Object Oriented Software Engineering Project StockChecKR

Group 3:

- Leticia Lopez Castillo-301087698
- Sukhjinder Kaur -301087895
- Joshua Timbol-301068352
- Nguyen Hoang Long -301085990
- Hussam Eldin Mohamed - 301090956
- Ahmad (0% contribution)

Contents part A

Contents part B

Contents part C

Section 1: Problem statement

1.1 a) Problem & Need

Covid has changed people's needs, security action plans like social distance and staying safe are the main reasons why this application was designed for online grocery shopping this would help shorten the time people spend in the store and even support people bringing the grocery to their homes with a delivery service, people certainly know what they come looking for, but usually come to stores without even knowing if they'll find all they need. Not finding an important product usually ends up in visiting more than one grocery store in a day/week which might be a problem not contributing to social distance.

Even before covid, this solution would have been a lot of help for customers because it involves just opening an app and searching for the product name or brand and making sure how many pieces would they find in stock on several stores, and making the way to pick a grocery store to go to easier.

b) List of:

(i) capabilities

- Provide the user with a list of products
- Filter products by name, brand, location and more..
- Search the number of items found per store
- Make a cart mixing products of different stores
- Buy products for in-store pickup
- Get notifications on stock arrival
- Give feedback to the system

(ii) Benefits

- The user would be able to make grocery online
- Grocery shopping would allow several stores stock so that user can have exactly the product desired

- User can consult on every desired product before wasting time and money on going to the shop
- If the user desires just to visit one store, he or she would be able to find a similar product for inexistent products, such as different brand or ingredients

1.1 b) Identify the stakeholders and their roles

Stakeholders could be anyone who does grocery, workers, packers, big product companies, store managers, or any other user/ possible user.

Direct stakeholders: user (any person with the app): its role is to navigate in the app to find products, make a cart, buy, or just search for something in stock

Stores (a store manager): upgrade on every item in stock as it is registered for sale including quantities, unregister every item after it's been sold

Developing staff: keep the app working and testing on solutions for possible bugs or inconsistencies

Programmers: Update application on a weekly basis. Could include new features, bug fixes and overall customer feedback getting approved and cleared.

Customer Service Representatives: Help any confused customers with any questions or inquiries they have. Fulfil any refunds a customer requested.

1.1 c) Identify the sub-systems of your application (What are its functional components)

Payment System - Accurately and Securely check out customers items and send them confirmation email as soon as payment has been received on our end.

Search System - Would be useful to find a more specific product or brand or store.

GPS System - Helps users to find the nearest stores and pick according to the user's needs.

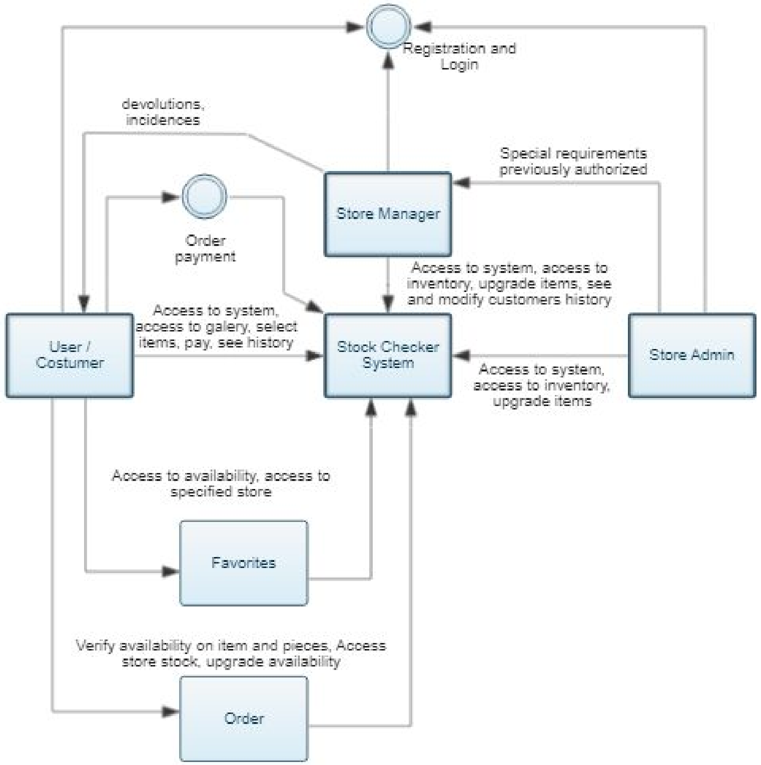
Maps External Linked System - This would help the user to move between maps apps and our System in case they need the specific route and instructions.

Wish List - Implemented wish list that users can add items in and share with friends.

1.4 d) Who are the intended users of the SRS documentation.

This project is a prototype for a universal stock checker. It is built to be useful for every individual especially during these tough times with COVID-19. The true intended audience we have is for people who want to quickly organize themselves before heading out by making sure every store they are going to, has the product they need without wasting any sort of time. StockCheckR also has the feature of an all-in-one shopping cart which can combine items from a variety of stores to one checkout price. This can help with the safety measures that the world has in place with its physical distancing. The intended users for the SRS documentation range from programmers, testers, developers, project managers, researchers, and designers.

Section 2: A Context Flow – Structured Modeling



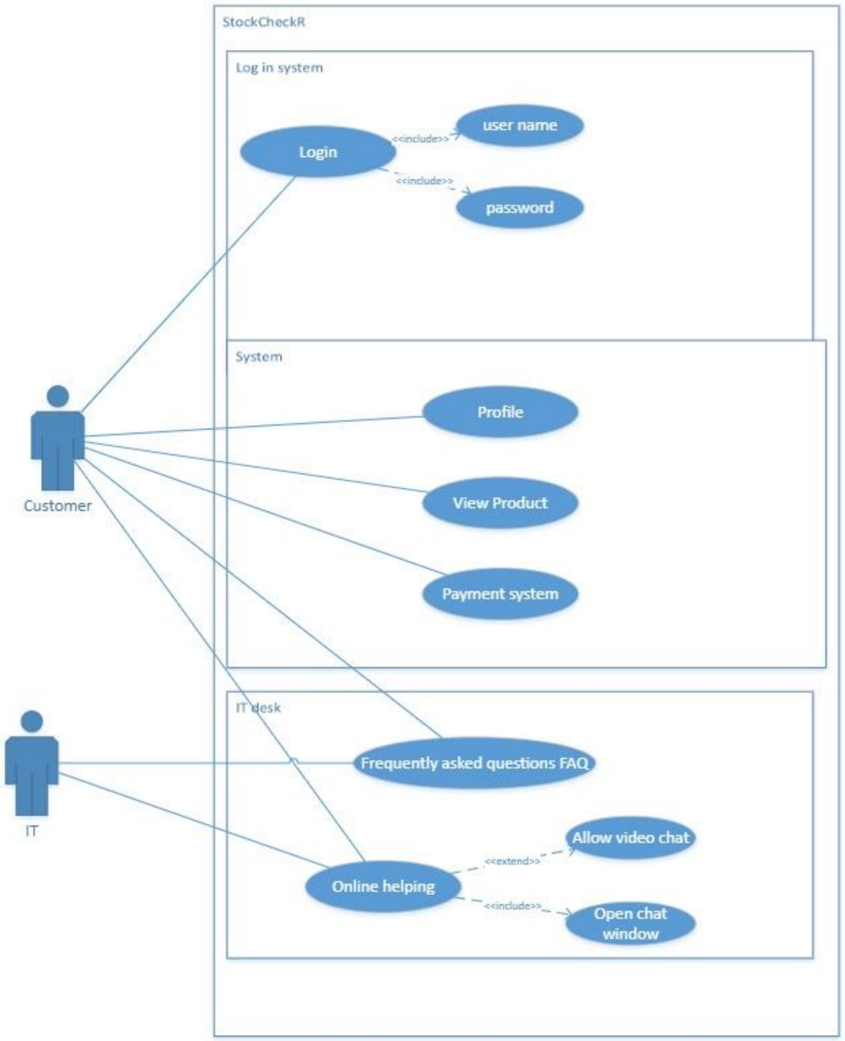
Registration and Login
devolutions, incidences
Special requirements previously authorized
Store Manager
Order payment
Access to system, access to inventory, upgrade items, see and modify customers history
Access to system, access to galery, select items, pay, see history
User / Costumer
Stock Checker System
Store Admin
Access to system, access to inventory, upgrade items
Access to availability, access to specified store
Favorites
Verify availability on item and pieces, Access store stock, upgrade availability
Order

Section 3: Requirements – Functional -- UML Use Case Modeling

3.1 Goal Use Cases

FR #	Goal Use Case	Role Player	Description
FR 01	Notify	Customer	The system gives the customer notification on the website or the mobile application when the store has updated stock information.
FR 02	View products	Customer	The system allows the customer to review the amount of the products and information about stocks.
FR 03	Payment system	Customer	The payment system interface supports credit CARDS and most popular online payment software such as Apple Pay and PayPal.
FR 04	Profile	Customer	The system will allow the customer to renew their profiles.
FR 05	Frequently asked question	Administrator	The system should have the ability to enable the manager to set up and operate frequently asked questions.
FR 06	Email	Store holder/ Customer	The system will automatically send emails to confirm that customers update the profile or purchase from StockCheckR.

3.2 Use case Diagrams



3.3 User Stories:

1) As a user, I want to search for the item so that I can decide to buy it.

Acceptance criteria:

- Should be able to find the product
- Should be able to check out securely and efficiently
- Accurately show stock of item from multiple vendors
- Having vendor contact numbers to stores nearby customers to possibly ask any questions they may have or to put items on hold (varies).

2) As a user, I want to access my information so I can update my profile.

Acceptance criteria:

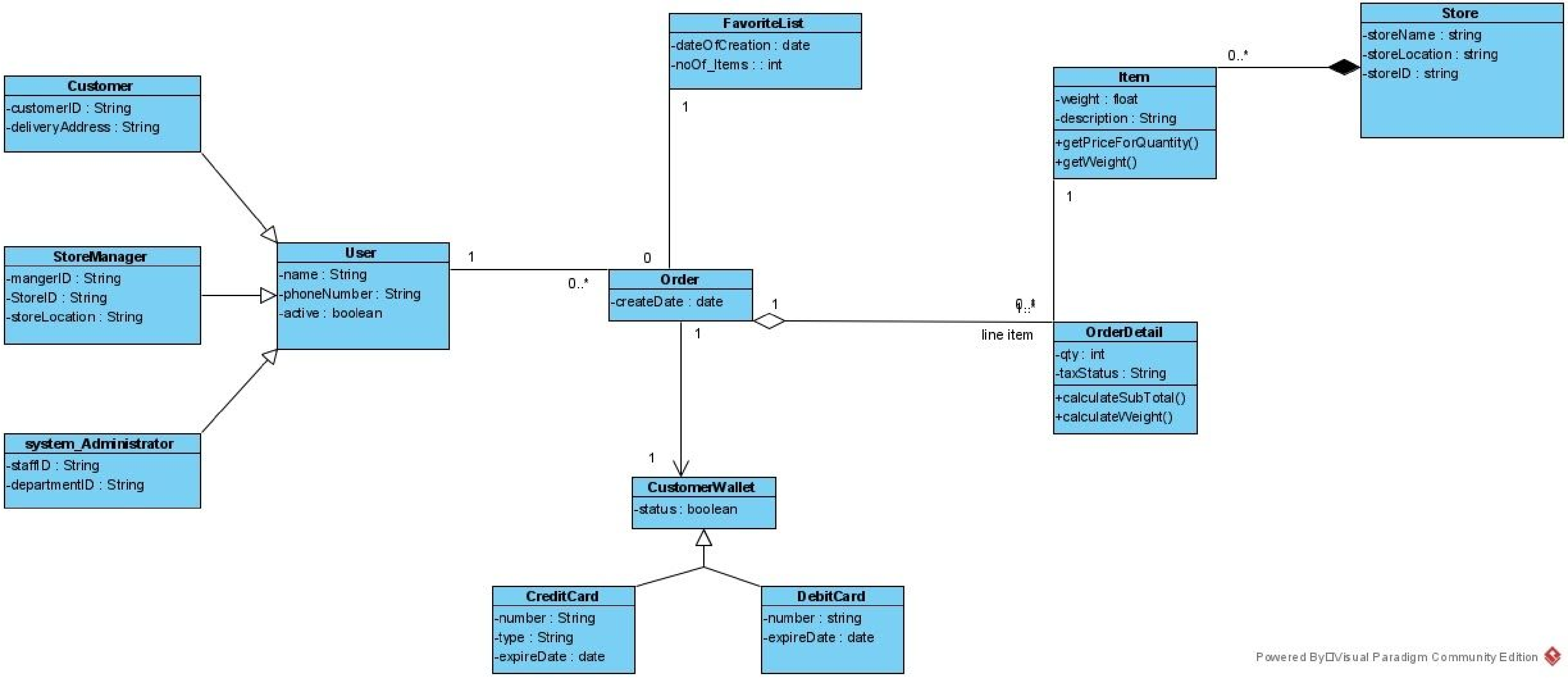
- Should be able to update personal information such as an address, name, picture, and phone number.
- Secure and Accurate security
- Logins always having some sort of 2-step verification to ensure maximum security
- Easy access to information whether it would be via desktop, tablet, or mobile phone

Section 4.0 UML Domain Class Diagram

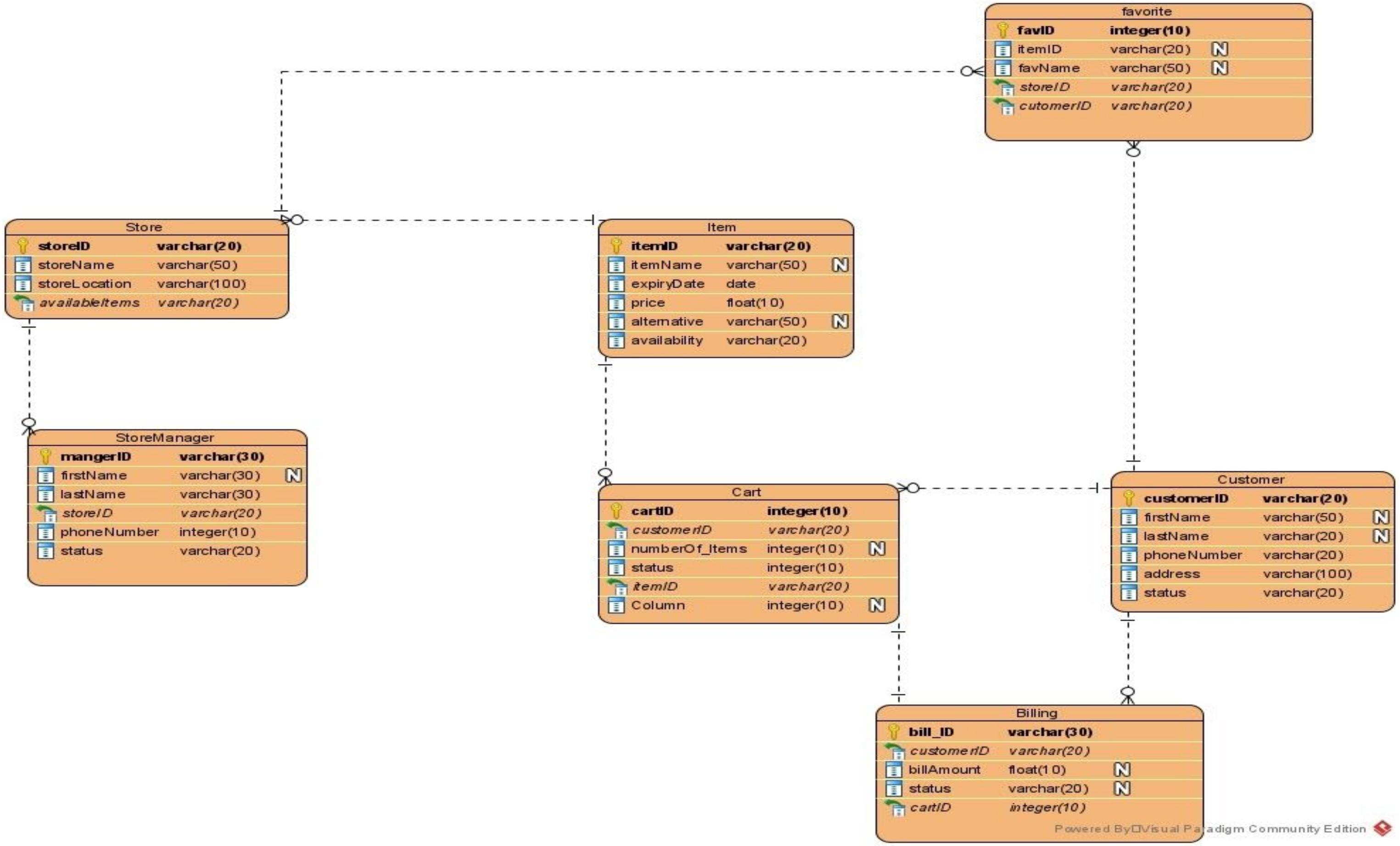
4.1 List of the classes

- Favourite
- Customer
- Cart
- Product
- Store
- Payment
- Availability
- Credit
- Cash
- Wire Transfer

4.2 Domain class diagram



Section 5.0 Sketch an Entity Relationship Diagram to show the tables for your database



favorite		
favID	integer(10)	
itemID	varchar(20)	N
favName	varchar(50)	N
storeID	*varchar(20)*	
cutomerID	*varchar(20)*	

Store		
storeID	varchar(20)	
storeName	varchar(50)	
storeLocation	varchar(100)	
availableItems	*varchar(20)*	

Item		
itemID	varchar(20)	
itemName	varchar(50)	N
expiryDate	date	
price	float(10)	
alternative	varchar(50)	N
availability	varchar(20)	

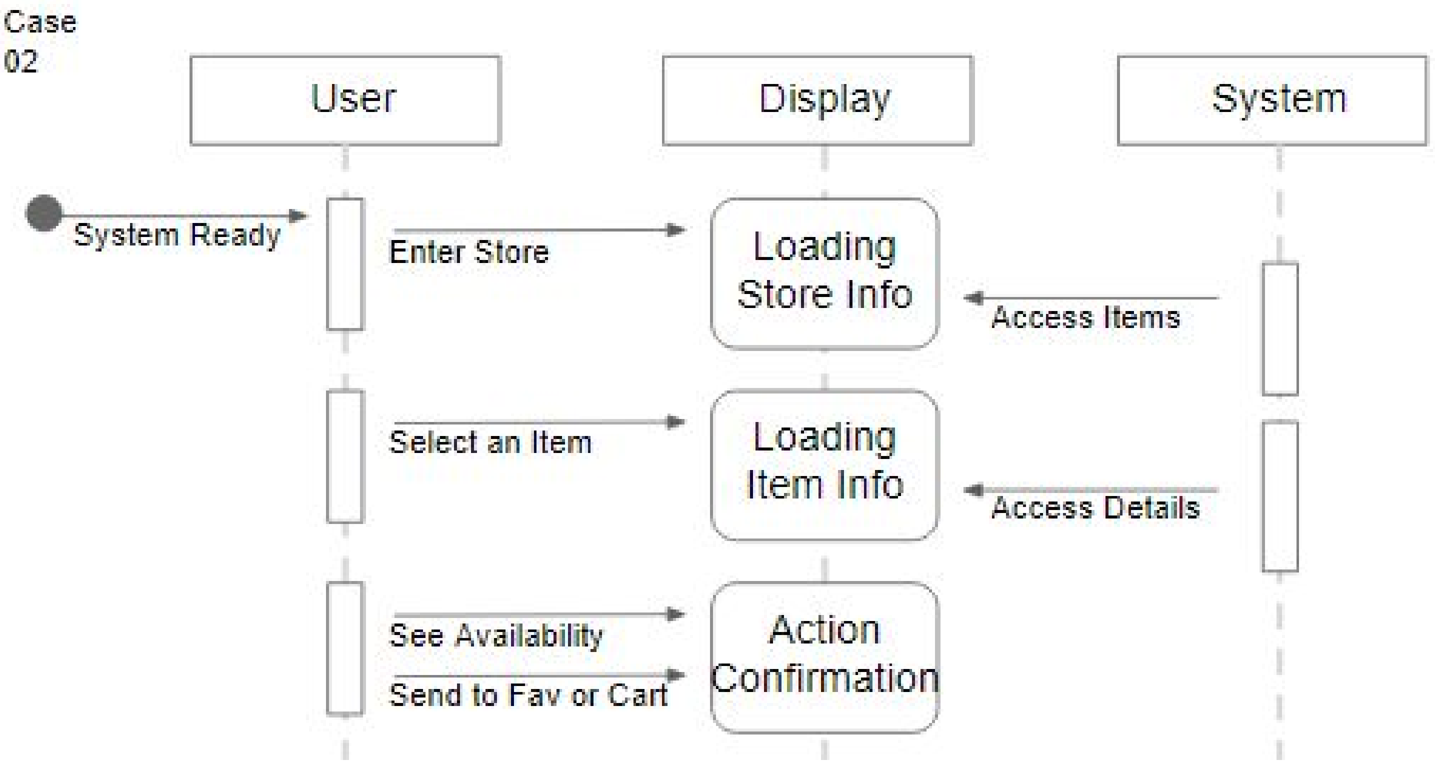
StoreManager		
mangerID	varchar(30)	
firstName	varchar(30)	N
lastName	varchar(30)	
storeID	*varchar(20)*	
phoneNumber	integer(10)	
status	varchar(20)	

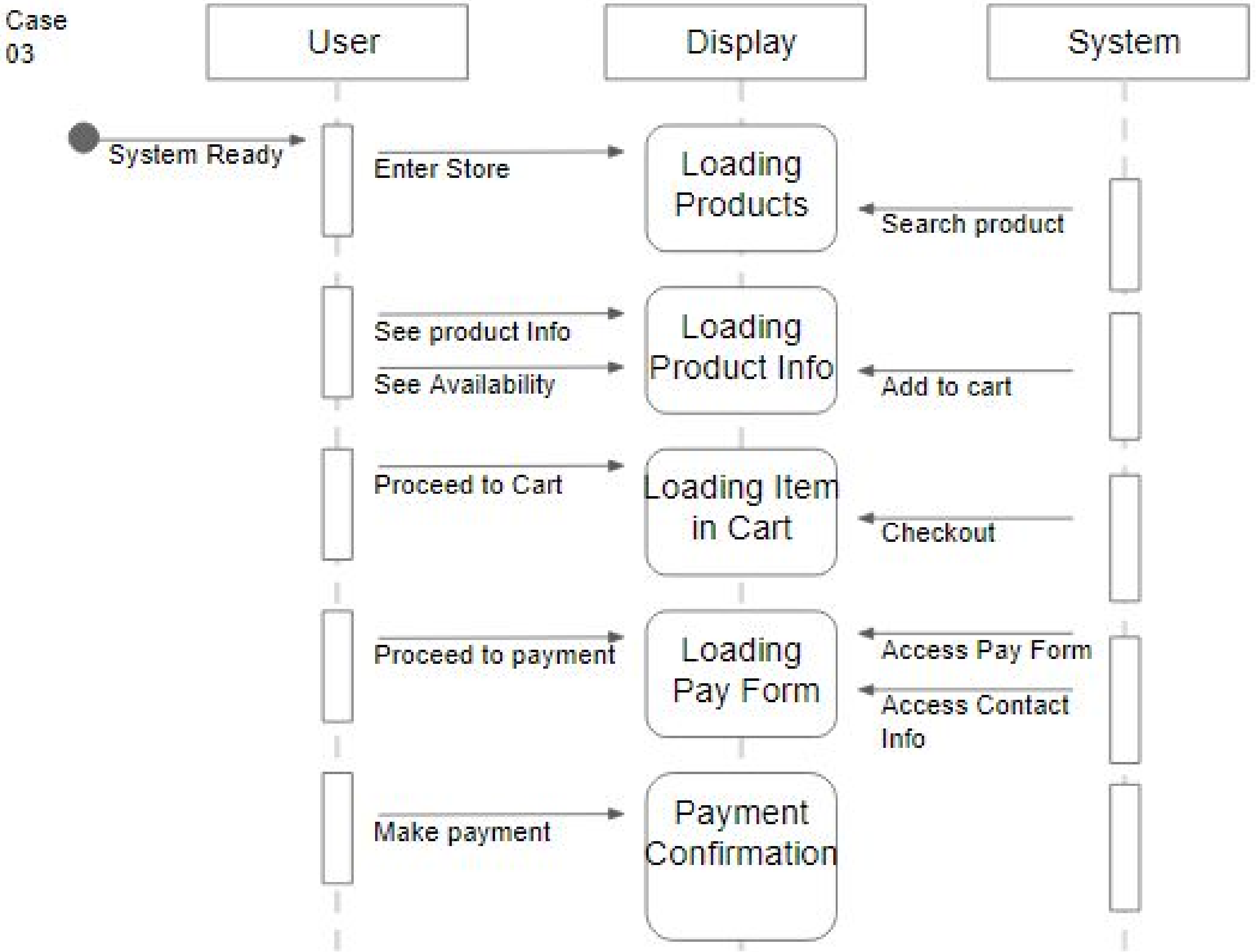
Cart		
cartID	integer(10)	
customerID	*varchar(20)*	
numberOf_Items	integer(10)	N
status	integer(10)	
itemID	*varchar(20)*	
Column	integer(10)	N

Customer		
customerID	varchar(20)	
firstName	varchar(50)	N
lastName	varchar(20)	N
phoneNumber	varchar(20)	
address	varchar(100)	
status	varchar(20)	

Billing		
bill_ID	varchar(30)	
customerID	*varchar(20)*	
billAmount	float(10)	N
status	varchar(20)	N
cartID	*integer(10)*	

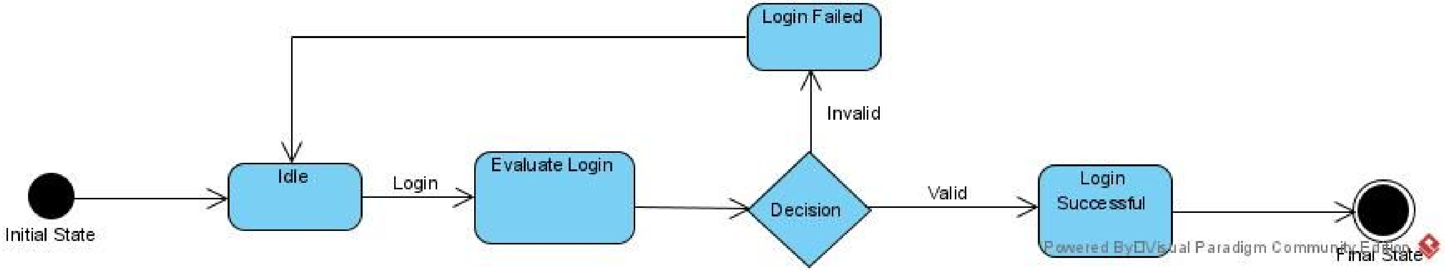
Section 6.0: Sketch Two UML Systems Sequence diagrams



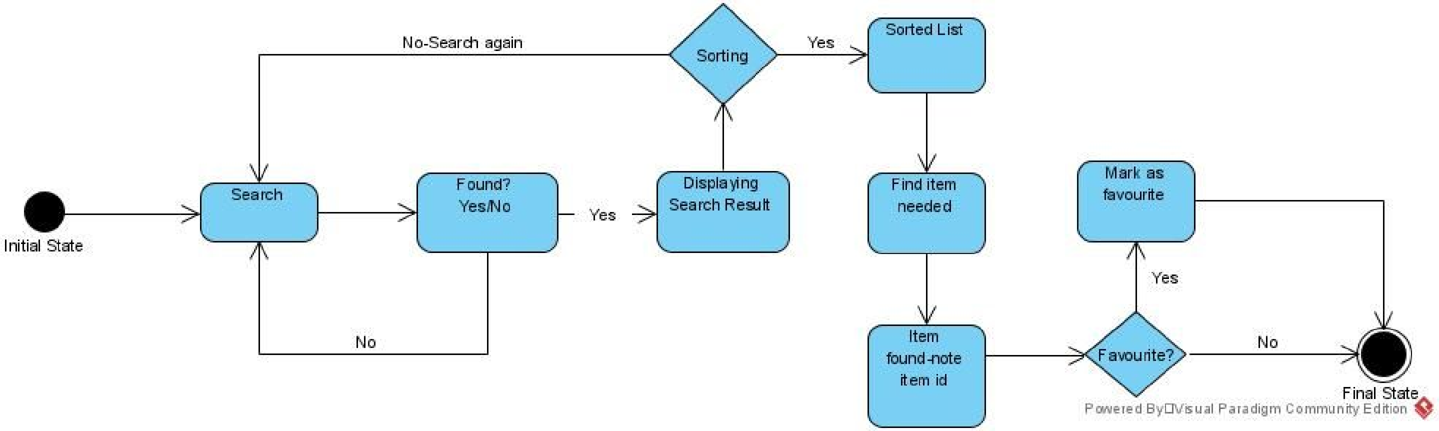

Case
03
User
Display
System
System Ready
Enter Store
Loading Products
Search product
See product Info
See Availability
Loading Product Info
Add to cart
Proceed to Cart
Loading Item in Cart
Checkout
Proceed to payment
Loading Pay Form
Access Pay Form
Access Contact Info
Make payment
Payment Confirmation

Section 7.0 Sketch Two UML State Diagrams

7.1 Login to the StockChecKR system



7.2 Search an item in StockChecKR



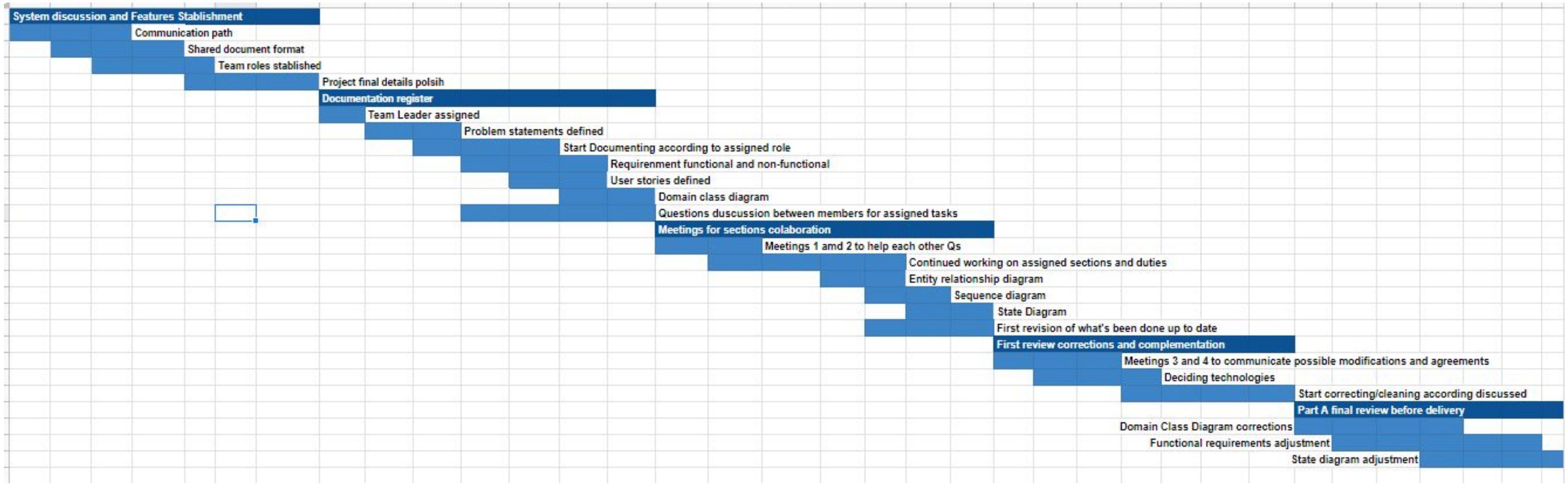
Section 8.0: Technologies

8.0 Technologies used for Stock ChecKR software development are listed below

Categories	Technologies Used	Comments
Platform	● Windows ● Android	The interface used for web view The interface usedfor the Mobile view
Programming Languages	**Client-Side Development** ● Java ● C# **Server-Side Development** ● HTML 5 ● JavaScript ● CSS3 ● NodeJS	Programming languages used for the application development (both back-end and front-end development)
Business Logic /Documentation	● Visual Paradigm ● Visio ● Lucidchart	Tools used for the diagrams
Data side/Database	● SQL-Developer ● Oracle	The database used for processing data.
Version Control	● GitHub	Online technology used for code backup and version control.

Section 9.0: Project Management

9.1 Gantt Chart for the activities from Sections 1 to 8 of this Part A



PART B - Software Design Architecture

Section 1: Requirements Edits to Part A

1.1 Problem & Statement (modified in part A)

1.2 The CFD has to show the complete list of stakeholders, interfaces, entities. (modified in part A)

Section 2: Overview Model

2.1 Intended users for the Software Design Document- SDD

SDD is the written description of the software product that gives a total portrayal of the framework plan so that the software development team can have a full comprehension of what should be manufactured. This gives the team overall guidance to the architecture of the software project. This document not only describes the software already in place but also enforces the compatibility for future modifications or add-ons. Thus, the target group for this report is the software developers, software engineers, researchers and the project managers. They are liable for framework improvement.

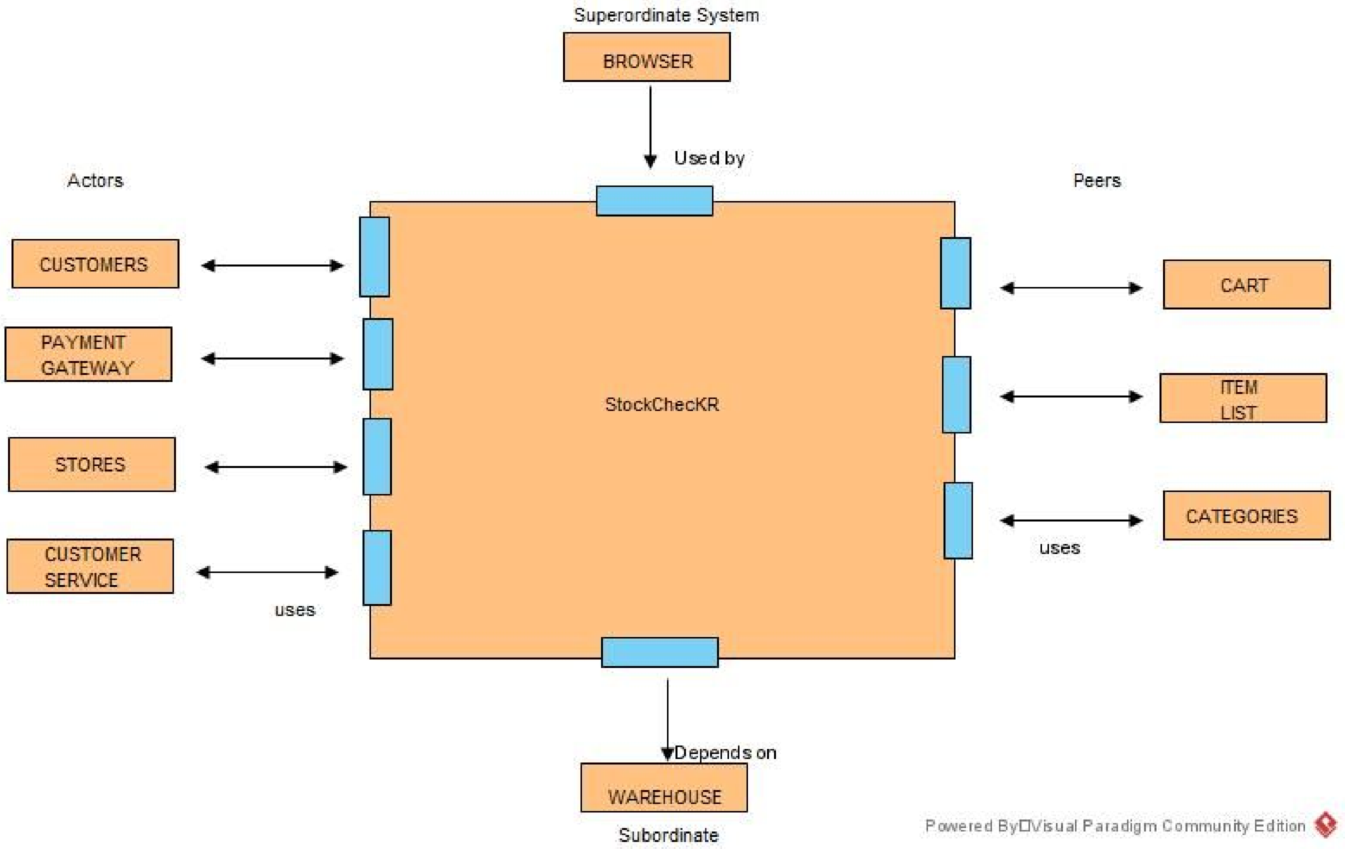
2.2 Architectural Context Diagram

Architectural Context Diagram is a graphical representation of the StockChecKR which shows the interaction of the external entities with it.

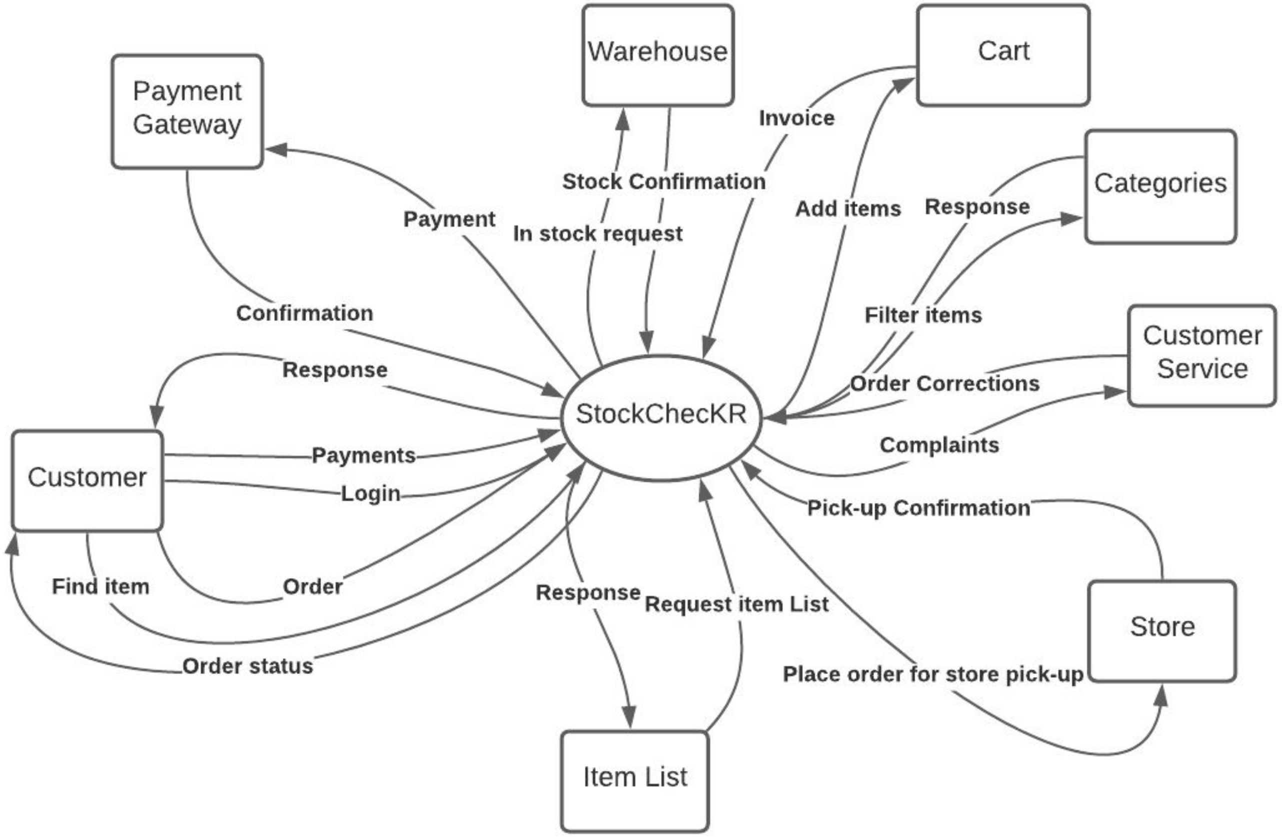
Four elements of ACD are below:

1. Superiors System - the system used to realize the function
2. Subordinated System - System which is used by the main system
3. Peers - Internal systems in the main system use the systems
4. Actors - External entities with consumes the data of the main system

2.2.1 HOW overview- Architectural Context diagram



2.2.2 What overview- Architectural Context diagram



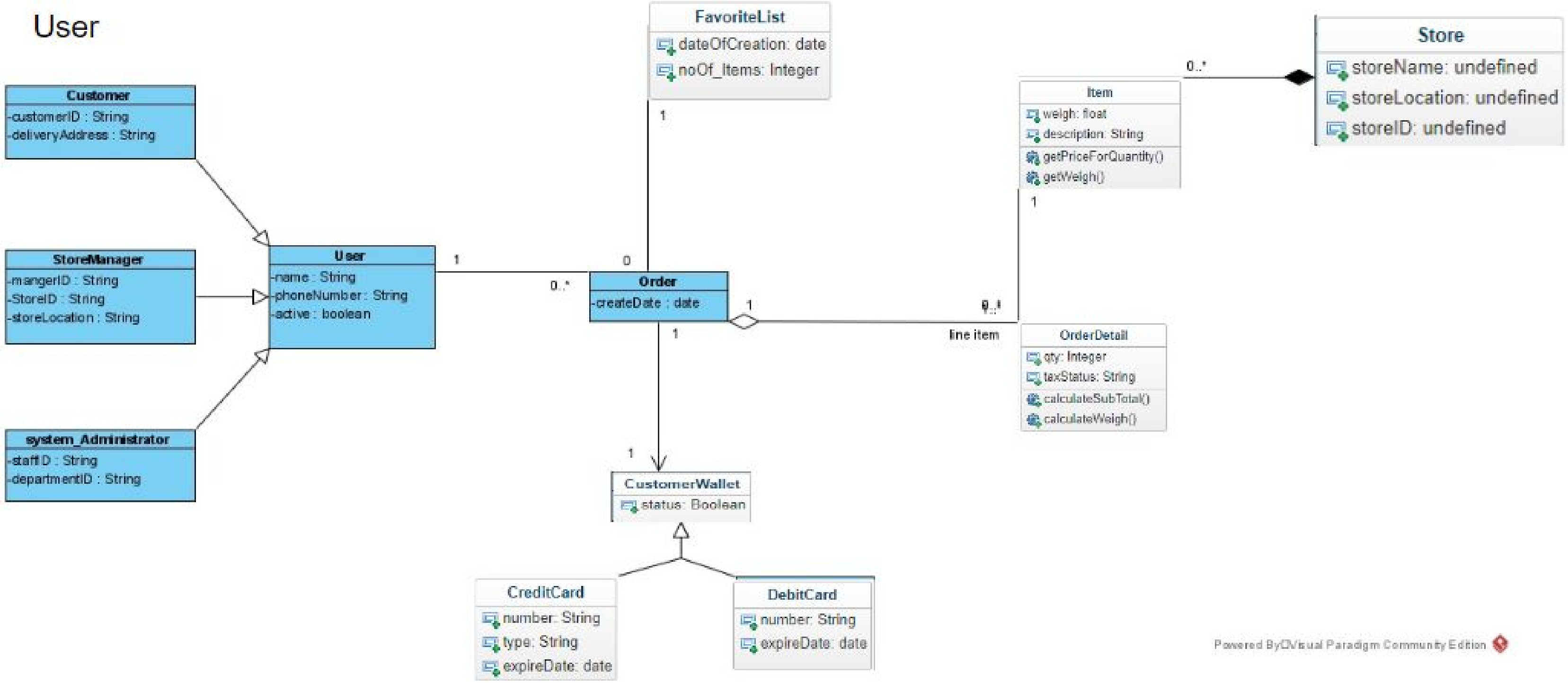
Section 3: Modularization

Develop the Design classes as per Sub-system Component

3.1 Partitioning of the analysis model

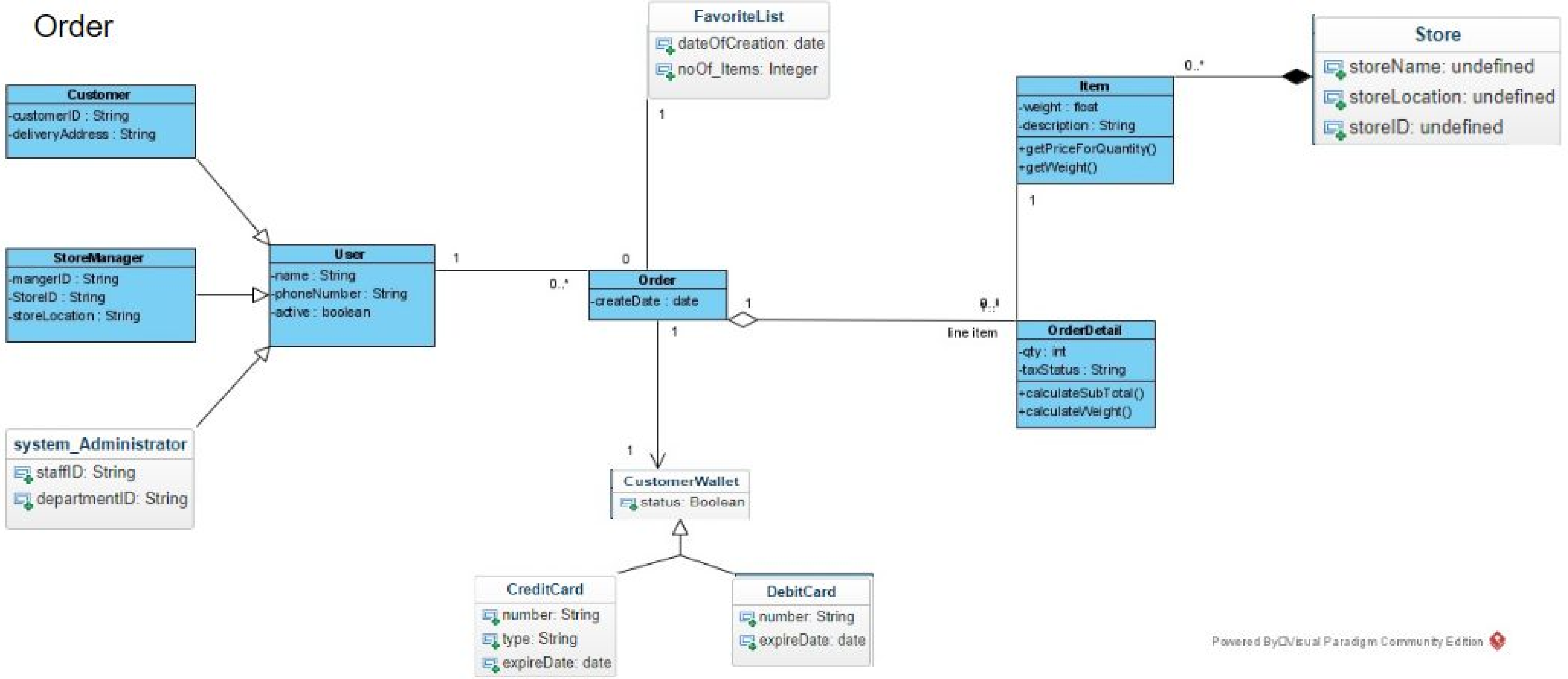
3.1.1 User Subsystem

- Order
- User
- Customer
- StoreManager
- system_Administrator



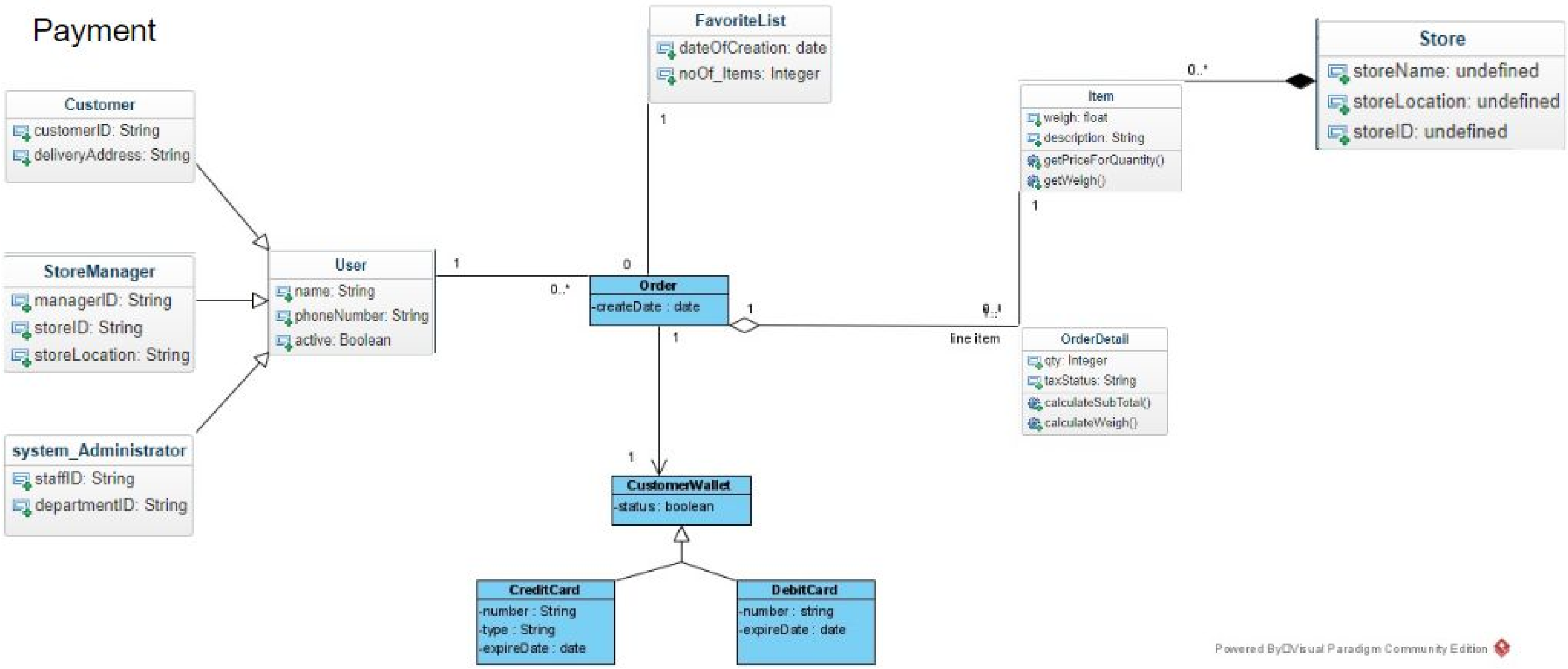
3.1.2 Order Subsystem

- Order
- User
- Customer
- StoreManager
- Item
- OrderDetail



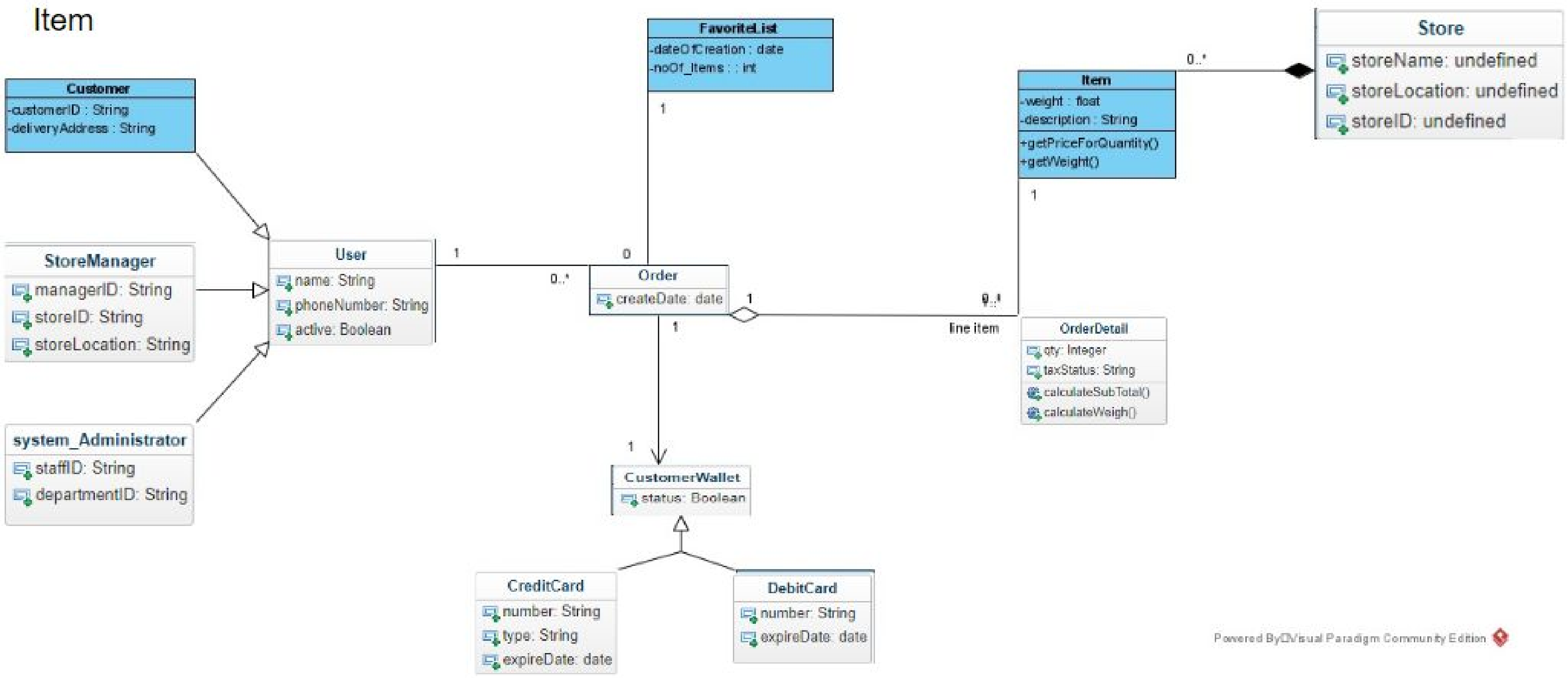
3.1.3 Payment Subsystem

- Order
- CustomerWallet
- CreditCard
- DebitCard

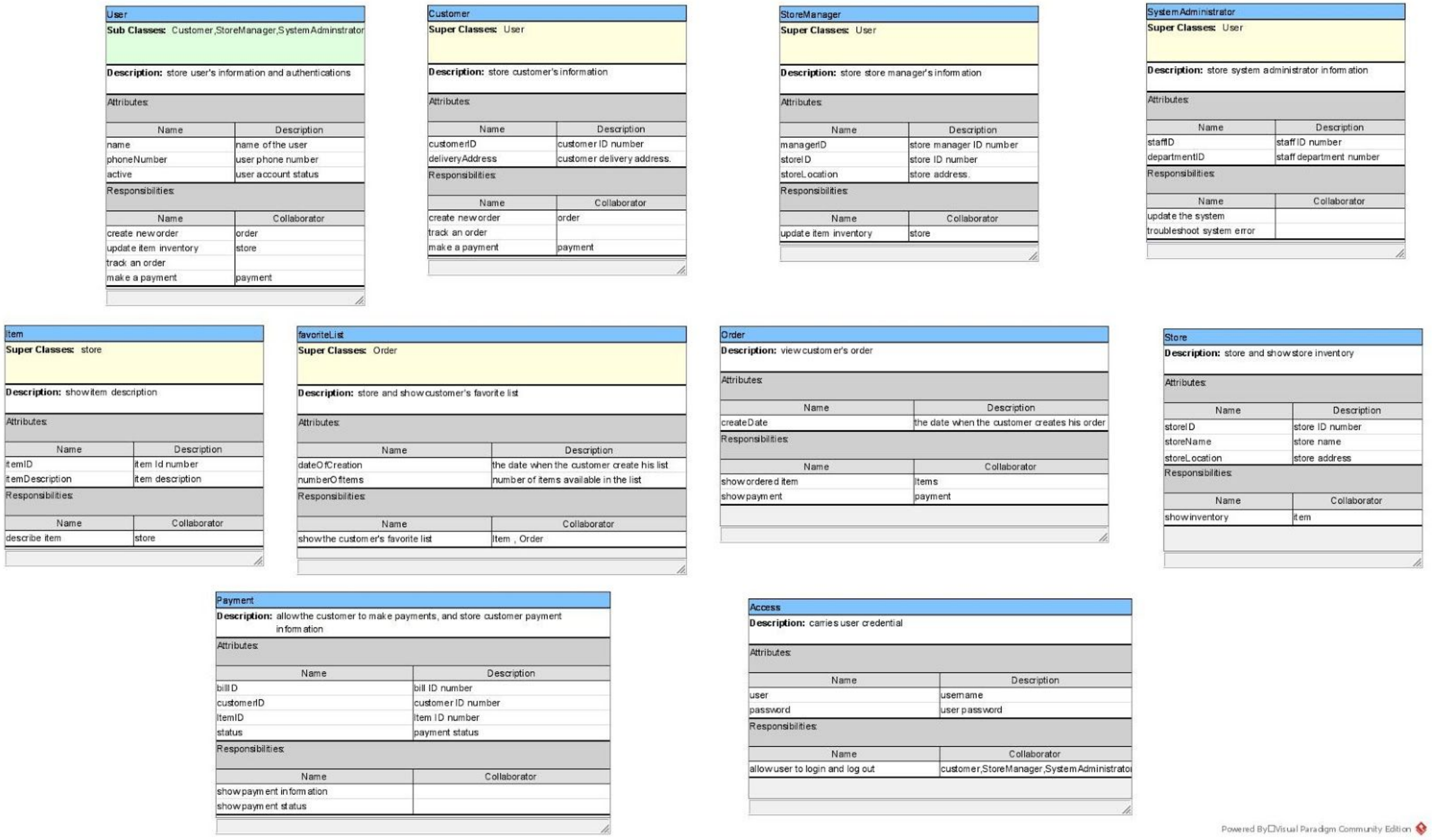


3.1.4 Item Subsystem

- Store
- FavoriteList
- User
- StoreManager



3.2 Class Responsibility Collaboration-CRC



User

Sub Classes: Customer,StoreManager,SystemAdminstrator

Description: store user's information and authentications

Attributes:

Name	Description
name	name of the user
phoneNumber	user phone number
active	user account status

Responsibilities:

Name	Collaborator
create neworder	order
update item inventory	store
track an order	
make a payment	payment

Customer

Super Classes: User

Description: store customer's information

Attributes:

Name	Description
customerID	customer ID number
deliveryAddress	customer delivery address.

Responsibilities:

Name	Collaborator
create neworder	order
track an order	
make a payment	payment

StoreManager

Super Classes: User

Description: store store manager's information

Attributes:

Name	Description
managerID	store manager ID number
storeID	store ID number
storeLocation	store address.

Responsibilities:

Name	Collaborator
update item inventory	store

SystemAdministrator

Super Classes: User

Description: store system administrator information

Attributes:

Name	Description
staffID	staff ID number
departmentID	staff department number

Responsibilities:

Name	Collaborator
update the system	
troubleshoot system error	

Item

Super Classes: store

Description: showitem description

Attributes:

Name	Description
itemID	item Id number
itemDescription	item description

Responsibilities:

Name	Collaborator
describe item	store

favoriteList

Super Classes: Order

Description: store and showcustomer's favorite list

Attributes:

Name	Description
dateOfCreation	the date when the customer create his list
numberOfItems	number of items available in the list

Responsibilities:

Name	Collaborator
showthe customer's favorite list	Item , Order

Order

Description: viewcustomer's order

Attributes:

Name	Description
createDate	the date when the customer creates his order

Responsibilities:

Name	Collaborator
showordered item	Items
showpayment	payment

Store

Description: store and showstore inventory

Attributes:

Name	Description
storeID	store ID number
storeName	store name
storeLocation	store address

Responsibilities:

Name	Collaborator
showinventory	item

Payment

Description: allowthe customer to make payments, and store customer payment information

Attributes:

Name	Description
billID	bill ID number
customerID	customer ID number
ItemID	Item ID number
status	payment status

Responsibilities:

Name	Collaborator
showpayment information	
showpayment status	

Access

Description: carries user credential

Attributes:

Name	Description
user	username
password	user password

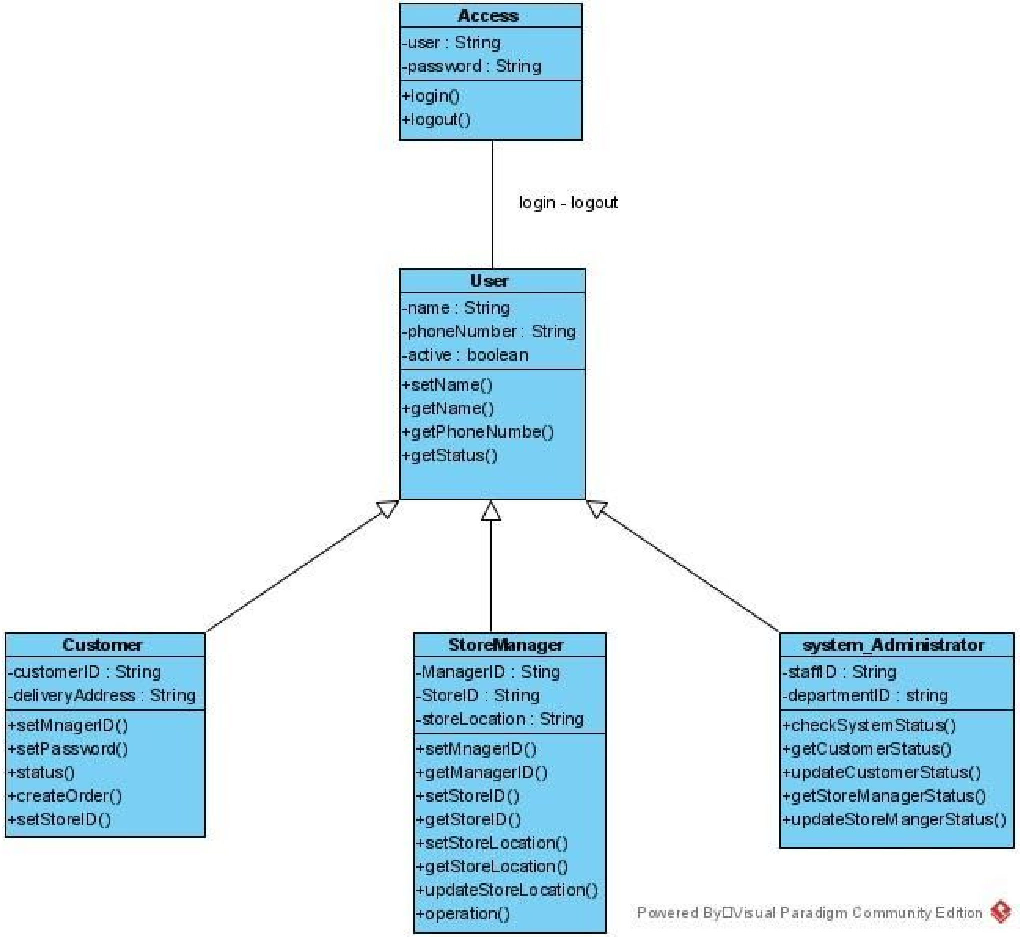
Responsibilities:

Name	Collaborator
allowuser to login and log out	customer,StoreManager,SystemAdministrator

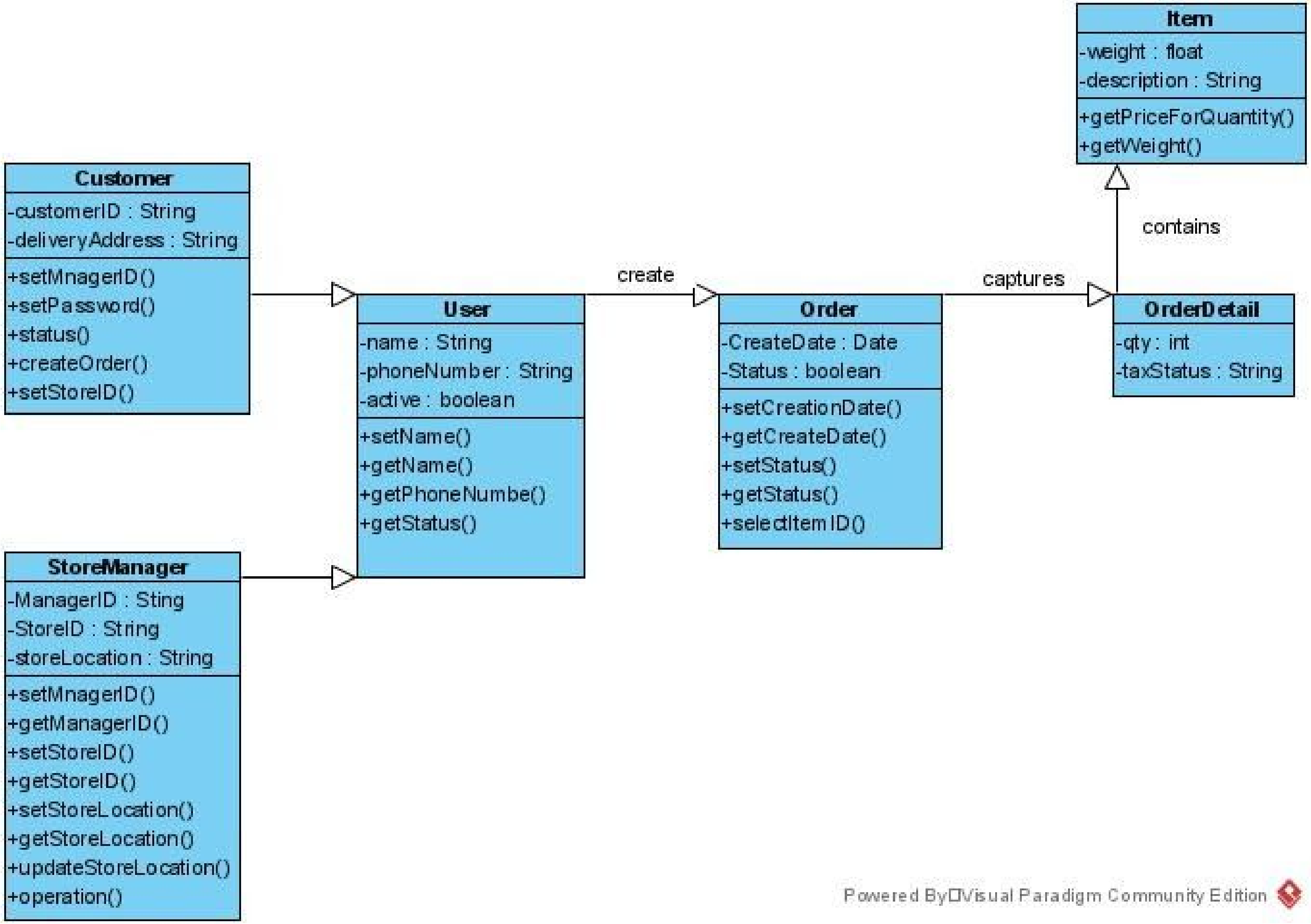
Powered By□Visual Paradigm Community Edition

3.3 Design classes diagram

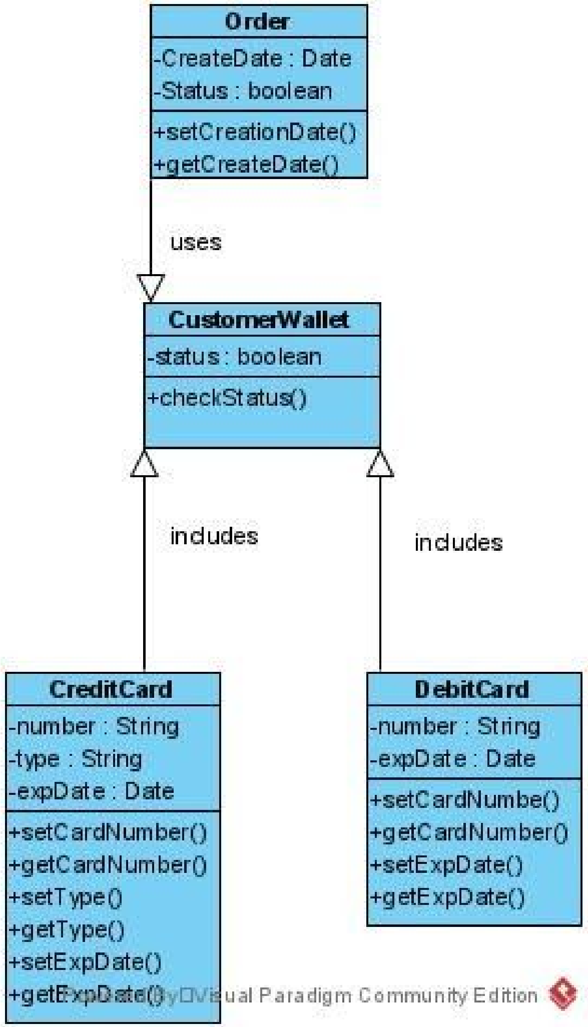
3.3.1 User Login Subsystem



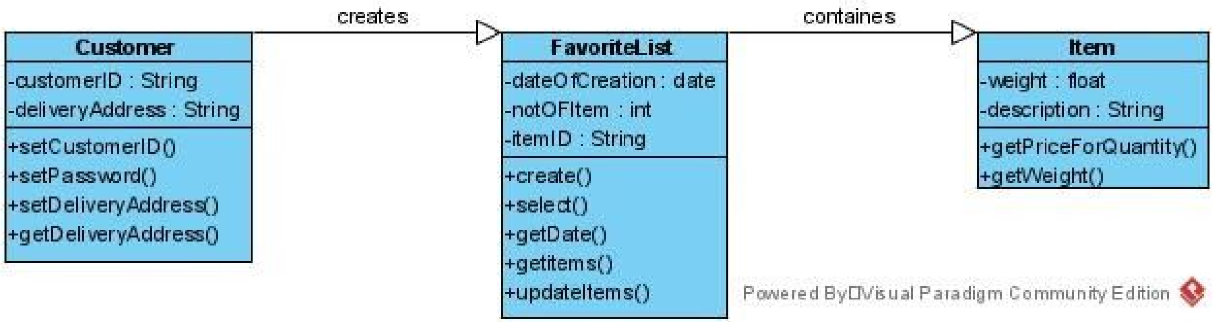
3.3.2 Order Subsystem



3.3.3 Payment Subsystem



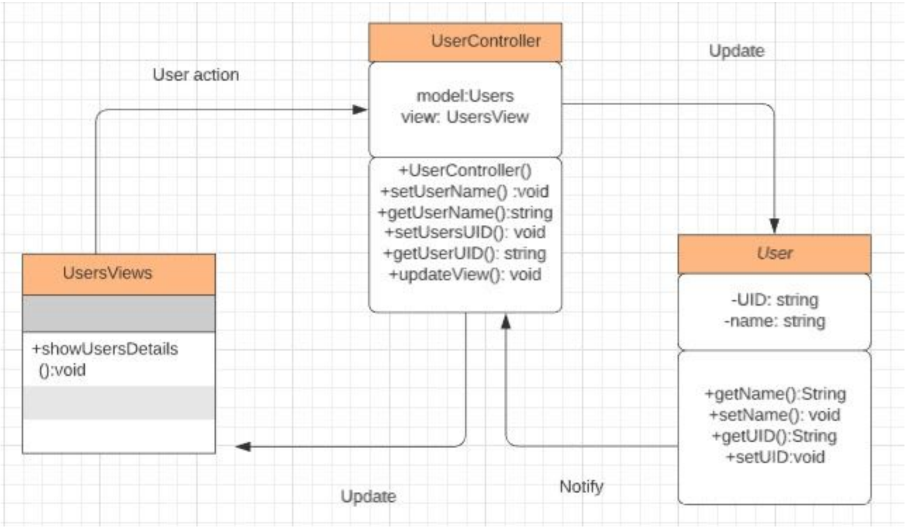
3.3.4 Favorites Subsystem



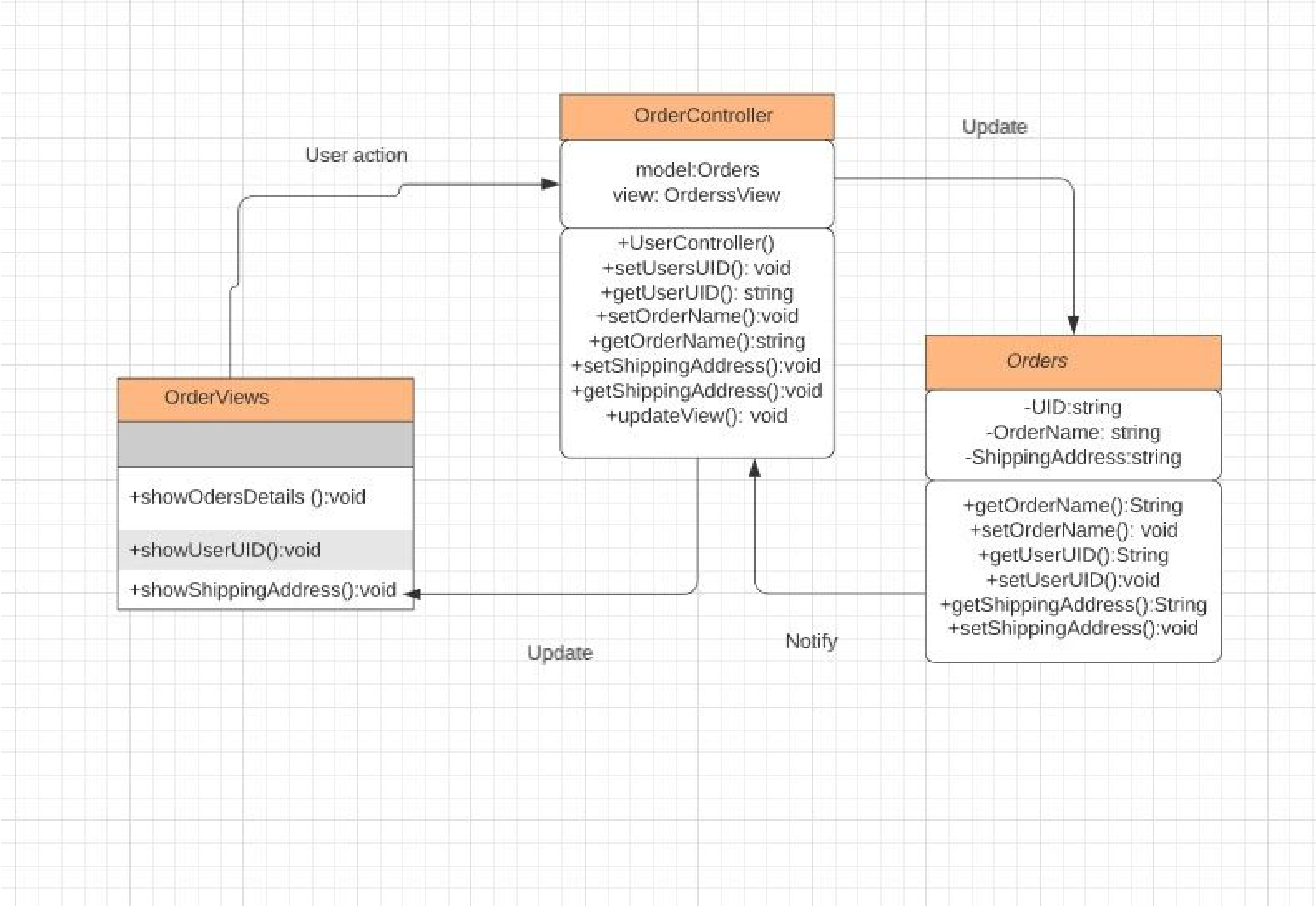
Section 4.0 Framework Model View Controller -MVC

4.1 For each Class diagram subsystem component

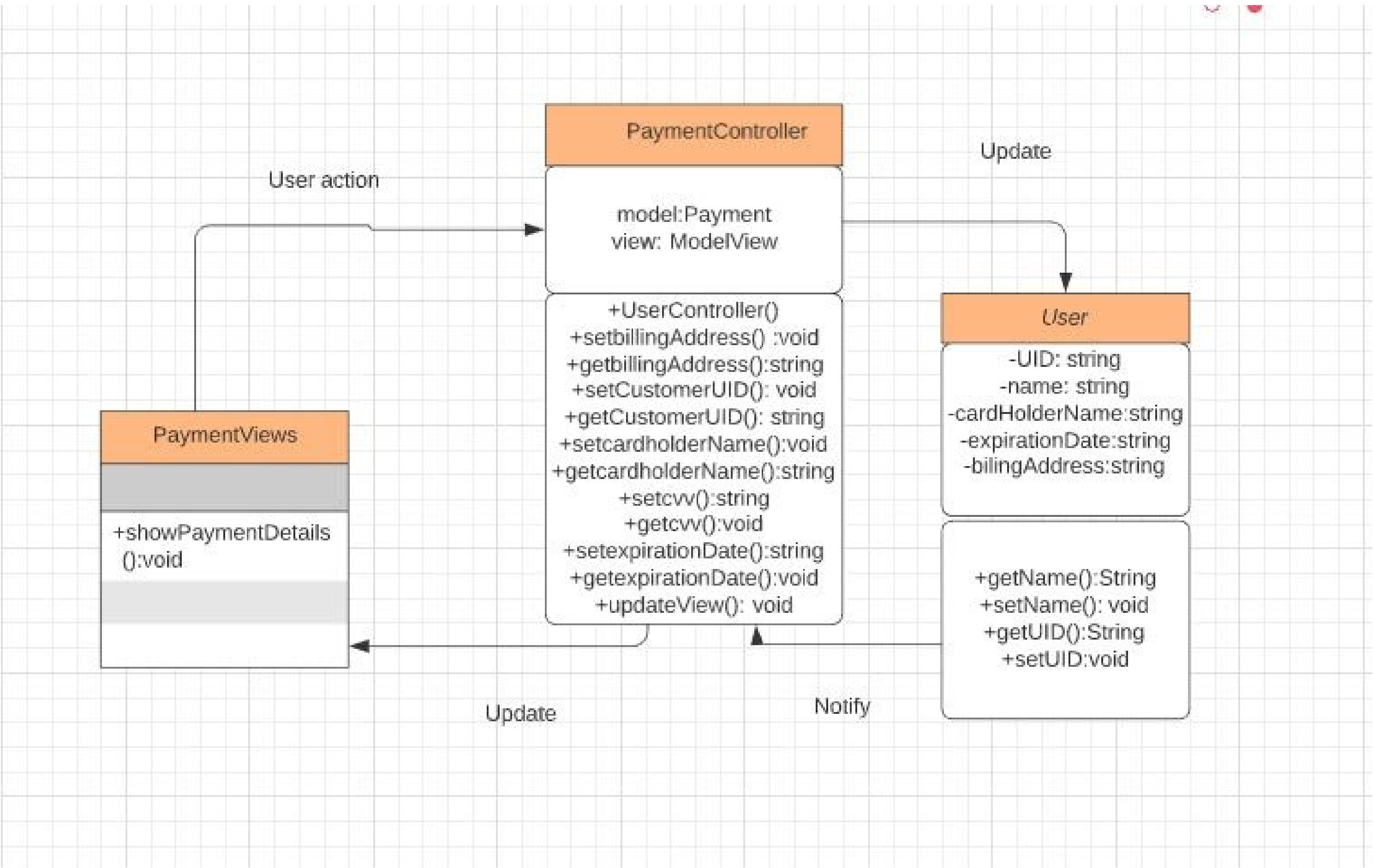
User



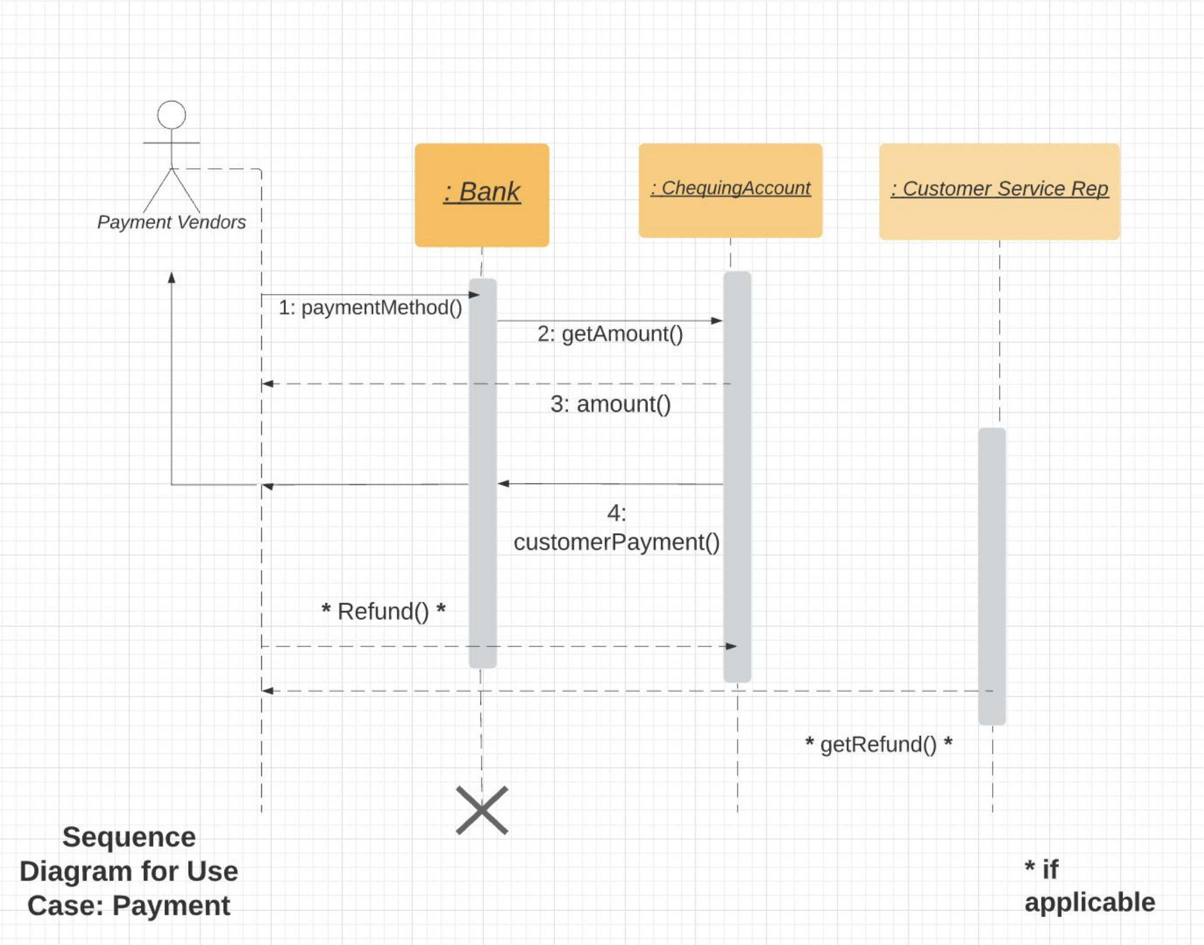
Order

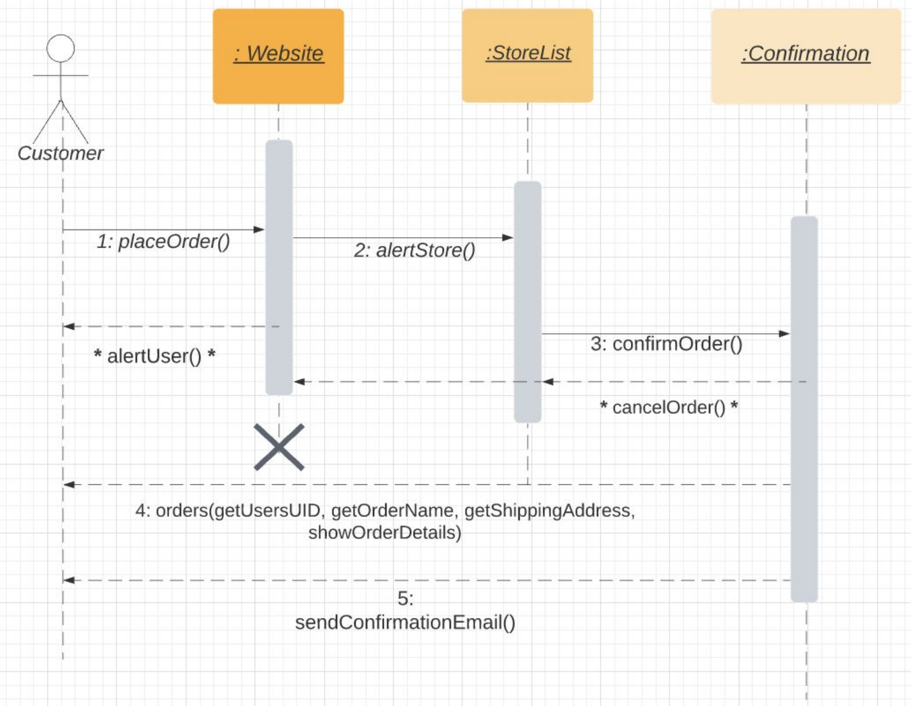

OrderController
model:Orders
view: OrderssView
+UserController()
+setUsersUID(): void
+getUserUID(): string
+setOrderName():void
+getOrderName():string
+setShippingAddress():void
+getShippingAddress():void
+updateView(): void
User action
Update
OrderViews
+showOdersDetails ():void
+showUserUID():void
+showShippingAddress():void
Orders
-UID:string
-OrderName: string
-ShippingAddress:string
+getOrderName():String
+setOrderName(): void
+getUserUID():String
+setUserUID():void
+getShippingAddress():String
+setShippingAddress():void
Update
Notify

Payment


PaymentController
model:Payment
view: ModelView
+UserController()
+setbillingAddress() :void
+getbillingAddress():string
+setCustomerUID(): void
+getCustomerUID(): string
+setcardholderName():void
+getcardholderName():string
+setcvv():string
+getcvv():void
+setexpirationDate():string
+getexpirationDate():void
+updateView(): void
User action
Update
PaymentViews
+showPaymentDetails ():void
User
-UID: string
-name: string
-cardHolderName:string
-expirationDate:string
-bilingAddress:string
+getName():String
+setName(): void
+getUID():String
+setUID:void
Update
Notify

4.2 Full Sequence diagrams



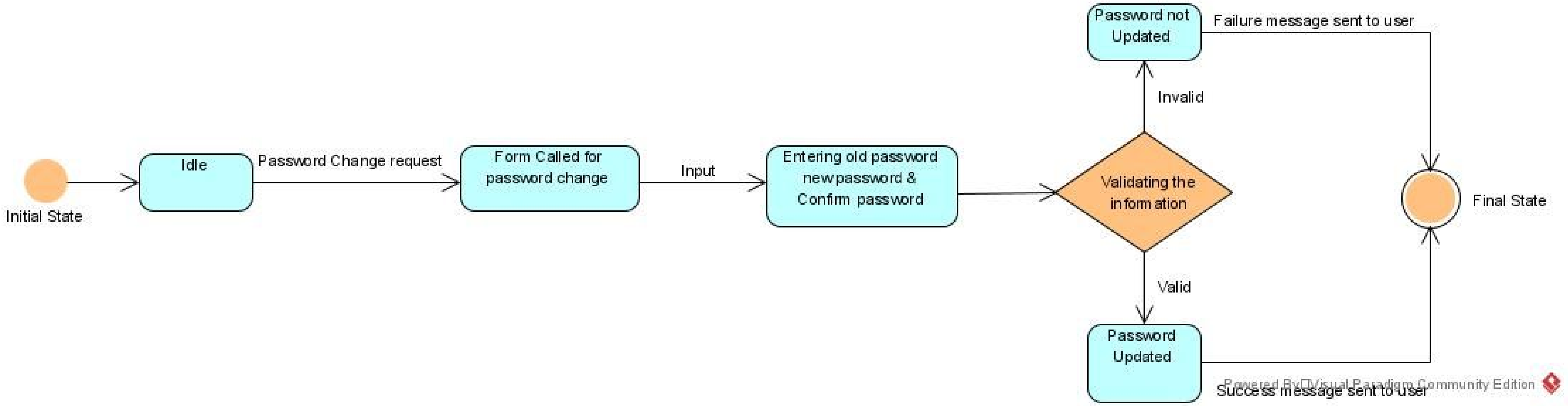


Sequence Diagram for Use Case: Order

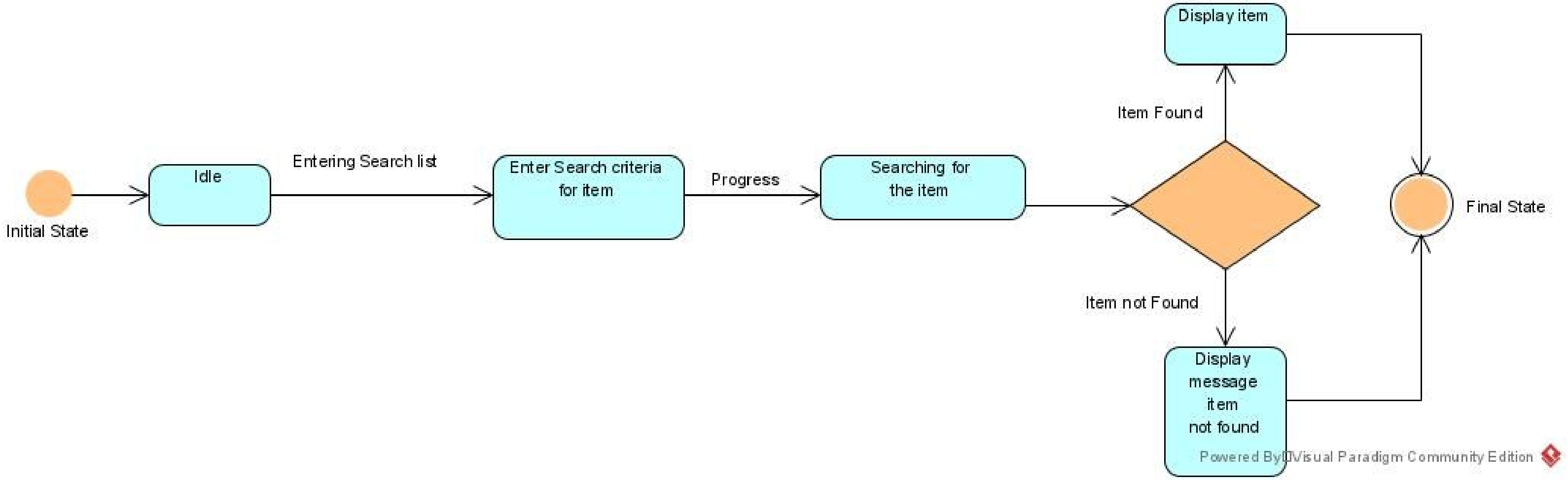
*** if applicable**

4.3 State Machine Diagrams

4.3.1 Password change request to system

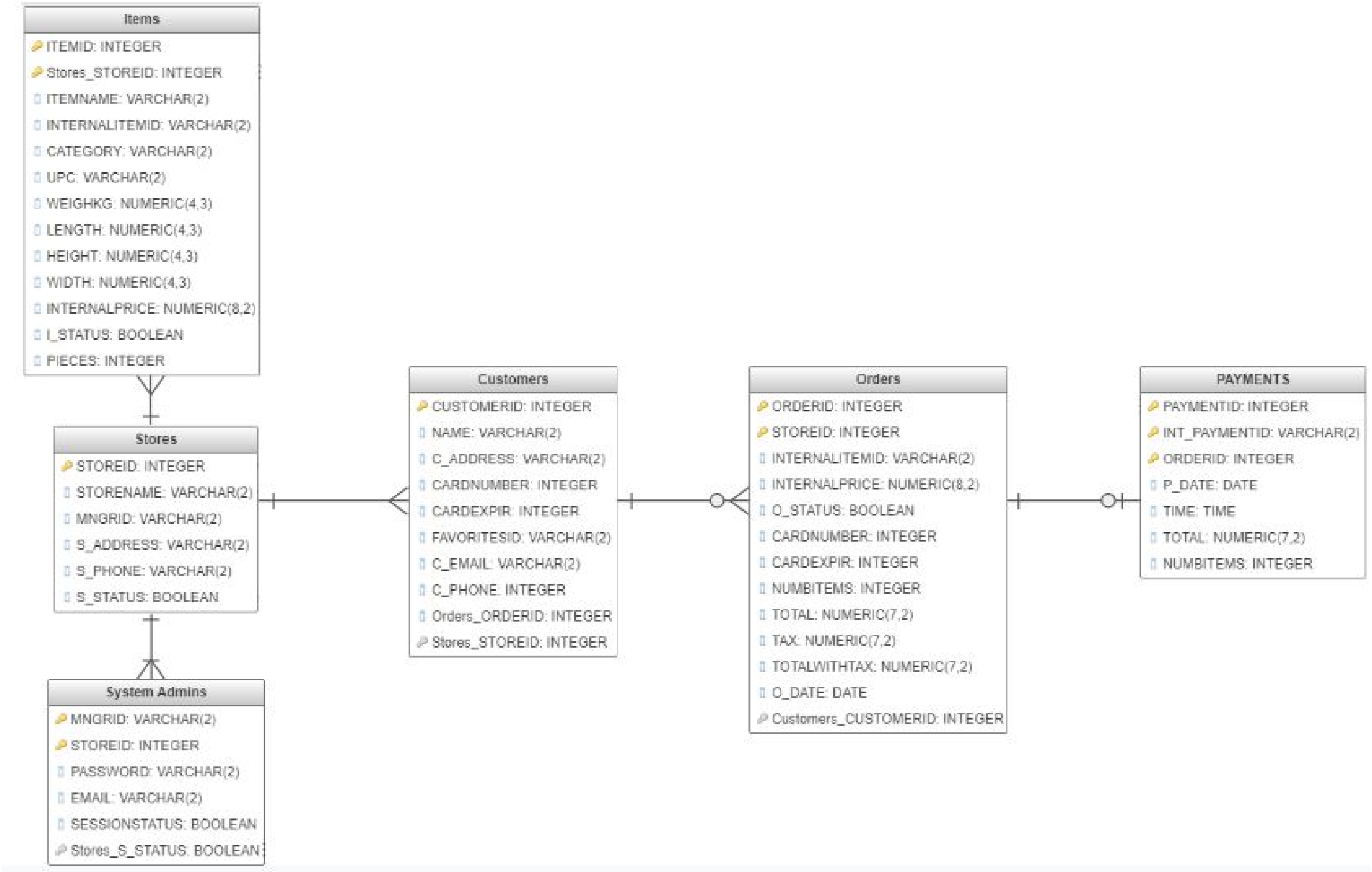

Password not Updated
Failure message sent to user
Invalid
Idle
Password Change request
Form Called for password change
Input
Entering old password new password & Confirm password
Validating the information
Final State
Initial State
Valid
Password Updated
Success message sent to user
Powered By Visual Paradigm Community Edition

4.3.2 Search for item in the system



Section 5.0 Data Layer

5.1. Database Schema



Items
ITEMID: INTEGER
Stores_STOREID: INTEGER
ITEMNAME: VARCHAR(2)
INTERNALITEMID: VARCHAR(2)
CATEGORY: VARCHAR(2)
UPC: VARCHAR(2)
WEIGHKG: NUMERIC(4,3)
LENGTH: NUMERIC(4,3)
HEIGHT: NUMERIC(4,3)
WIDTH: NUMERIC(4,3)
INTERNALPRICE: NUMERIC(8,2)
I_STATUS: BOOLEAN
PIECES: INTEGER

Stores
STOREID: INTEGER
STORENAME: VARCHAR(2)
MNGRID: VARCHAR(2)
S_ADDRESS: VARCHAR(2)
S_PHONE: VARCHAR(2)
S_STATUS: BOOLEAN

System Admins
MNGRID: VARCHAR(2)
STOREID: INTEGER
PASSWORD: VARCHAR(2)
EMAIL: VARCHAR(2)
SESSIONSTATUS: BOOLEAN
Stores_S_STATUS: BOOLEAN

Customers
CUSTOMERID: INTEGER
NAME: VARCHAR(2)
C_ADDRESS: VARCHAR(2)
CARDNUMBER: INTEGER
CARDEXPIR: INTEGER
FAVORITESID: VARCHAR(2)
C_EMAIL: VARCHAR(2)
C_PHONE: INTEGER
Orders_ORDERID: INTEGER
Stores_STOREID: INTEGER

Orders
ORDERID: INTEGER
STOREID: INTEGER
INTERNALITEMID: VARCHAR(2)
INTERNALPRICE: NUMERIC(8,2)
O_STATUS: BOOLEAN
CARDNUMBER: INTEGER
CARDEXPIR: INTEGER
NUMBITEMS: INTEGER
TOTAL: NUMERIC(7,2)
TAX: NUMERIC(7,2)
TOTALWITHTAX: NUMERIC(7,2)
O_DATE: DATE
Customers_CUSTOMERID: INTEGER

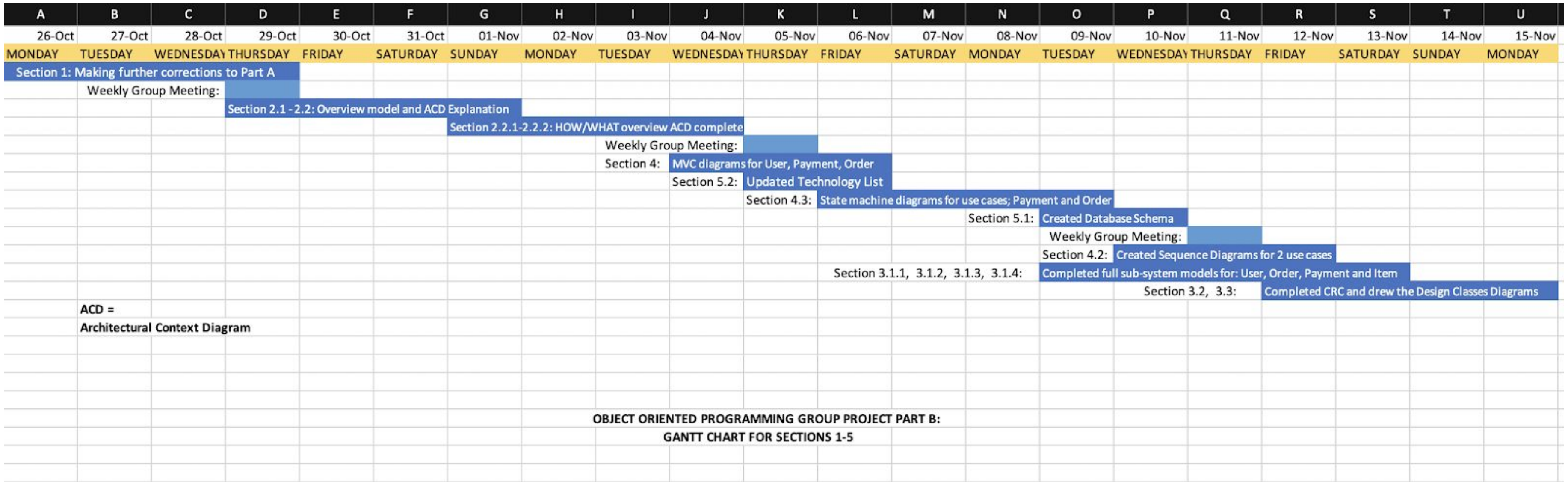
PAYMENTS
PAYMENTID: INTEGER
INT_PAYMENTID: VARCHAR(2)
ORDERID: INTEGER
P_DATE: DATE
TIME: TIME
TOTAL: NUMERIC(7,2)
NUMBITEMS: INTEGER

5.2 Update your Technology List

Changes in technology in the Server-side development programming and the Database.

Categories	Technologies Used	Comments
Platform	● Windows ● Android	The interface used for web view The interface used for the Mobile view
Programming Languages (Updated)	**Client-Side Development** ● Java ● C# **Server-Side Development** ● HTML 5 ● JavaScript ● CSS3 ● Express & NodeJS	Programming languages used for the application development (both back-end and front-end development)
Business Logic /Documentation (Updated)	● Visual Paradigm ● Visio ● Lucidchart	Tools used for the diagrams
Data side/Database (Updated)	● MongoDB ● Mongo Atlas	The database used for processing data.
Version Control	● GitHub	Online technology used for code backup and version control.

Section 6.0 Update the Gannt chart



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
26-Oct	27-Oct	28-Oct	29-Oct	30-Oct	31-Oct	01-Nov	02-Nov	03-Nov	04-Nov	05-Nov	06-Nov	07-Nov	08-Nov	09-Nov	10-Nov	11-Nov	12-Nov	13-Nov	14-Nov	15-Nov
MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY	MONDAY
Section 1: Making further corrections to Part A																				
		Weekly Group Meeting:																		
			Section 2.1 - 2.2: Overview model and ACD Explanation																	
						Section 2.2.1-2.2.2: HOW/WHAT overview ACD complete														
								Weekly Group Meeting:												
								Section 4:	MVC diagrams for User, Payment, Order											
									Section 5.2:	Updated Technology List										
										Section 4.3:	State machine diagrams for use cases; Payment and Order									
													Section 5.1:	Created Database Schema						
														Weekly Group Meeting:						
														Section 4.2:	Created Sequence Diagrams for 2 use cases					
													Section 3.1.1, 3.1.2, 3.1.3, 3.1.4:	Completed full sub-system models for: User, Order, Payment and Item						
																Section 3.2, 3.3:	Completed CRC and drew the Design Classes Diagrams			
	ACD =																			
	Architectural Context Diagram																			

OBJECT ORIENTED PROGRAMMING GROUP PROJECT PART B:
GANTT CHART FOR SECTIONS 1-5

PART C - Software Design Architecture Construction

Section 1 Corrections to Design Specifications Part B

1.1 Gannt chart corrections (modified in part A)
1.2 ERD section 5 adjustments (modified in part A)

Section 1.0 Software Design Patterns

1.1 Pattern : Composite

Composite pattern will allow the application to composite objects uniformly.

1.2 Pattern : State

State pattern main characteristic is to allow an object to alter its behavior when the object’s internal state changes, an object can have several states and each state can have different behavior.

It is applicable when the object's behavior depends on its state,and changes its behavior at runtime.

1.3 Pattern : Observer

When an object changes its state all the dependents should be notified and updated automatically, it is used to maintain consistency with related objects, it helps to avoid manual updates.

It is important to consider all the objects it affects in the implementation of the observer pattern and notify them.

Section 2.0 Using common software design patterns

2.1 Pattern : Composite

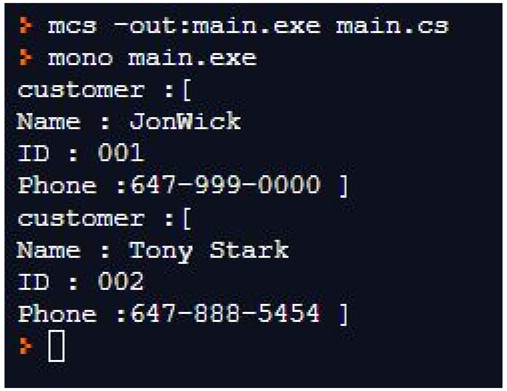
Name	Composite
Problem	stockChecKR app will create multiple numbers of users based on their role. In initial phases, roles will be limited to : - Customer. - storeManager. - IT staff All these roles will share the same structure but with different functions.
Solution	- We will create a user class. - Then customer class will inherit the user class. - Then the customer class will be responsible for initiating, printing, handling, and storing customers' data. - All customers will have the same class structure. - We will be able to differentiate customers based on customer ID

Example:

```
public class User {
  public string name;
  public string phoneNumber;
}
public class Customer : User {
  public string customerID;
  public string deliveryAddress;
  public Customer(string userName, string userPhone, string cusID, string delAdd){
    this.name = userName;
    this.phoneNumber = userPhone;
    this.customerID = cusID;
    this.deliveryAddress = delAdd;
  }
  public string toString(){
    return ("customer :[ \nName : " + name + "\nID : " + customerID + "\nPhone :" +  phoneNumber+" ]");
  }
}
```

Sample output:

<table>
<tr>
<td></td>
<td>



```
> mcs -out:main.exe main.cs
> mono main.exe
customer :[
Name : JonWick
ID : 001
Phone :647-999-0000 ]
customer :[
Name : Tony Stark
ID : 002
Phone :647-888-5454 ]
> 
```

</td>
</tr>
<tr>
<td>Benefits and consequences</td>
<td>

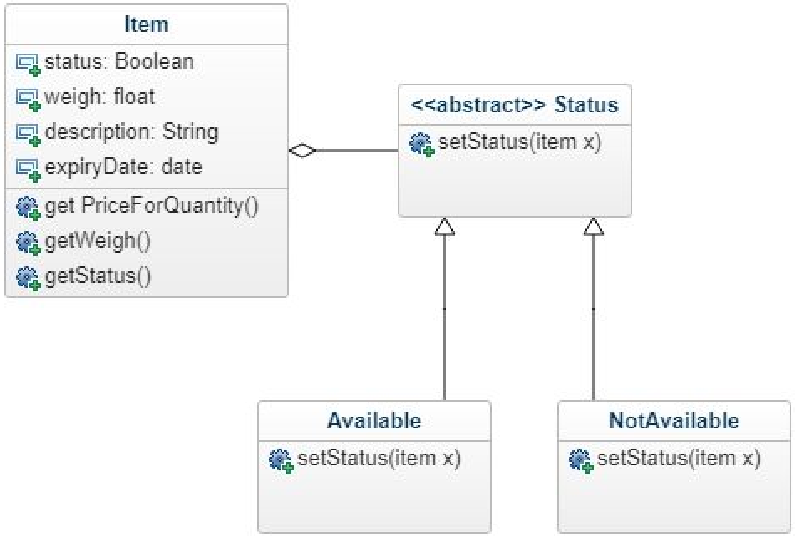
- All Customers will have the same information field to be filled.
- The app will be able to handle all types of users because fields and structure are well known.
- It will allow us to implement further speciation in the future, for example:

Ontario's Customers => will inherit customer class.

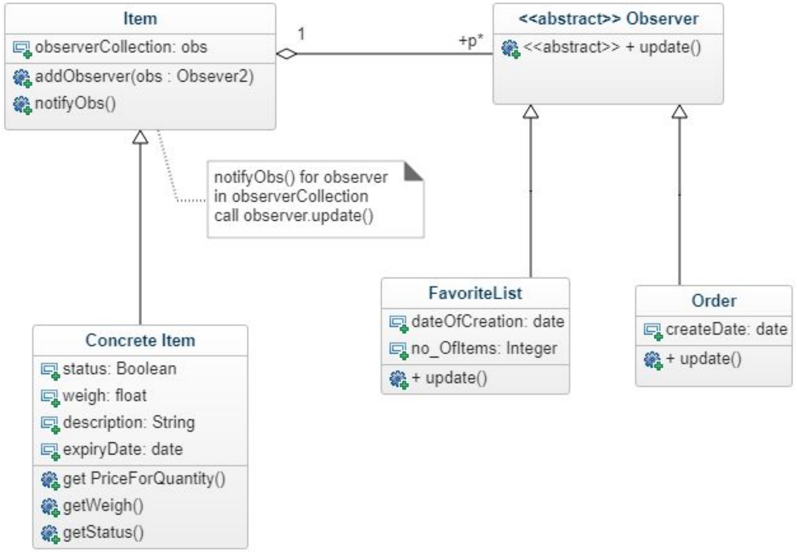
IT supervisor => will inherit IT Staff class

</td>
</tr>
</table>

1.2 Pattern : State

Name	State
Problem	Product availability needs to automatically change when product reaches expiry date to non available and notify the store administrator about this stock change.
Solution	Every time the store administrator adds a new product to the available stock, he or she should register the expiry date, this will create an automatic timer.
Graph	 Item status: Boolean weigh: float description: String expiryDate: date get PriceForQuantity() getWeigh() getStatus() <<abstract>> Status setStatus(item x) Available setStatus(item x) NotAvailable setStatus(item x)
Benefits and consequences	Store manager can know when a whole product batch has expired and dispose of it. Updates in product availability regarding expiry would be automatic. Users can certainly know in real time the number of products available.

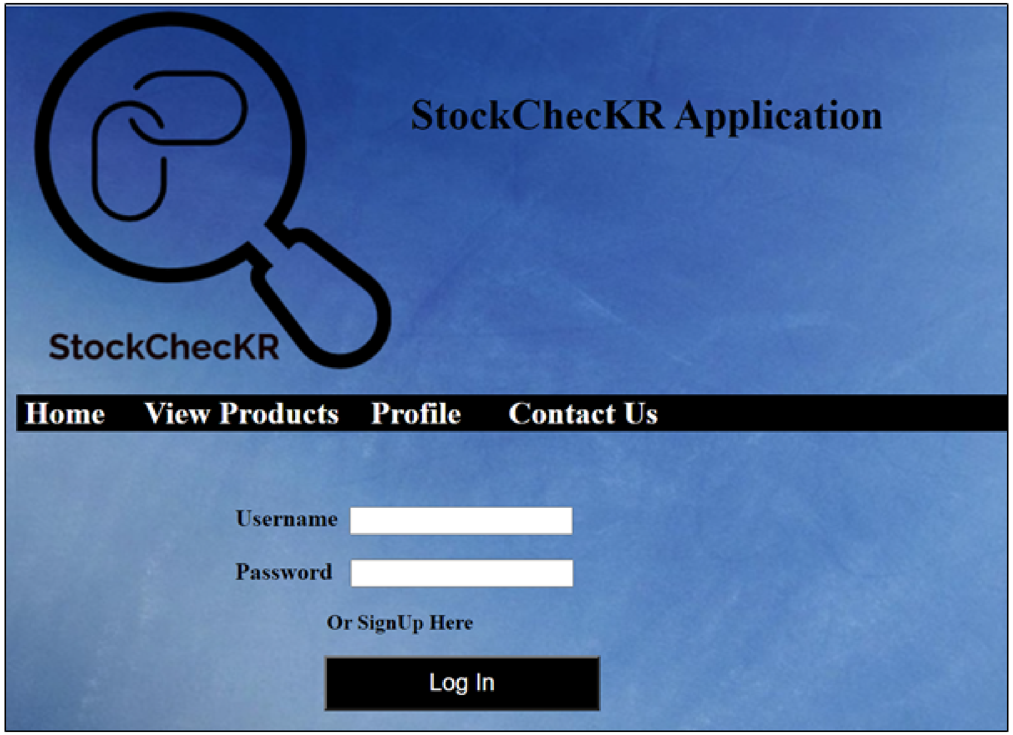
1.3 Pattern : Observer

Name	Observer
Problem	When modifying the Item class to adapt it to the state pattern, there are some other classes that might need to be updated and notified to improve usability and avoid manual error.
Solution	When a product or item runs out of stock by expiry date or any other reason it would automatically update the user’s favorite section and cart (in case to have the item added) and notify the user of the new updates.
Graph	 Item observerCollection: obs addObserver(obs : Obsever2) notifyObs() 1 +p* <<abstract>> Observer <<abstract>> + update() notifyObs() for observer in observerCollection call observer.update() FavoriteList dateOfCreation: date no_OfItems: Integer + update() Order createDate: date + update() Concrete Item status: Boolean weigh: float description: String expiryDate: date get PriceForQuantity() getWeigh() getStatus()
Benefits and consequences	Users would be able to know when a product of the cart is no longer available and pick between proceeding the order or switching to another grocery store. Users would be able to add new favorite items from other grocery stores when notified that a previous item is no longer available.

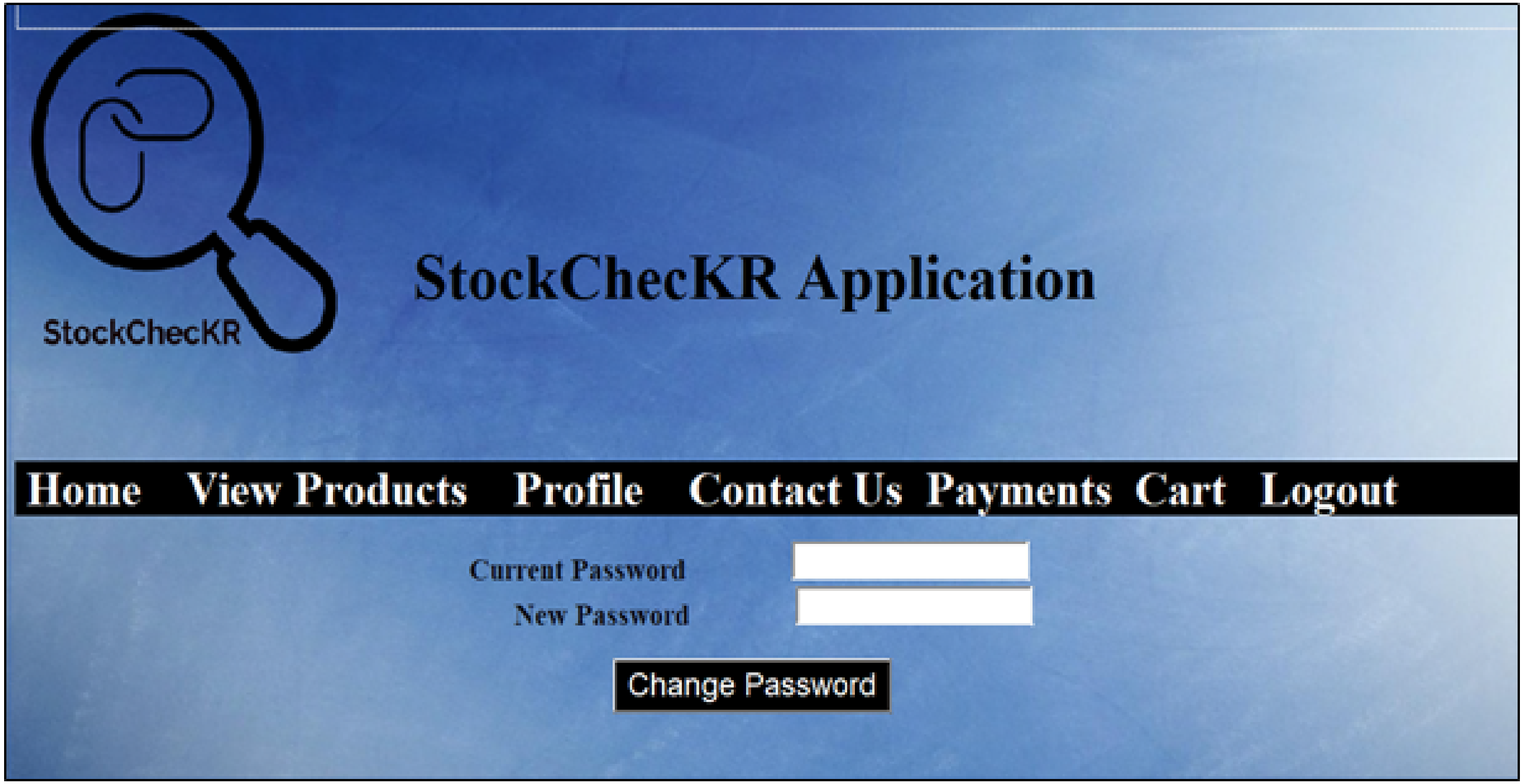
	It might be tedious for the user to switch all the items from one store to another, but the app allows adding items from different stores to the same order.

Section 3.0 UI/UX design

3.1 Home => Login View



3.2 After Login => If someone needs to change password

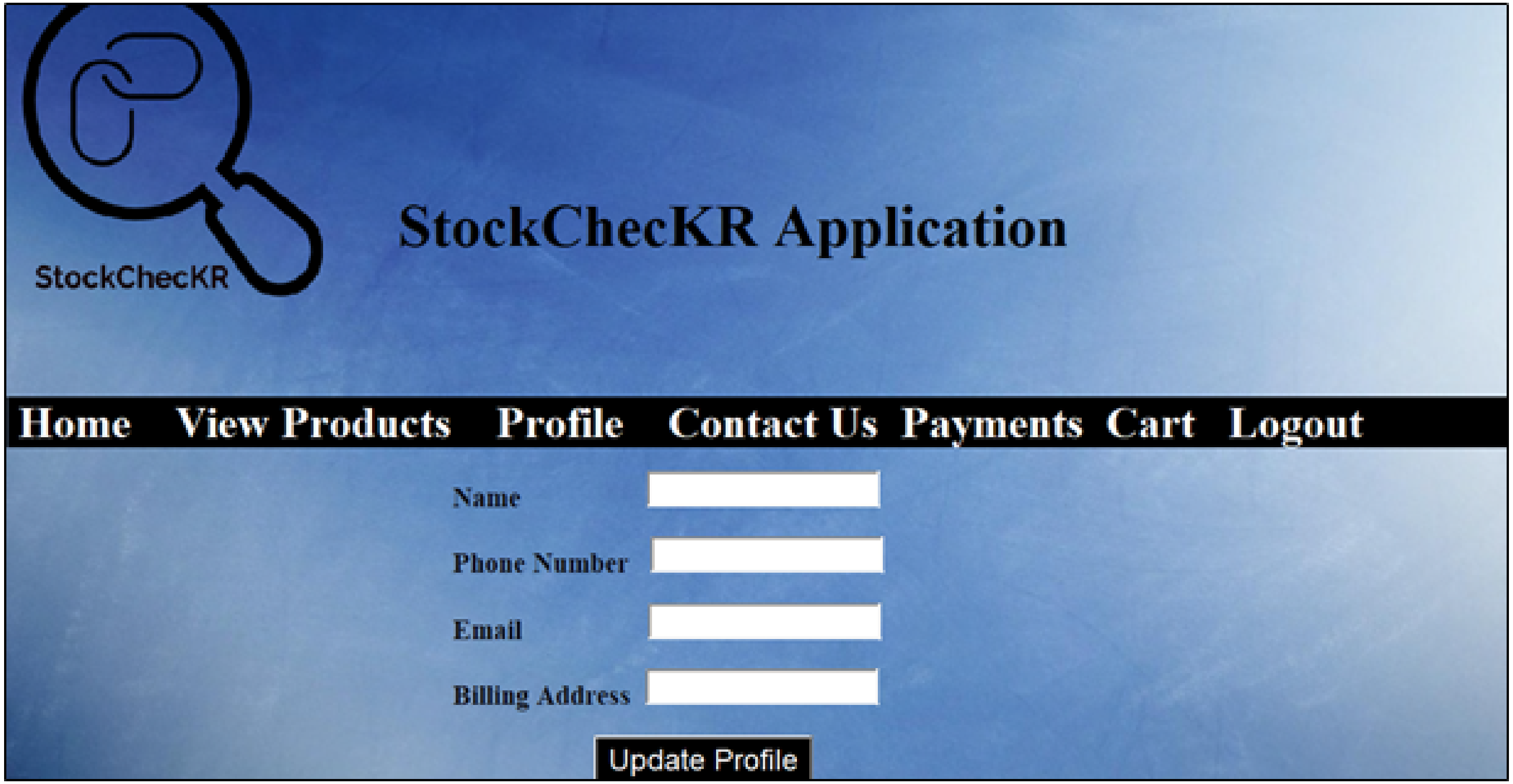


StockChecKR Application

Current Password

New Password

3.3 After Login => If someone needs to update profile



StockChecKR Application

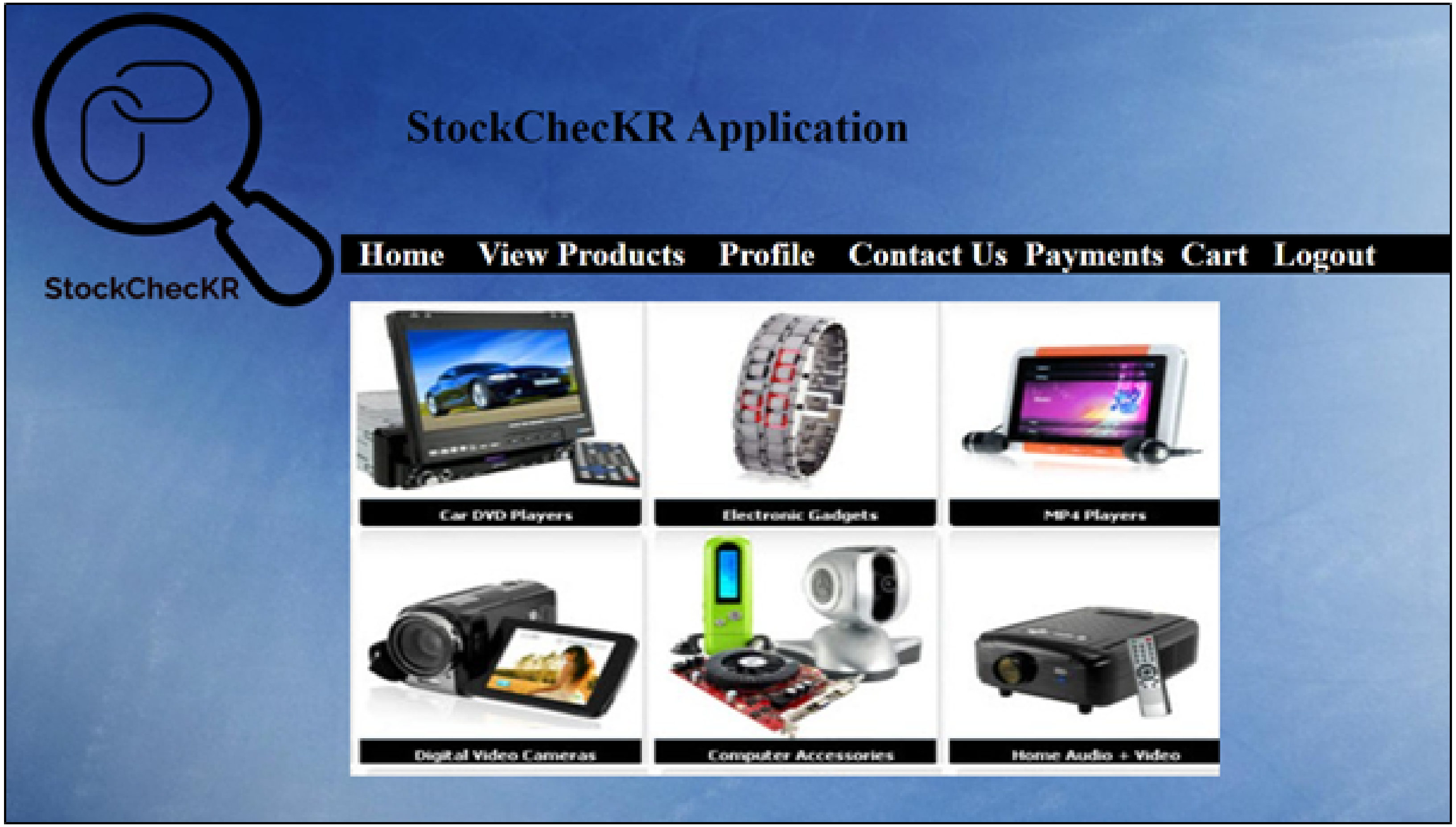
Name

Phone Number

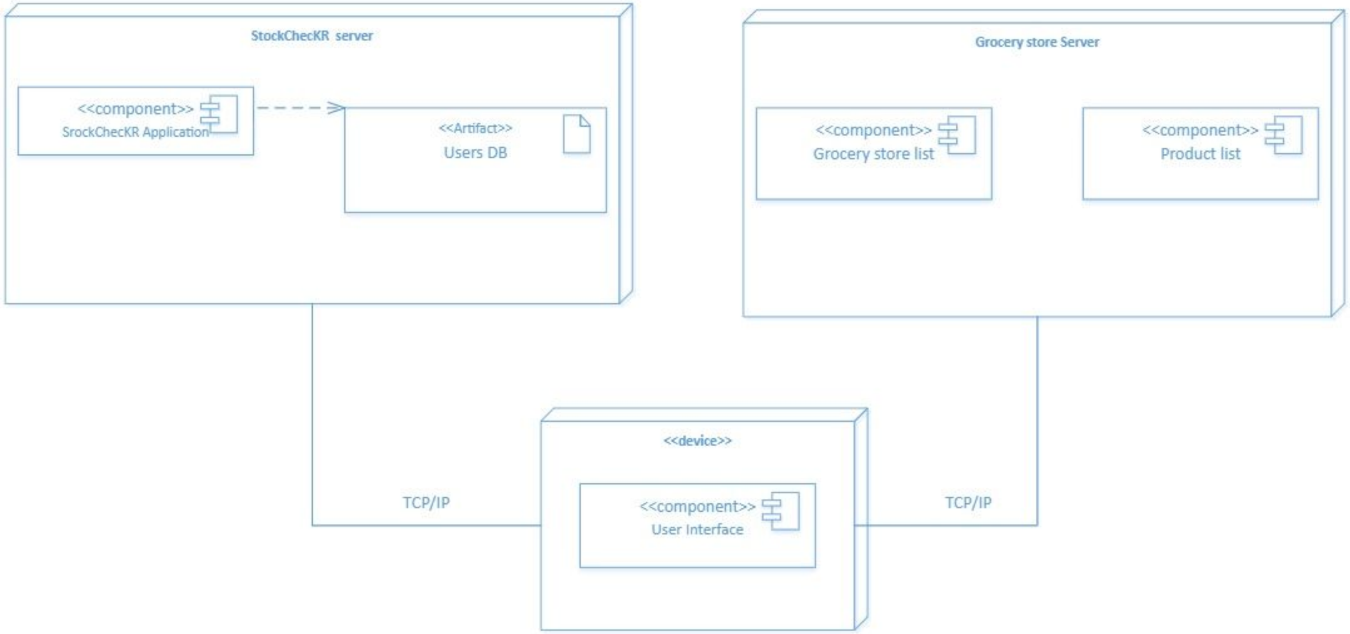
Email

Billing Address

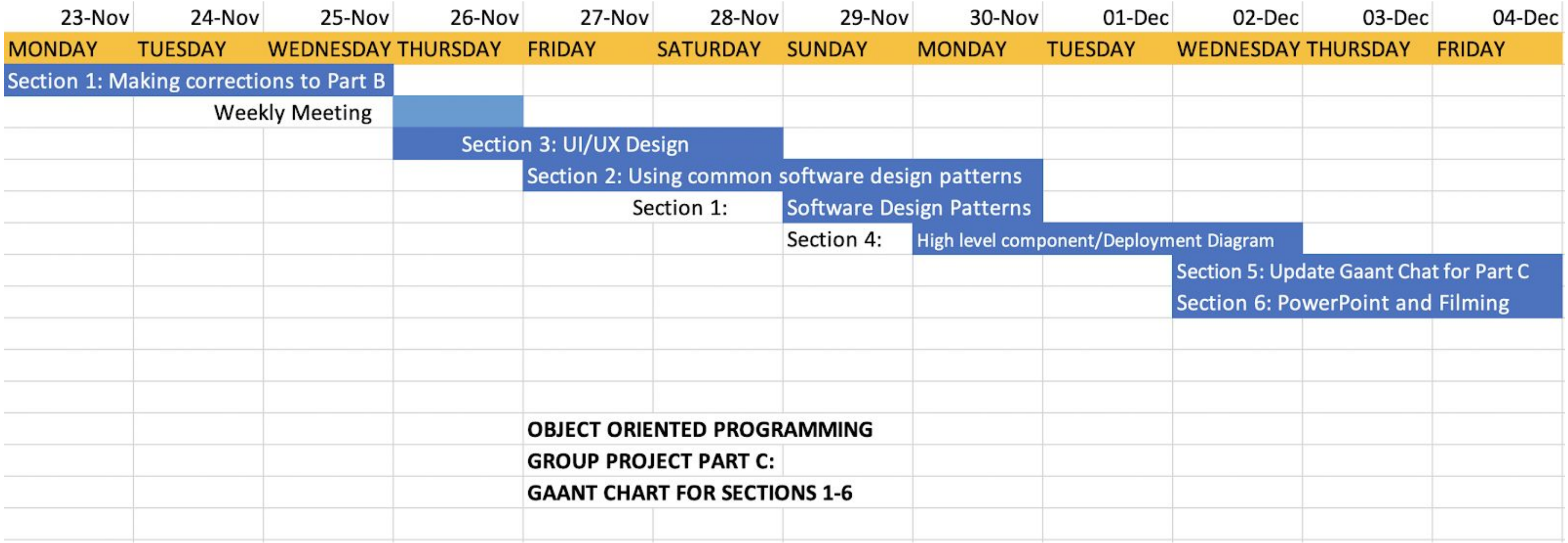
3.3 After Login => Searching for Items



Section 4.0 High-level Component/Deployment Diagram



Section 5.0 Update the Gannt chart to include Part C Tasks



23-Nov	24-Nov	25-Nov	26-Nov	27-Nov	28-Nov	29-Nov	30-Nov	01-Dec	02-Dec	03-Dec	04-Dec
MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
Section 1: Making corrections to Part B	■	■									
	Weekly Meeting		■								
			Section 3: UI/UX Design	■	■						
				Section 2: Using common software design patterns	■	■	■				
				Section 1:		Software Design Patterns	■				
						Section 4:	High level component/Deployment Diagram	■	■		
									Section 5: Update Gaant Chat for Part C	■	■
									Section 6: PowerPoint and Filming	■	■
				OBJECT ORIENTED PROGRAMMING							
				GROUP PROJECT PART C:							
				GAANT CHART FOR SECTIONS 1-6							

Section 6.0 Project Presentation

Prepare a Powerpoint presentation, a max of 12 slides for a group presentation to the rest of the class.