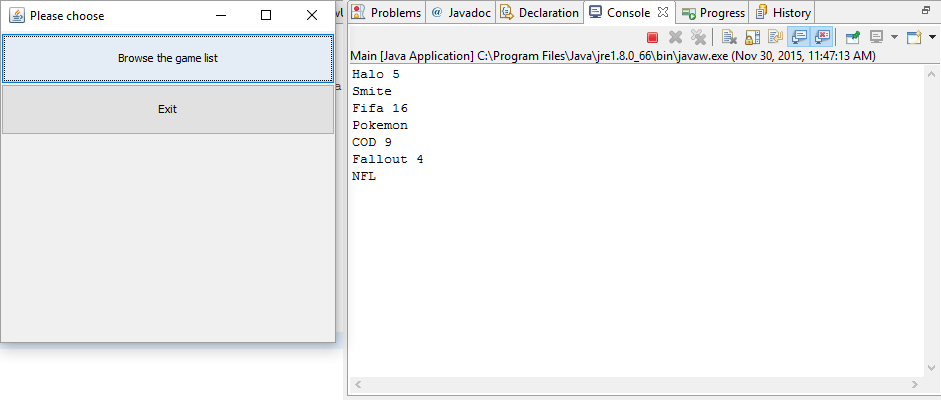
|  |  |
| --- | --- |
| Use Case: 06 | Register |
| Actor Action: | System Response: |
| 1. User enters information (username & password) & clicks register. | 1. System creates account for new user & stores information. |
| Alternate Action: | |
| 1. User enters information (username & password) & clicks register. | 1. System already has an account using that information. Display Error message. |

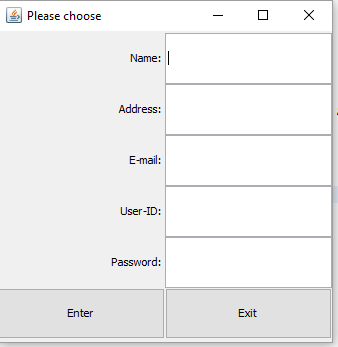
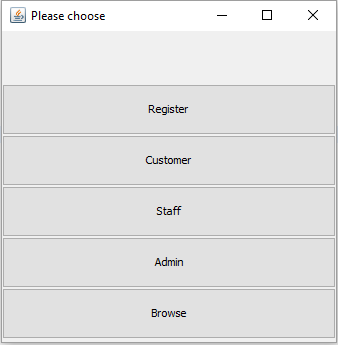
Non-functional Requirement: Security

The system should encrypt the user’s info and store it securely.

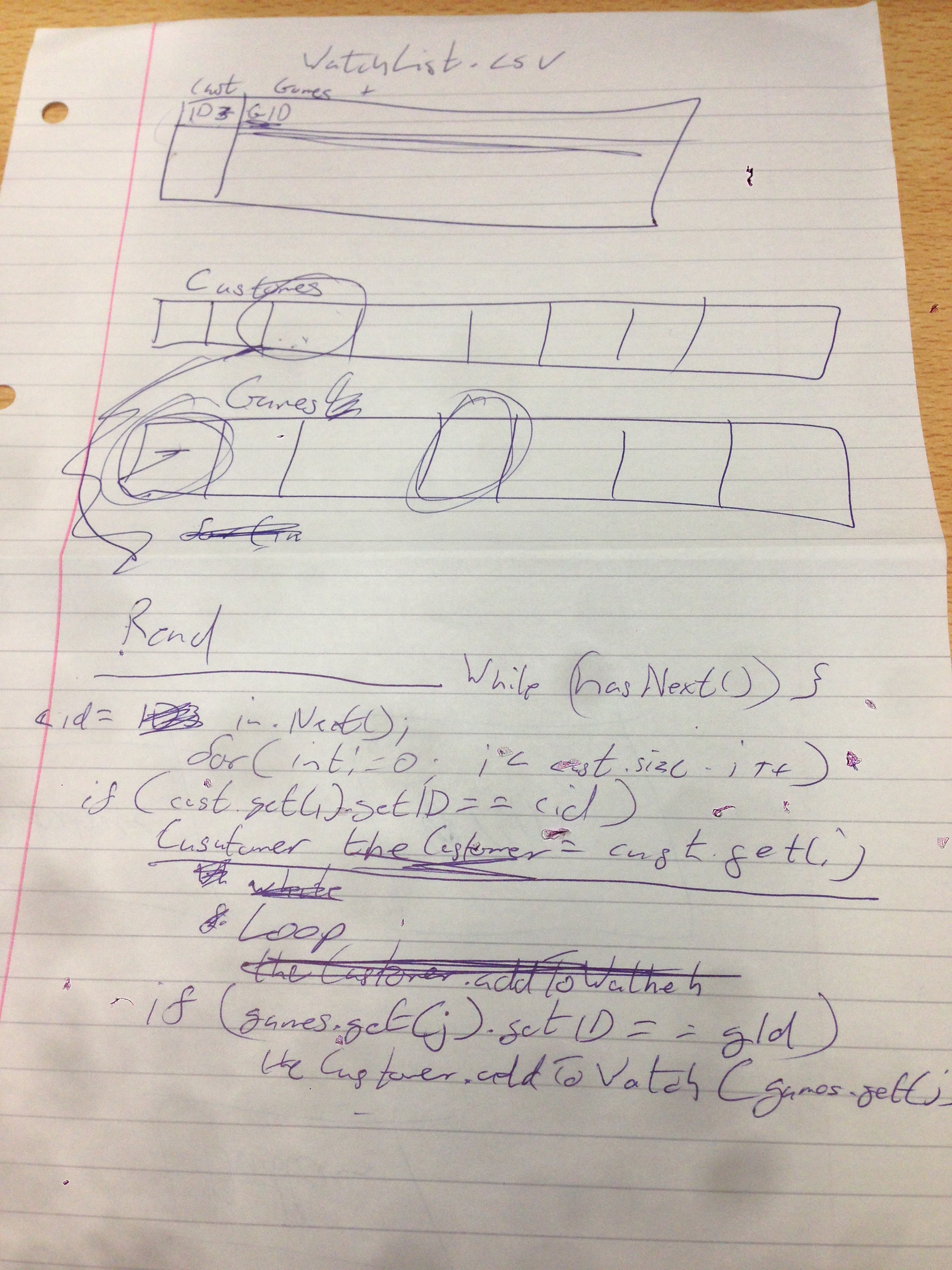
# Screen Shots

Browse

****



# Observer pattern for multiple watched games



# System Architecture

UML Workbench

We chose to use creately and lucid chart as our UML’s for this project. We explored other options but ended up choosing these workbenches. However we did discover that these workbenches had their own set of problems.

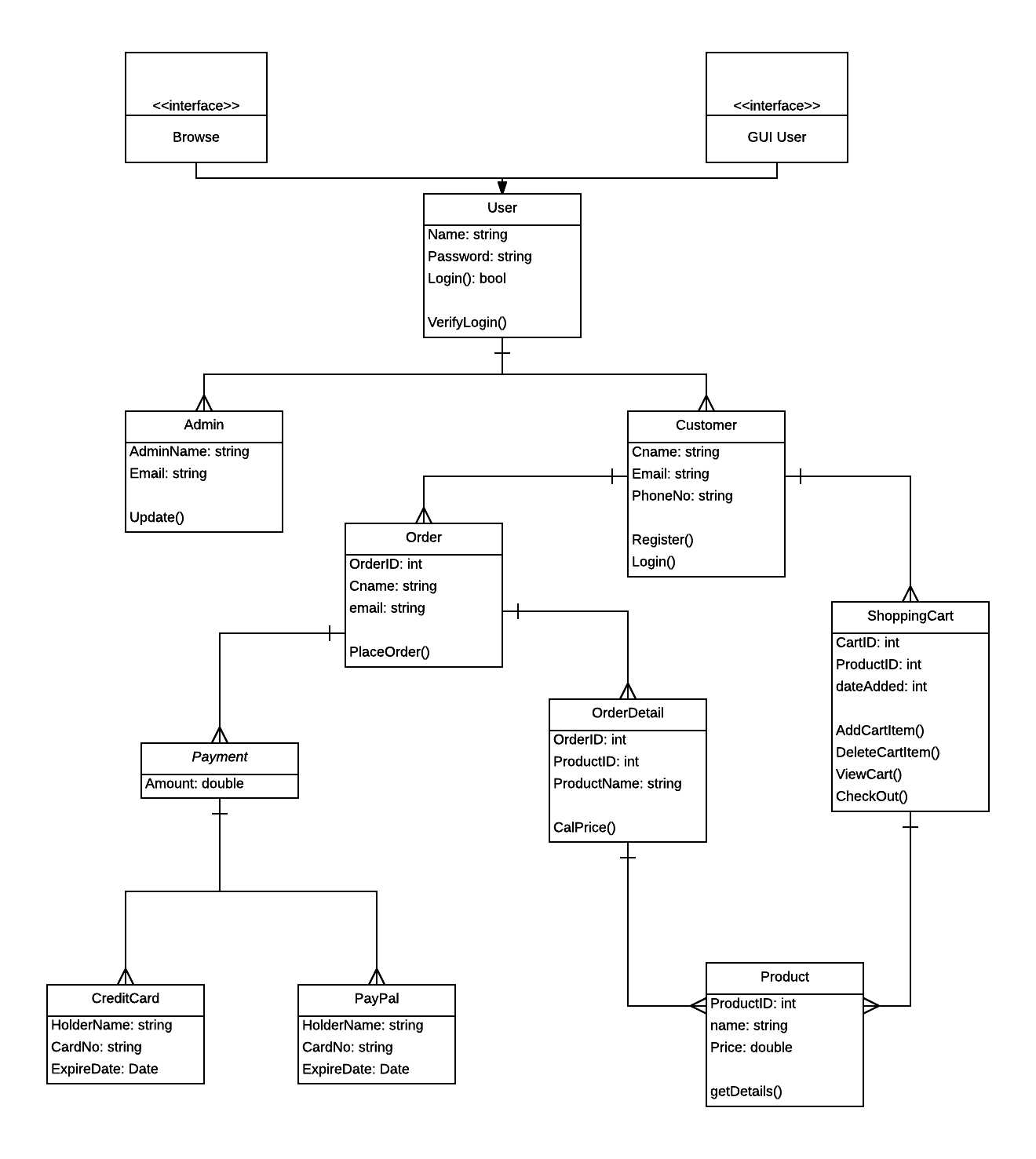
Lucid chart is difficult to set up but offers a lot of tutorials and skeleton diagram patterns which can be helpful. How it does not allow the user to add dialog or interfaces to class diagrams. Also there is a size constraint on how much you can fit on a sheet and you cannot link different sheets so large projects will suffer.

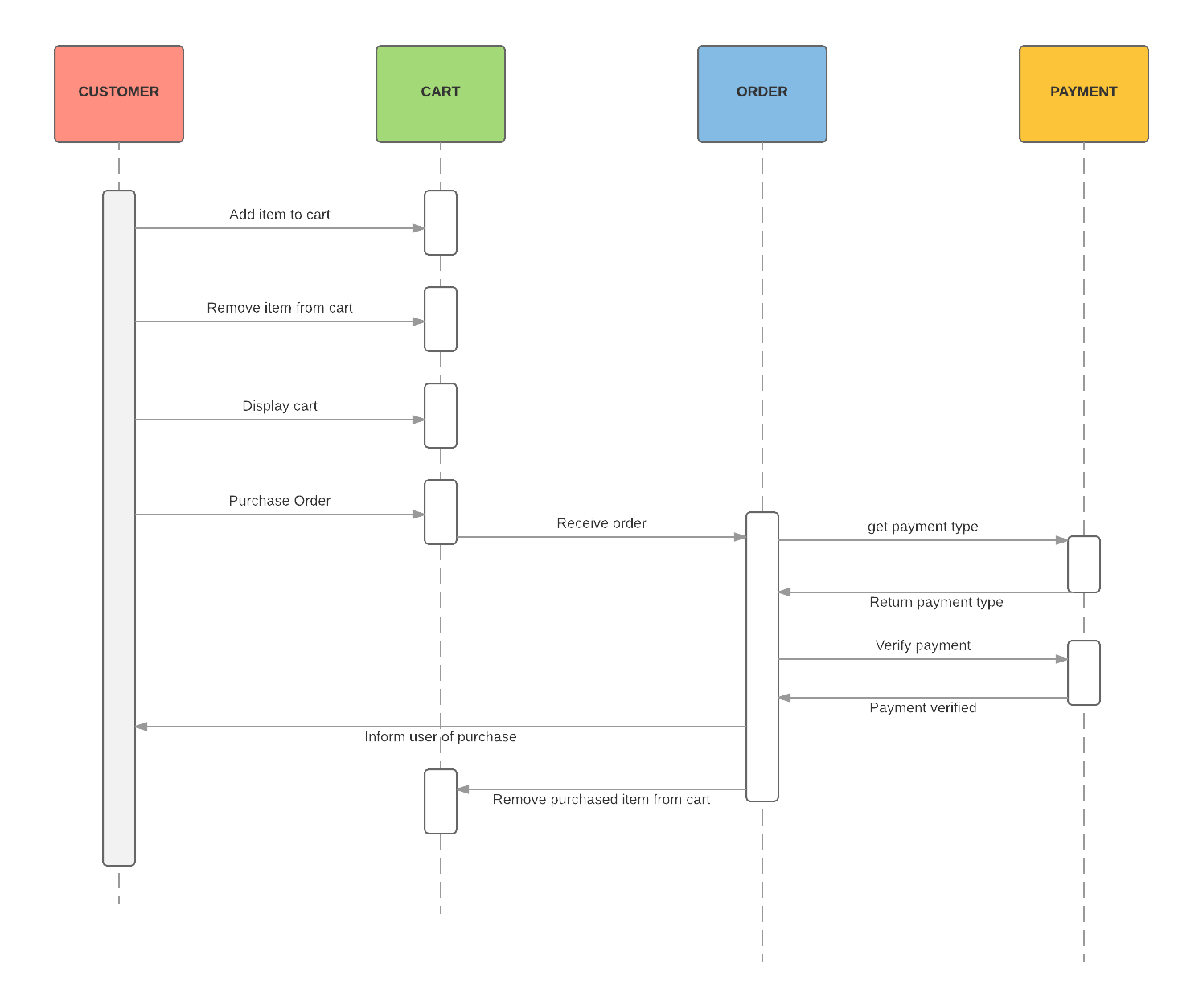
Patterns Description

There are a two design patterns in our project at current with the possible implication of adding the singleton if we have the time to do so. The design patterns that we do have are the observer and the factory patterns. The factory pattern is a creational design pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This makes the project easier to set new users and all the options current users have they will have.

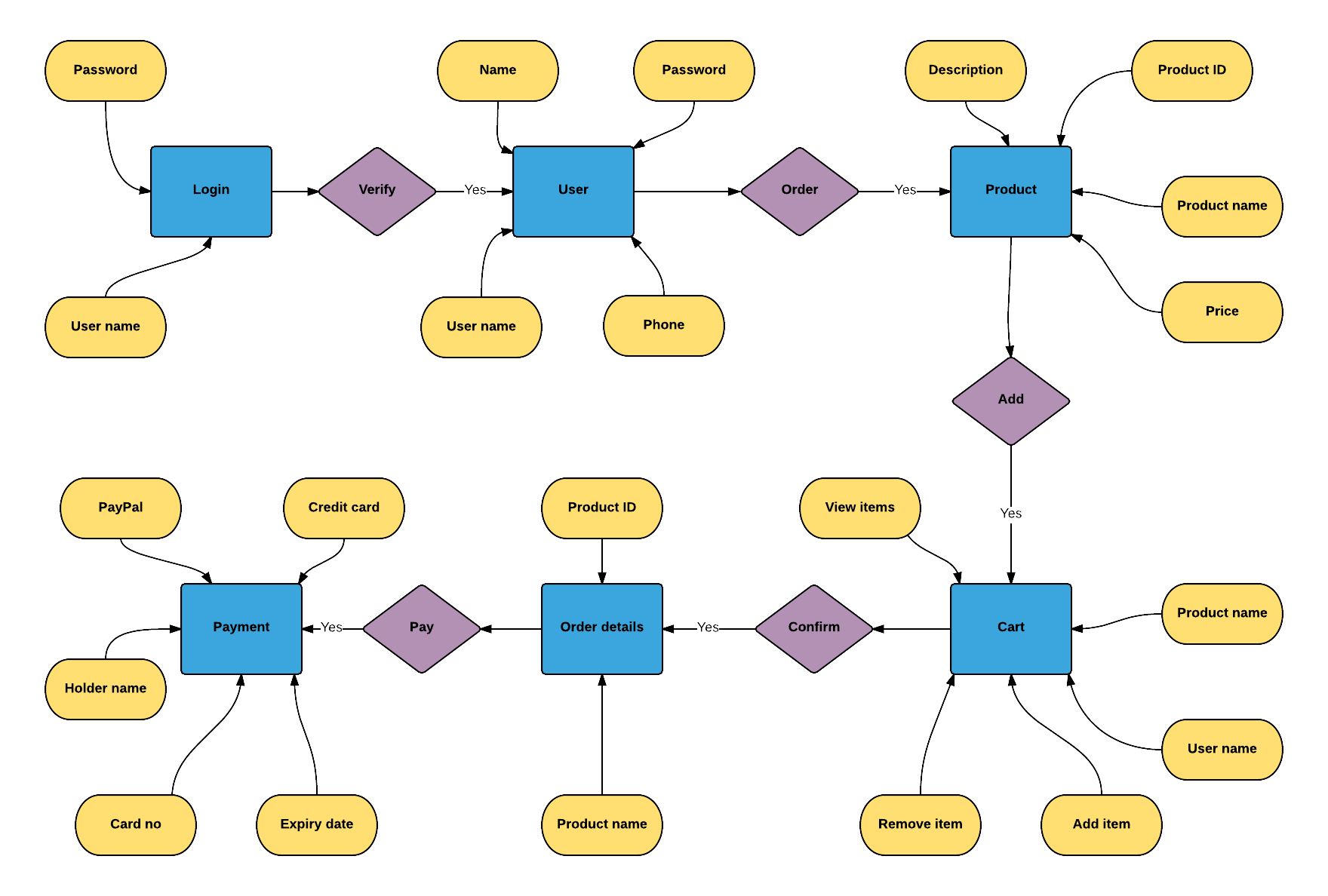
The observer design pattern which is a behavioural design pattern, that maintains a list of dependents called observers and notifies them of any state change.

# Class Diagram

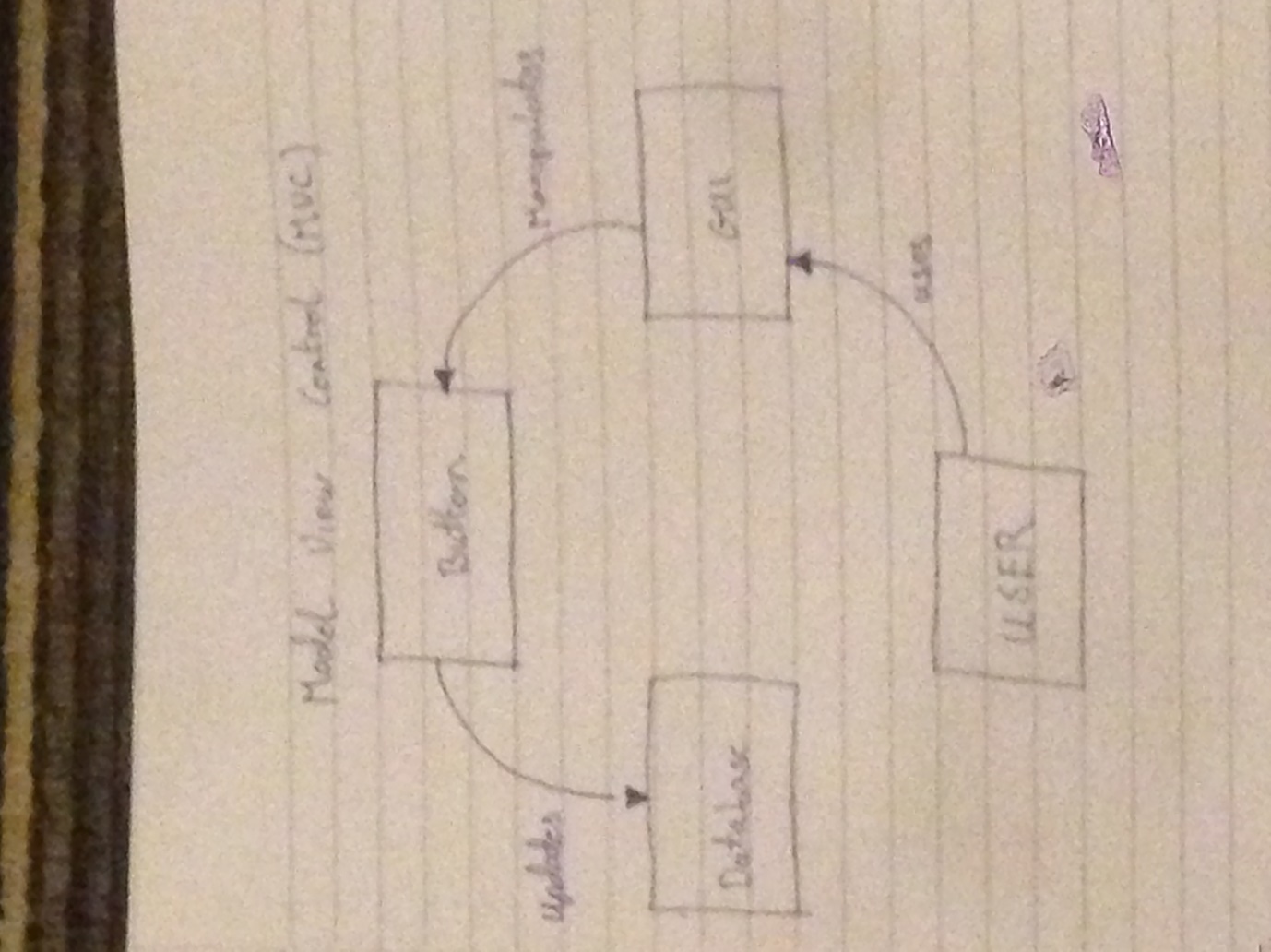
**

Interaction Diagram

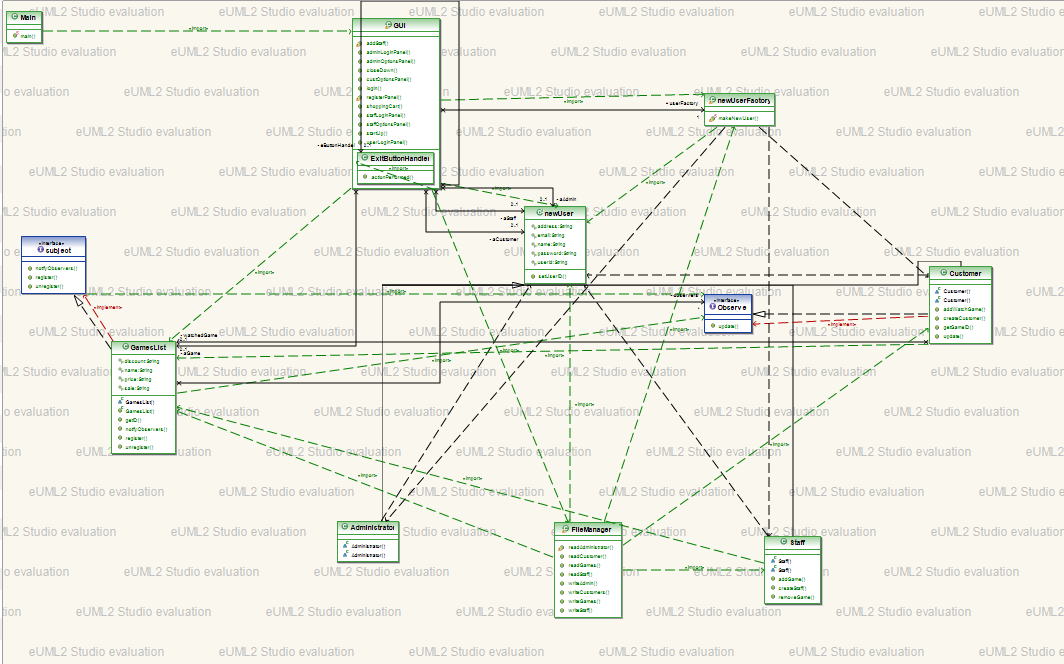
# Entity Relationship



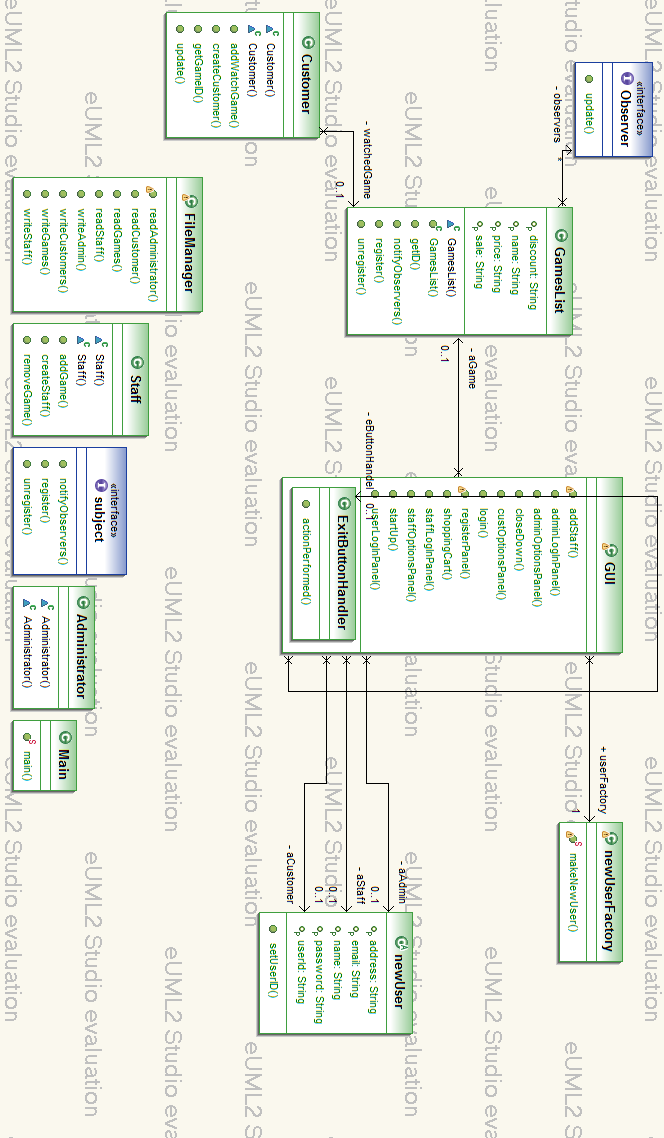
# MVC



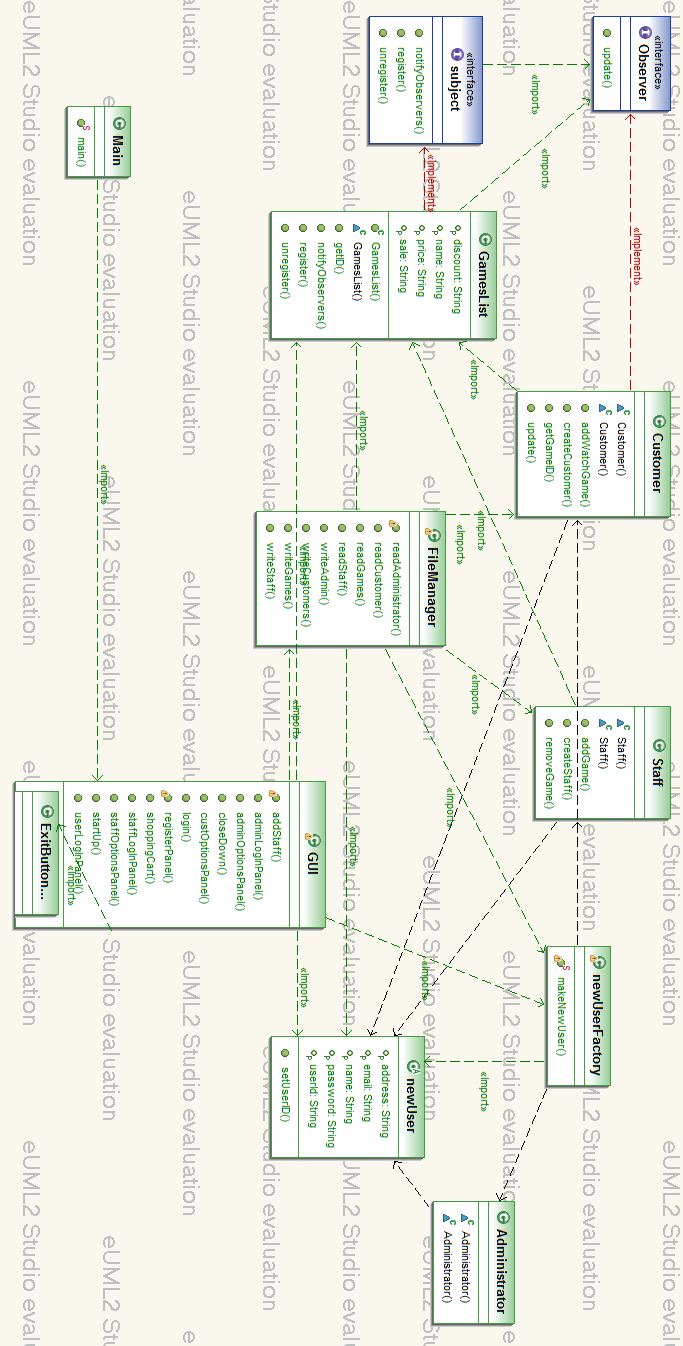
# Architectural Diagram



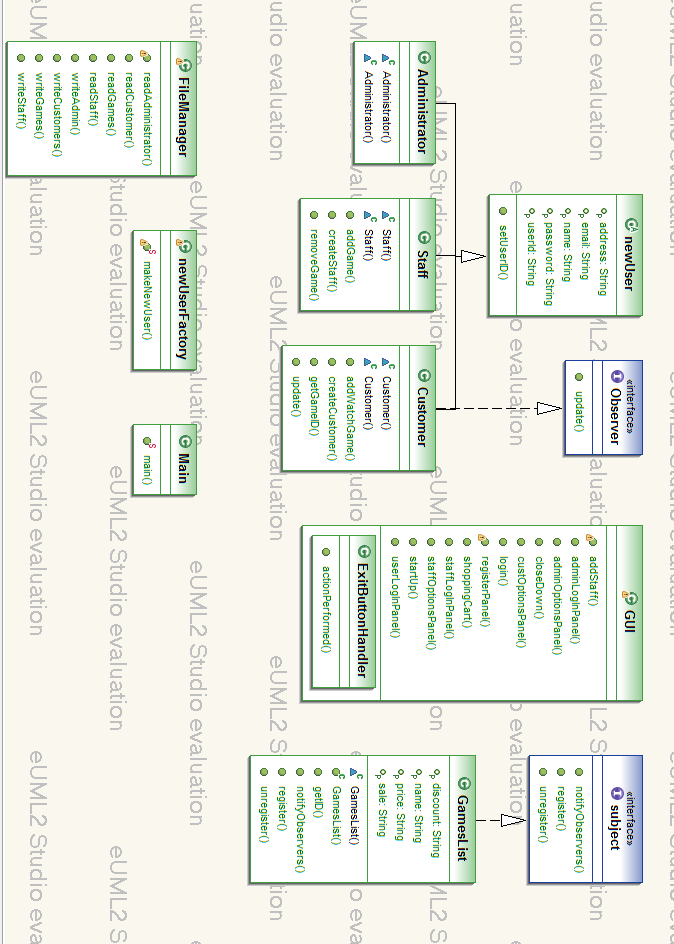
# Class Diagram (Associations)



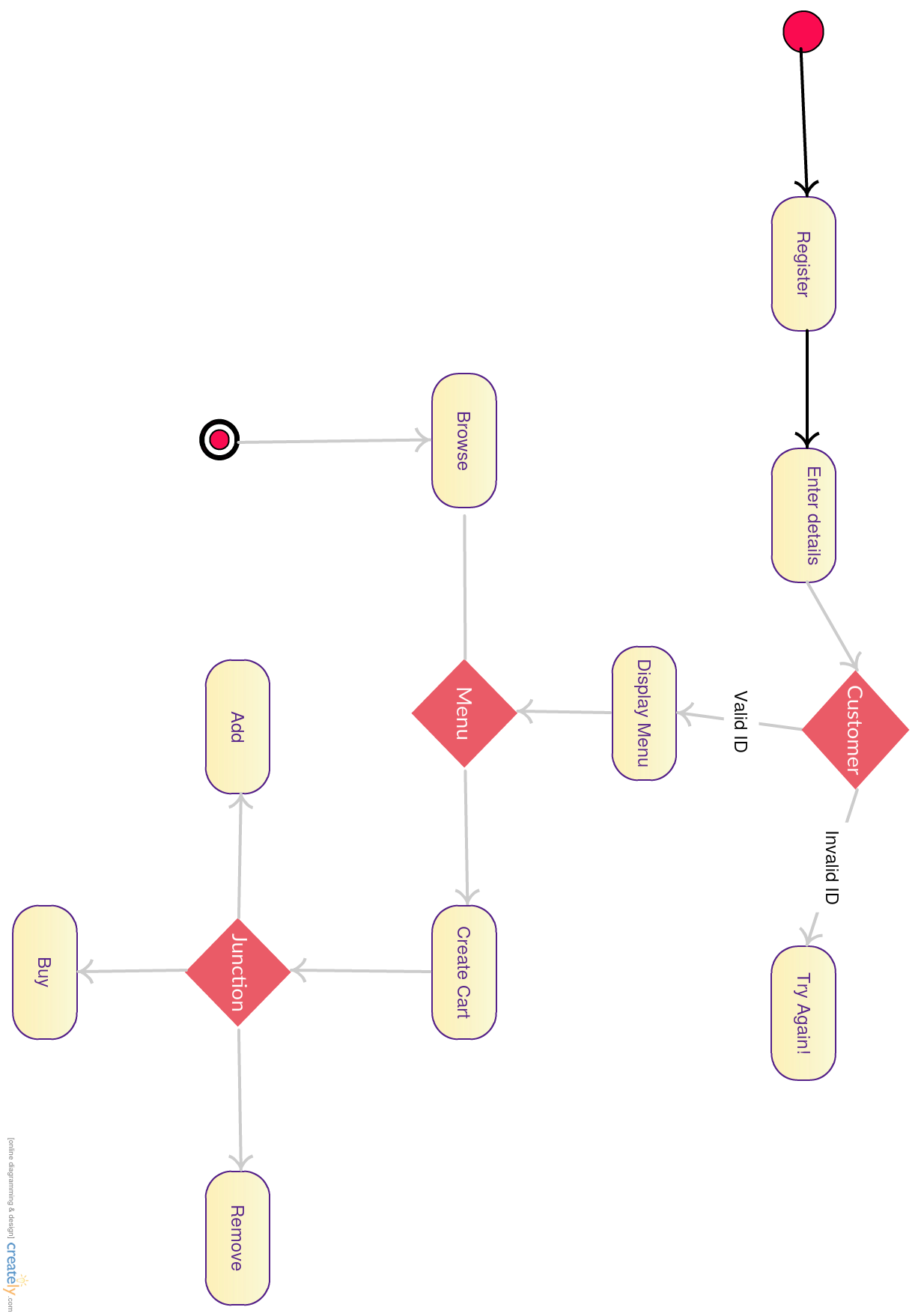
# Class Diagram (Dependencies)



# **Class Diagram (Inheritance)**



# State Chart



# Descriptions of Patterns

# Factory

The main benefit of factory design pattern is that it localizes the creation of objects within the system. This makes it easier to maintain and modify the system as the systemm grows. In our project we use factory classes for instances of customer and staff. These factors create concrete objects based on what type passes into the make function inside the factory.

# Observer

The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. Where we use this in our project is for the discount. When the admin changes the discount for a game the program looks at the games ID and then checks all the customers watch list to see if any of them have that unique ID. If so they are sent an email saying that the game they are watching has a discount.

# Critique

For our design pattern we had considered using a third pattern singleton. Singleton would have been used on the Admin class as there is only one instance of an Admin during the projects life cycle. The GOF’s definition is as follows: The singleton pattern ensures that only one object of a particular class is ever created. All further references to objects of the singleton class refer to the same underlying instance.

Since the observer pattern is used to allow an object to publish changes to its state. Other objects subscribe to be immediately notified of any changes. If time had been permitting we would have used the observer so that it could watch multiple games, by creating an ArrayList<watchList>.

As with our factory method we could have extended that even further by adding more users such as a game reviewer. This new user would have all the same constructors as staff and customer; his extra features would be rating a game and writing reviews that can be voted on.

Another issue is that instead of having to close down the program and restarting it we would have liked to have a return to menu button.

# References

Cite it Rite guide to Harvard referencing style [online], available: <https://www2.ul.ie/pdf/467372218.pdf>

Tutorials Point – UML Interaction Diagram [online], available: <http://www.tutorialspoint.com/uml/uml_interaction_diagram.htm>

Black Wasp- Gang of Four Patterns [online], available: <http://www.blackwasp.co.uk/gofpatterns.aspx>

Warren Carr [image online] available: <http://pre15.deviantart.net/b8bd/th/pre/i/2013/065/4/1/monolith_by_warrencarr-d5x6apb.jpg>