Zeus CompTech

Future Tech
Design Report
For Online Computer E-Commerce

Version <2.0>

Revision History

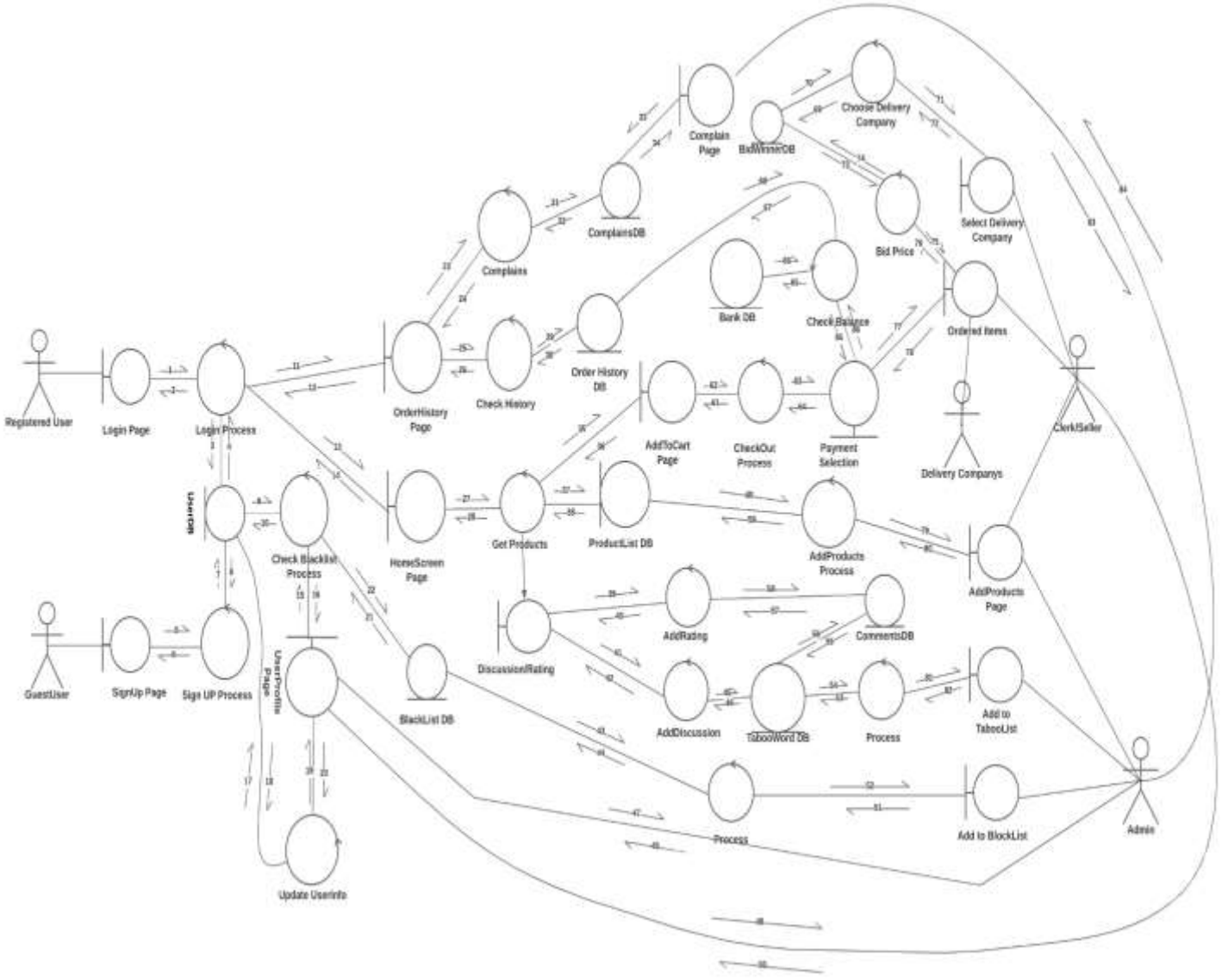
Date	Version	Description	Author
23/03/21	1.0		Tenzin Tashi Tenzin Choklang Shuren He Sam Mosavarpour Rezwana Kabita
25/04/21	2.0	Added the collaboration class diagram Scenarios for each use case E-R diagram for the entire system pseudo-code for main functionalities demonstrate major GUI screens	Tenzin Tashi Tenzin Choklang Shuren He Sam Mosavarpour Rezwana Kabita

Table of Contents

Software Requirements Specification

1. Introduction

The purpose of this project is to build an Online ecommerce website system where users can shop for pc, pc parts & components easily. In addition, this system will help managers to handle daily ecommerce issues such as managing users/employees, add or remove words from taboo lists, handling customers' compliments and complaints. This system also allows the customers to browse and order their preferred PC or PC Parts, and allows them to file a rating(1-5) everytime they place an order. This document will also serve as reader friendly for everyone who will take interest in understanding this software, and various other technical dependencies.



\\

1. Enter info 2. Delete info 3. Go to database 4. Retrieve login info 5. Guests sign up info

6. retrieve guest signup info 7. Check if exist in DB 8. Return does not exist 9. Check if in blacklist

10. Return if it is in blacklist 11. Show order history 12. Return order history 13. Go to home screen page

14. Return from home screen page 15. show user profile age 16. Return user profile page 17. Input user info in

18. Return user info from DB 19. edit/modify user info 20. Return updated user info 21. Return from blacklist

22. Check if it is in blacklist 23. Check for complaints 24. show/display complains 25. Check for order history

26. Return order history 27. Go to product 28. Return product 29. Check order history in DB

30. return order history in DB 31. Input complains 32. Return complains 33. Filter return filtered complain

34. Filter complaint in DB 35. Add to cart 36. Return to cart 37. check if product exist in DB

38. return product info from DB 39. Add rating 40. Return rating 41. Add discussion 42. Return discussion

43. Return blacklisted 44. Add blacklist in DB 45. Add discussion 46. Return discussion

47. Update admin profile information 48. Return admin profile information 49. Update clerk information

50. Return clerk information 51. Process to block list 52. return block list 53. Process taboo words

54. Return filtered words 55. Return comment from DB 56. Add comment to DB 57. Return rating

58.add ratings 59. Add product by Admin/Clerk process idem for bid 60. remove product by admin/clerk

61. Return from check out process 62. check out process 63. Select payment 64. Return if payment denied

65. Check bank DB for balance 66. Return the balance form bank DB 67. After order put in order history Db

68. If not purchased return to check balance 69. Choose delivery company 70. return if delivery info

71. Select the reasonable delivery company 72. return tracking from delivery 73. Bid for shipping

74. return if bid did not win 75. return bidding information 76. process item for bid 77. return payment info

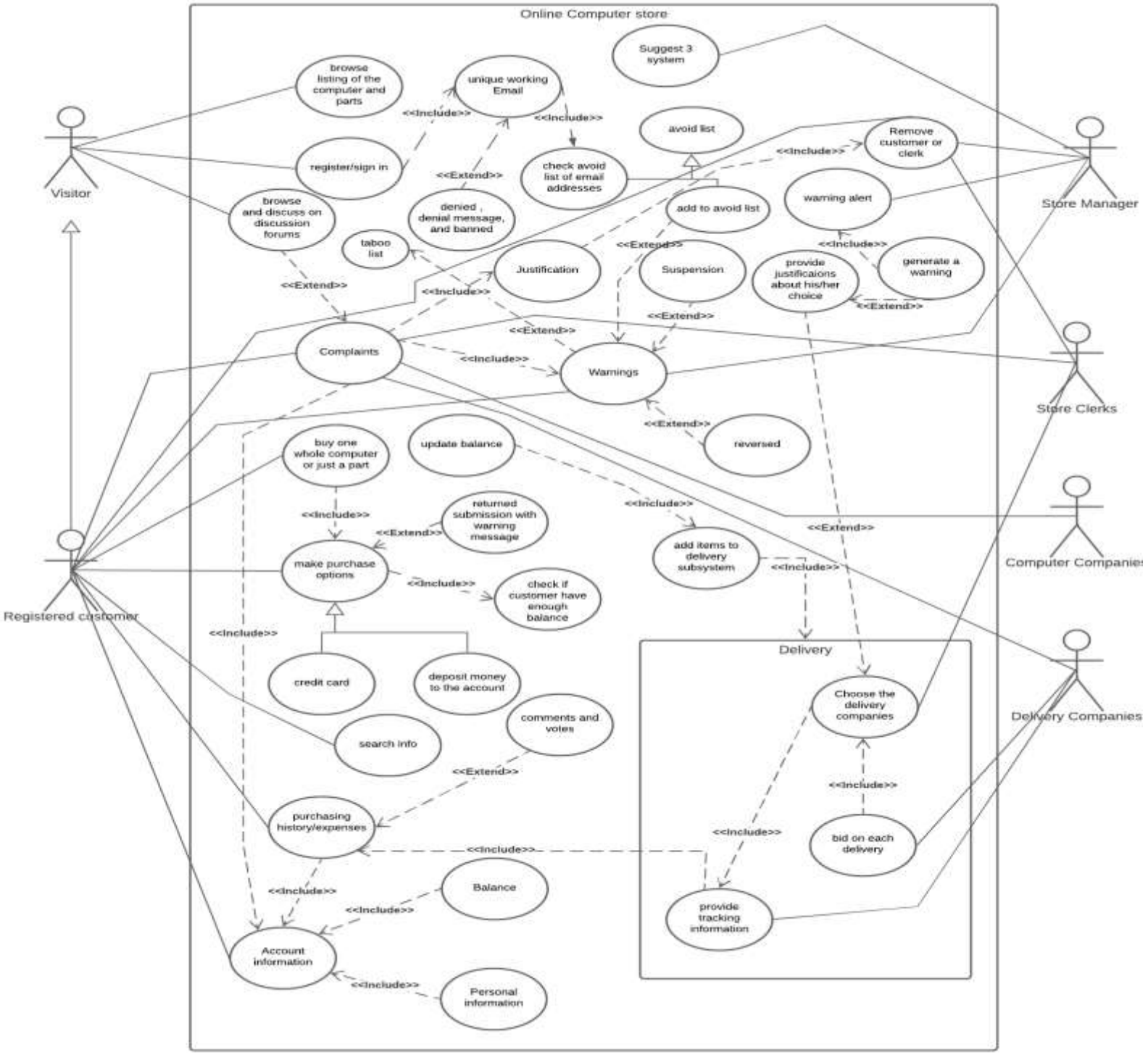
78. payment type selection 79. Process the product from cart 80. Return the product if removed

81. Add to taboo list 82. return the taboo list 83.return complain 84. Access complaint page 85. Return balance

86. Check balance in bank DB

2. All use case

2.1 Use-Case Model Survey



2.2 Use-Case Reports

Store Manager to “Suggest 3 system” use case relationship:

- Store manager suggests 3 systems for the home page.

Visitor to “browse listing of the computer and parts” use case relationship:

- Visitors can browse the listing of the computers and parts.

Visitor to “browse and discuss on discussion forums” use case relationship:

- Visitors can browse through the discussion forums.

Visitor to “register/sign in” use case relationship:

- Visitors can also register to become a register customer.
- “Register/sign in” use case to “unique working Email” use case relationship:
    - To become a registered customer, the visitor needs a unique working Email.
- “Unique working Email” use case to “check avoid list of email addresses” use case relationship:
    - The email address is checked if it's present in the avoid list of email addresses.
- “Denied, denial message, and banned” use case to “unique working Email” use case relationship:
    - If the email address is present in the avoid list, then the visitor is denied being the registered customer and a denial message is sent, any future applications will be denied without reply.

Registered customer to “make purchase option” use case relationship:

- Registered customers can only make purchases if balances is supplied through:
    - “Credit card” to “make purchase option”
        - If the registered customer provides working credit card
    - “Deposit money to the account” to “make purchase option”
        - If the registered customer deposit money to the account

Registered customer to “search info” use case relationship:

- Register customer can search info

Registered customer to “purchasing history/expenses”:

- Registered customers can check their purchased history or expenses.
- “Comments and votes” to “purchasing history/expenses” use case relationship:
- Registered customers can comment or vote on the items they purchased.

Registered customer to “buy one whole computer or just a part” use case relationship:

- Registered customers have an option to either buy a whole computer or just parts.
- “Buy one whole computer or just a part” to “make purchase option” use case relationship:
- Registered customer can make a purchase of either whole computer or just a part

Registered customers can also browse the discussion forms or start a discussion, which is inherited from the Visitor.

“make purchase option” to “check if customer have enough balance” use case relationship:

- When a purchase option is made the system check if the register customer has enough balance or not
- “check if customer have enough balance” to “update balance” use case relationship:
- If the registered customer has enough balance, then the system charges the customer and update the *balance.*
    - “update balance” to “add items to delivery subsystem” use case relationship:
        - Once the balance has been updated then the items will be added to the delivery subsystem.

"returned submission with warning message" to "make purchase option" use case relationship:

- If the register customer doesn't have enough money on their account, then the purchase option is returned with a warning message.

Registered customer to "Account information" use case relationship:

- Registered customer will have access to their account information:
    - The account information will include the customer's balance, personal information, past complain, past purchases, and past votes on past purchases.

Registered customer to "complaints" use case relationship:

- Registered customers can complain about the item they purchased, the clerk, or the delivery companies he/she dealt with.

Store clerks or delivery companies or computer companies to "Complaints" use case relationship:

- The complained clerks or delivery company or computer companies will have access to the complaints made by the customer.
- "Complaints" to "justification" use case relationship:
    - The complained clerks or delivery company or computer companies need to provide a justification to each of the complaints.
- "Complaints" to "Warnings" use case relationship:
    - For each complaint a warning will be generated to the respected party.
- "Warnings" to "Suspension" use case relationship:
    - If the individual party gets 3 warnings the system will suspend their all rights automatically.
- "Reversed" to "Warnings" use case relationship:
    - If the complaint started by the customer gets reversed the customer will get one warnings

Store Manager to "Warnings" use case relationship:

- The manager has the power to suspend any parties even without getting 3 warnings.
- "add to avoid list" to "Warnings" use case relationship:
- Any customer with 3 complaints will be added to the avoid list
  "add to avoid list" and "check avoid list of email addresses" to "avoid list" use case relationship:
- Both add to avoid list and check avoid list of email addresses inherited from the avoid list.
  "taboo list" to "warning" use case relationship:
- If a word from a taboo list is mentioned in the discussion form it will go to warnings use case which will then go to complaints use case.

Delivery Subsystem:

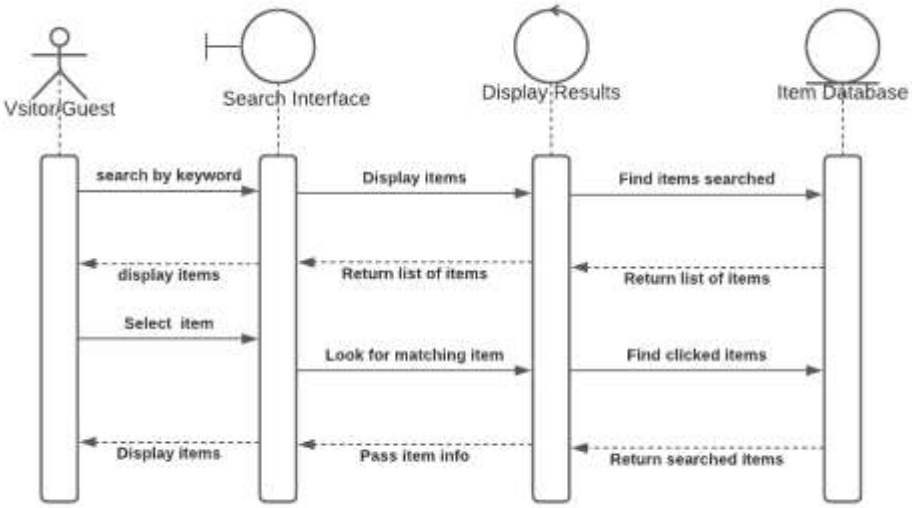
- Delivery companies to “bid on each delivery” use case relationship:
  - 2 delivery companies will bid on each delivery available after customer purchase.
- Store clerks to “Choose the delivery companies” use case relationship:
  - Store clerks will choose the delivery company based on the price the delivery companies give.
- “Provide justifications about his/her choice” to “choose the delivery companies” use case relationship:
  - If the store clerk doesn’t choose the delivery company with the lowest bidding, then he/she needs to provide a justification of their choice.
- “Generate a warning” to “provide justifications about his/her choice” use case relationship:
  - System will generate a warning if the store clerk doesn’t provide a justification of their choice.
- “Generate a warning” to “warning alert” use case relationship:
  - The system generated warning will be sent to the warning alert as a possible cheating by the store clerk.
- Store Manager to “warning alert” use case relationship:
  - Store manager will receive a warning alert of possible cheating.
- “Choose the delivery companies” and Delivery companies to “provide tracking information” use case relationship:
  - Once the delivery company has been chosen it will provide tracking information.
- “Provide tracking information” to “purchasing history/expenses” use case relationship:
  - Registered customers will have access to the tracking information of the items they purchased.

2.3 Collaboration/ Sequence Class diagrams:

1. **A Visitor/ Guest visits the page without sign-in:**

   Normal Scenario: A guest/ visitor visits the page and searches for computers or parts of computer at the search bar. The user’s page is populated with items he/she searched for. the visitor/ guest can now see the items.

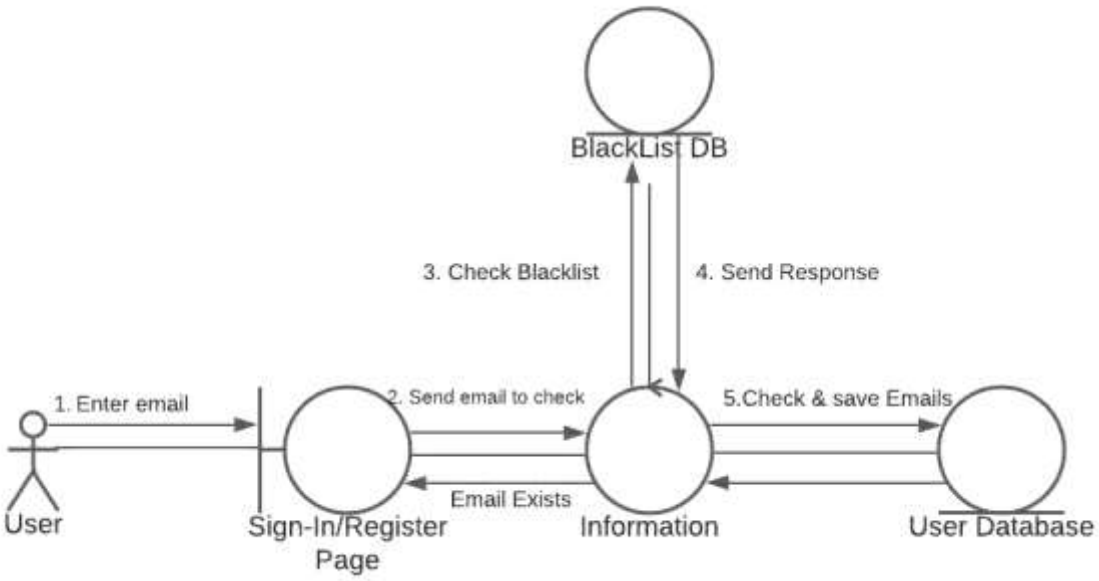
   Exceptional Scenario: No exceptions here.



2. A visitor Sign-in for the system:

Normal Scenario: A guest/ visitor can visit the page and register using their email address. When a guest enters his/her email address is checked with the existing email addresses from the DB.

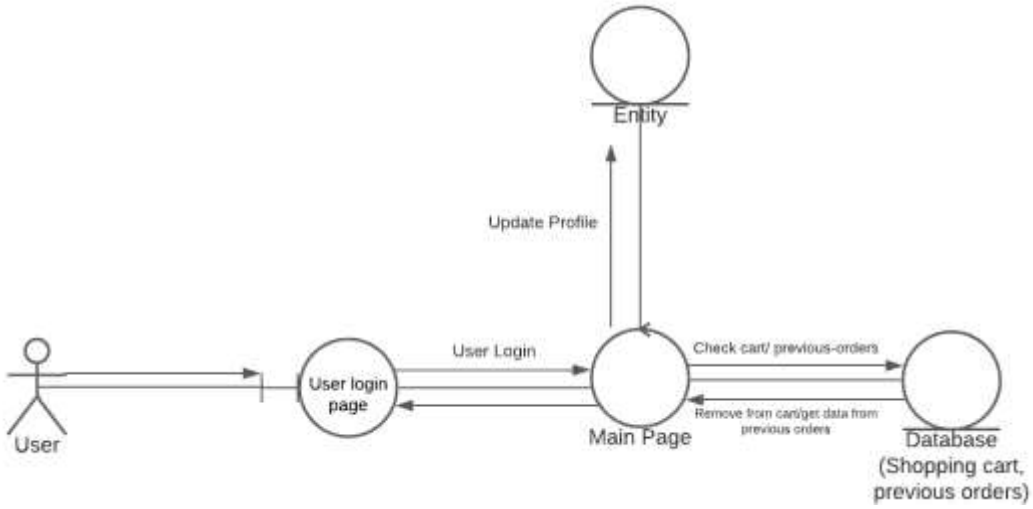
Exceptional Scenario: If the provided email address matches any already existing/ blocked email address he/ she will be denied and an email will be sent to the email address.



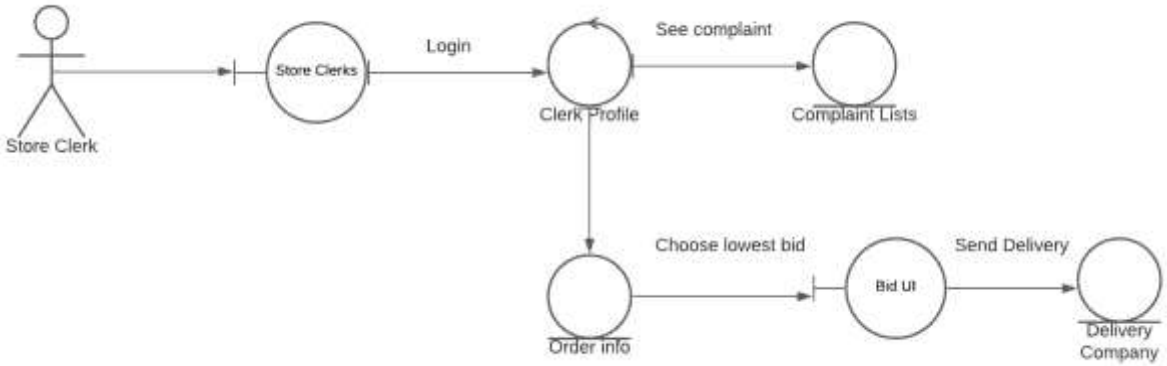
3. Registered Customer :

Normal Scenario: A user can log in and see his/her account and personal information and update it. He/she can view items, add items in his/her cart, remove items from his/her cart , and get previous order infos associated with this account.

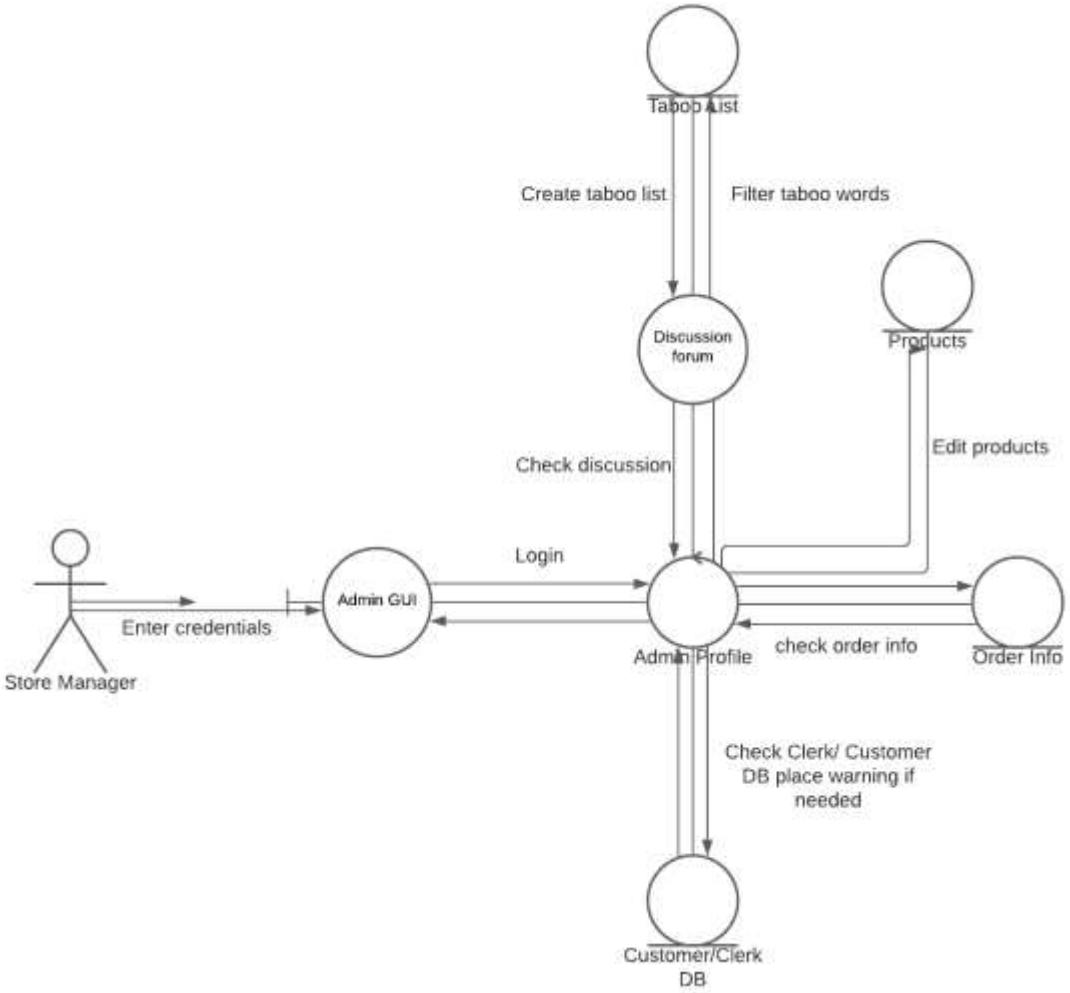
Exception: N/A



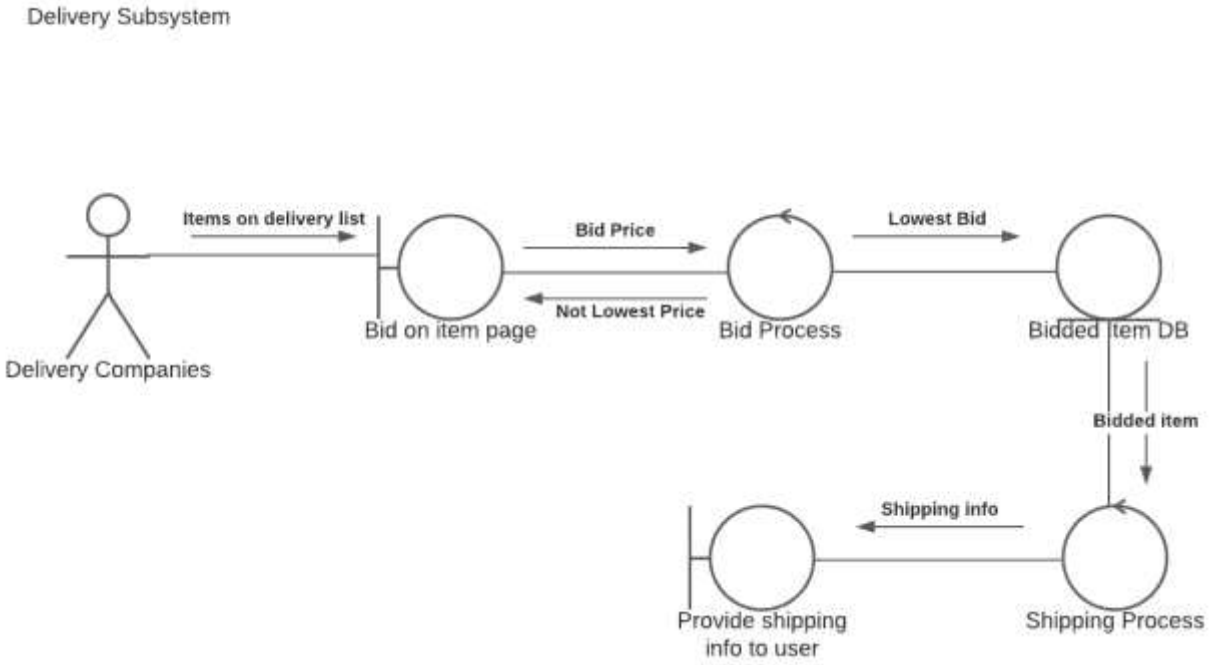
4. Store-Clerk:



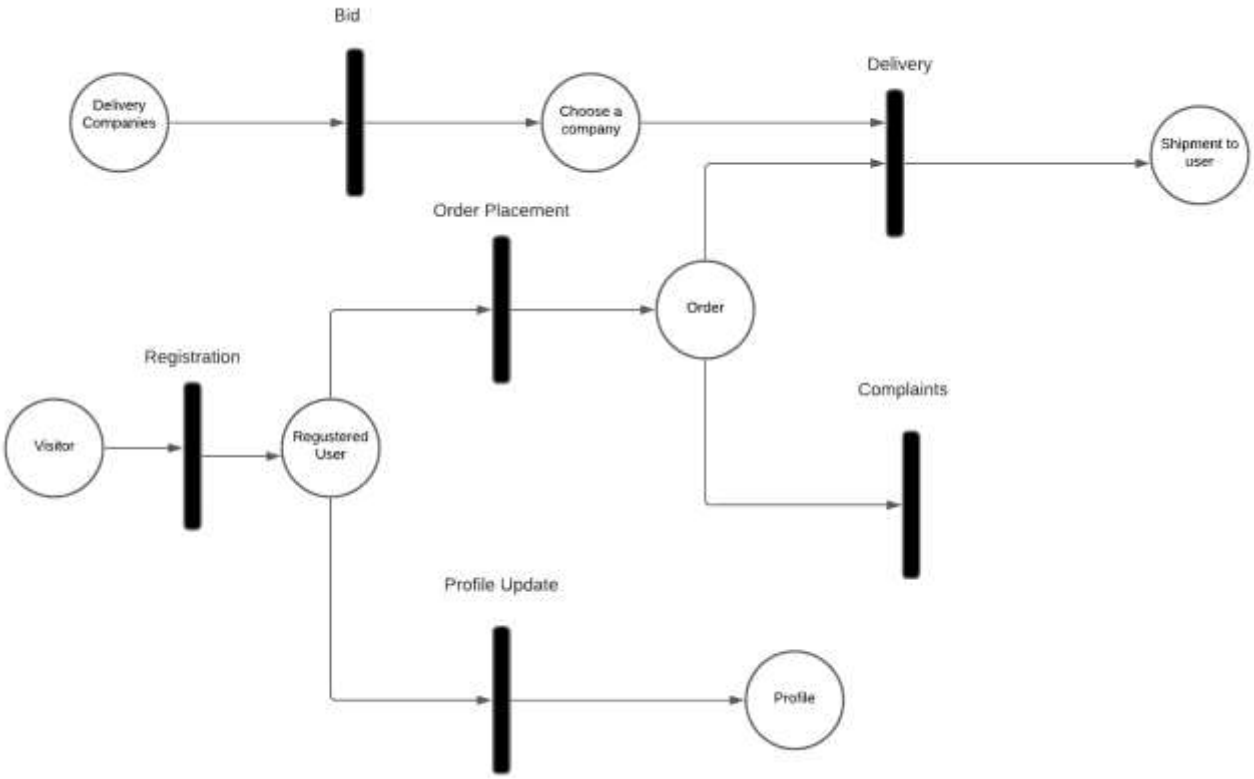
5. Admin/ Store Manager:



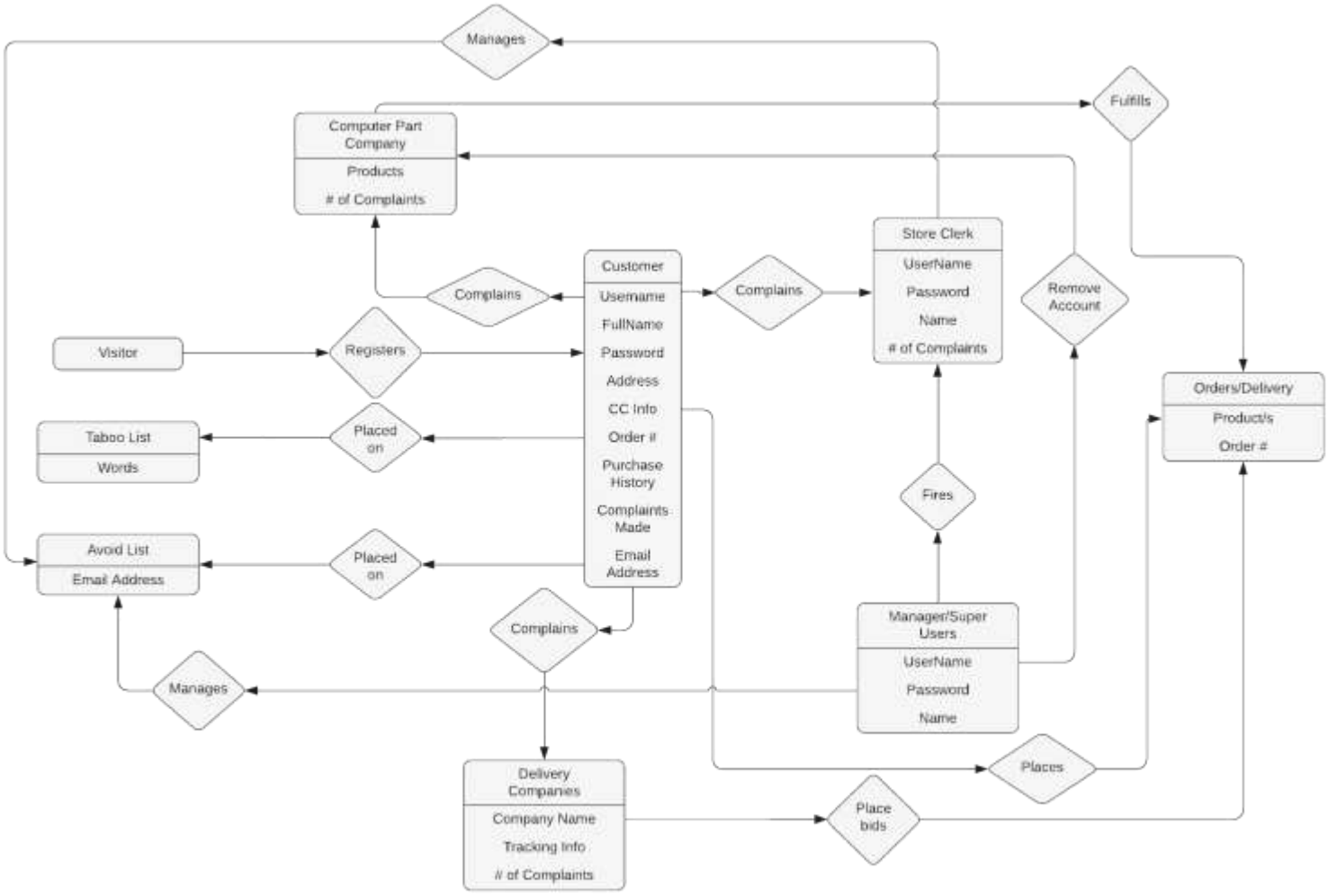
6. Delivery subsystem:



2.4 Petri-Net Diagram for few use-cases:



3. E-R Diagram:


Manages
Fulfills
Computer Part Company
Products
# of Complaints
Store Clerk
UserName
Password
Name
# of Complaints
Customer
Username
FullName
Password
Address
CC Info
Order #
Purchase History
Complaints Made
Email Address
Complains
Complains
Remove Account
Visitor
Registers
Taboo List
Words
Placed on
Orders/Delivery
Product/s
Order #
Fires
Avoid List
Email Address
Placed on
Complains
Manager/Super Users
UserName
Password
Name
Manages
Delivery Companies
Company Name
Tracking Info
# of Complaints
Places
Place bids

4. Detailed Design:

For EVERY method use pseudo-code to delineate the input/output and main functionalities

Brief introduction：

Class: User

Intent: Contains the attributes inherent to all users.

Contains the methods :“LogIn”、“register in”、 “UpdateUser”、“Get_id”、”DeleteUser” and ”ResetPassword”

- **LogIn(username, password)**

```
{ const user= await User.findone(username,password)
If(user)
If(!blocked)
        Res.send(user.information)
Else return (print (‘’))
Else return (print(“”))
//Username and password are passed to the database. Return to the new page link}
```

- **register(username, password)**

```
{ user=new user
User.name=username
User.password=password
Add user to register users
Print(“  ”)
//need Username and password to create a new user into the service .
Return the message: register successfully}
```

- **UpdateUser(username, password)**

{ const user= await User.findone(username,password)

If(user)

` User.name=somethingnew

User,birthdate=…….

User.sex=……

User.save();

Return Print(“”)

Else return Print(“ ”)

//send a request to the service expect to update the user information, inside the function,

add new phone number、email address、birthdate and gender. Save the data and return the message }

- **Get_Id(string email)**

{ if(user.email=email)

Something=user.id

return Print(“”+something)

else return print(“”)

//send a request to the service expect to get the id back by using his or her email address., If no email address is found in the database, the function returns the current User object and print the message}

- **DeleteUser(username, password)**

{ const user= await User.findone(username,password)

If(user)

User.remove();

Return Print(“”)

Else return Print(“ ”)

//inside the function, use remove method to run. If no username is found in the database or wrong password, the function returns the current User object and prints the message ask to input again . Else, it print some message}

● ResetPassword (string email)

{ if(user.email=email)

User.password=input(“”)

return Print(“”+something)

else return print(“”)

// send a request to the service expecting to reset the password by using his or her email address.}

Class Product:

Intent: brief introduction how our products create the attributes of our products .

●Product.name( )

// set the font size and weight of product name

●Product.brand( )

//set the font size and color of product brand

●Product.price( )

//set the font size and weight of product price

●Product.Screen( )

//set the display flex-direction, justify-content ,height of Product

●Product-image( )

//set the max-width, max-height of Product image

●creating product (String name, Int Id , ……)

{ const product =new product

Product.name =name

Product.Id = Id

Product.save()

Print()

//return the new product and add the new product into the product database }

●UpdateProduct(ProductObject)

{ if(Product)

Product.name=ProductObject.name

Product.price=ProductObjetc.price

Product.save()

Return Print(“ ”)

Else Print(“”)

//creates a new product object with the user specified values, and update the information for the current user in the product database table with the new information}

●DeleteProduct(String ProductId)

{ if(ProductId)

Product.delete()

Product.save()

Return Print(“ ”)

Else Print(“”)

//use ProductId to find product and use remove method to delete then return the message }

●ReviewProduct(String ProductId)

{ if(ProductId)

Const review =(

Name=req.user.name

Rating =number(rating)

Comment=req.user.comment)

}

Product.review.push(review)

Product.save()

Print(“”)

Else Print(“”)

//ask user to input ProductId to find the Product and review or write comment, save the information and return the message}

Class: Checkout_Order:

Intent: To check the status of our orders

●AddOrder()

{if(length(order)==0)

print(“the shopping cart is empty”)

else const =new order(

order.seller= seller

order.price=price

order.id=id

……)}

Const ordersaver=order.save()

Print(“ ”)

//Add new id to the order table, which includes order items, price, seller, image, and quantity }

●DeleteOrder(String ID)

{ const order =order.find(ID)

If(Order){

Order.remove()

Order.save()

Print(“”) }

Else Print(“ ”) }

//use order id to find one order in the order table, then send a request to the service to delete the related order}

●UpdateOrder(String ID)

{ const order =order.find(ID)

If(Order)

Order.price=price

Order.size=size

Order.save()

Print(“ ”)

Else Print(“ ”)}

//use order id to find one order in the order table, then update the order with new order items, price, seller, image, quantity and total_price)

●Pay_Order(String ID)

{ const order =order.find(ID)

If(Order)

Check(Order)// check the status of Order

Order.readytopay=true;

Order.payAT=Date.now()

Order.paymentResuly=(change)}

Const updateOrder=order.save();

Send message(from:

To:

Subject:

Html:)

Print(“ ”)

Else Print(“ ”) }

//use order id to find one order in the order table, check the order is pay or not? Set the pay time and use if condition to ask user ready to play or not ? if pay successful, then remove the order from database and print the message. Else, it print pay failed or order doesn’t exist depend of the situation.}

●OrderDelivery(String ID)

{ const order =order.find(ID)

If(Order)

Check(Order)// check the status of Order

Order.isDelivered=true

Order.deliveredAt = Date.no();

Const updateOrded= order.save();

Print(“ ”+ updateorder)

Else Print(“ ”)

}

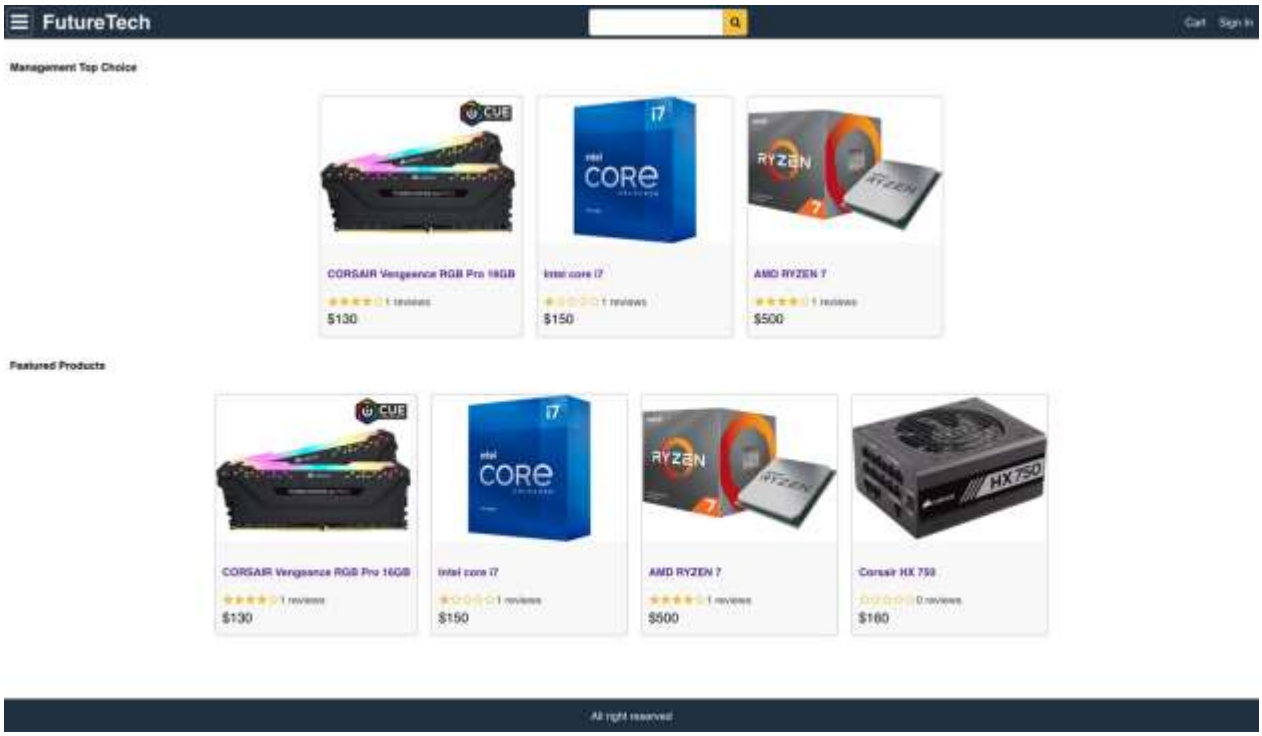
//use Order Id to find the order and check the status of order -Is Delivered? Save the information and return print the message }

● SendMessage(messageObject)

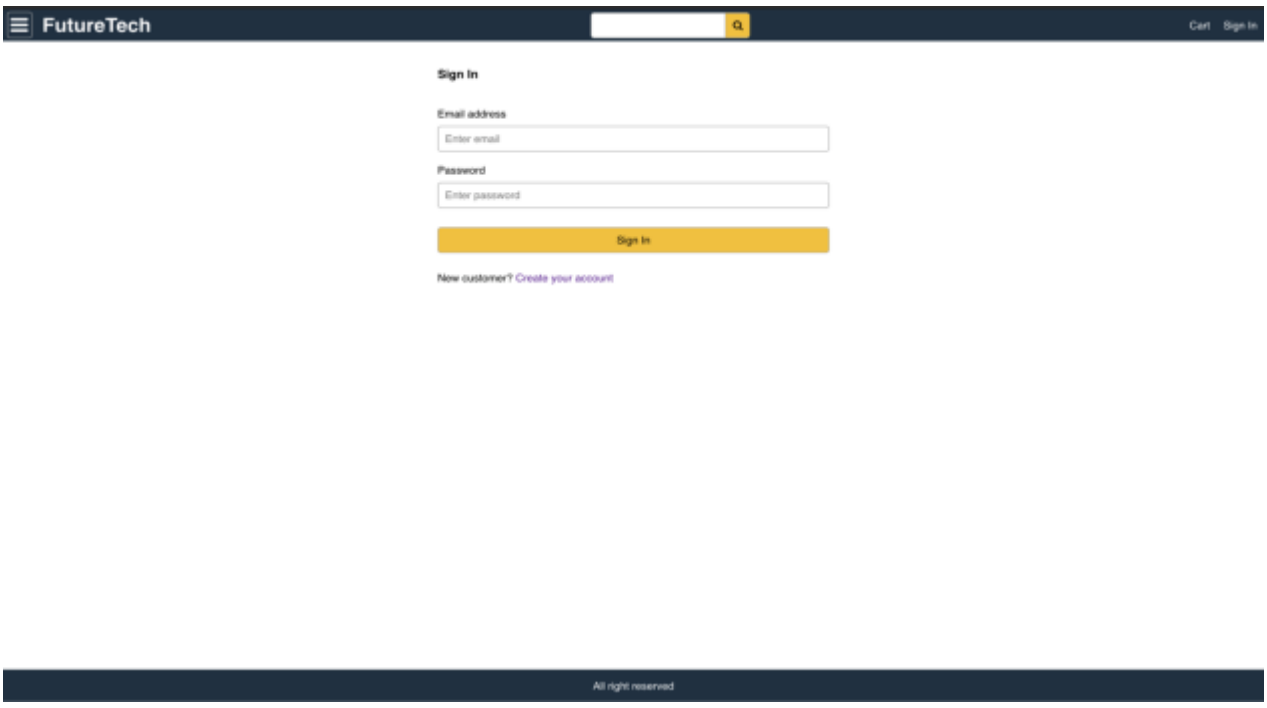
//will push to the database messages table the information contained in the messageObject

5. System Screens:

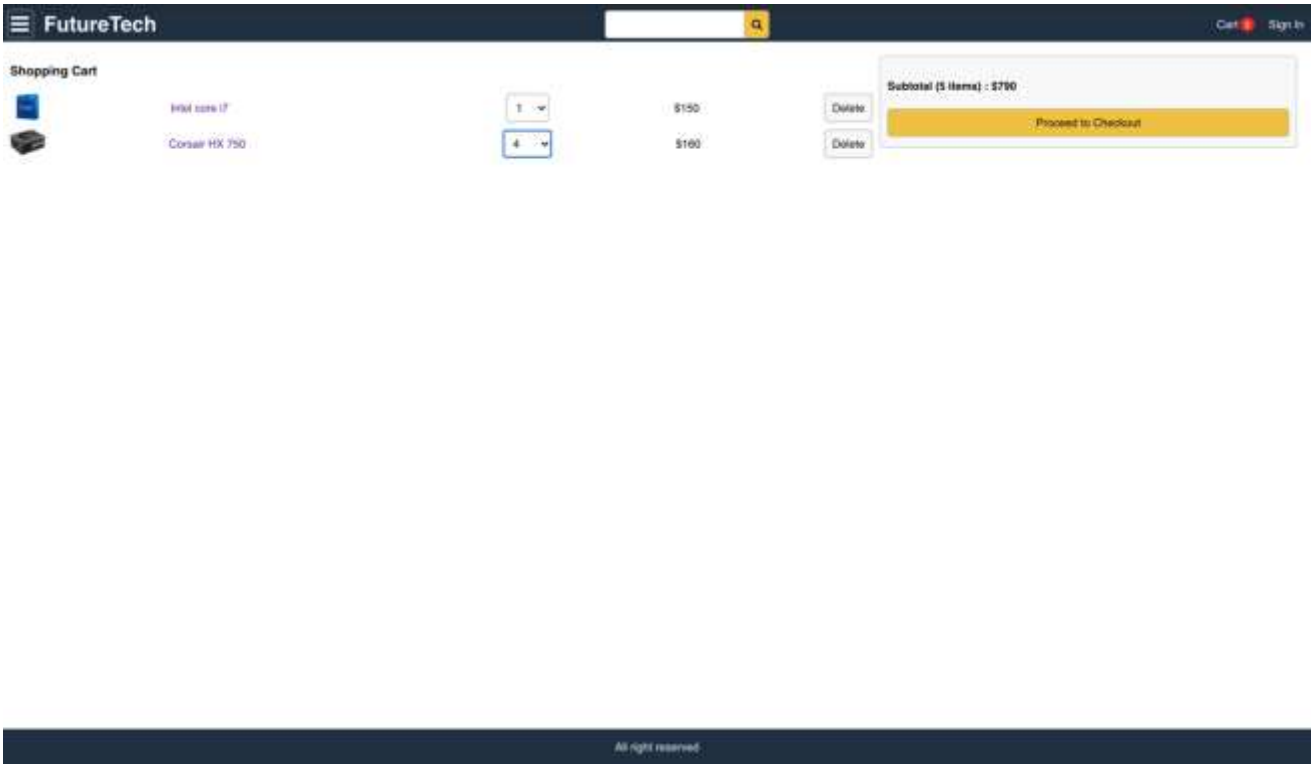
Home Screen with 3 suggested system by Manager.



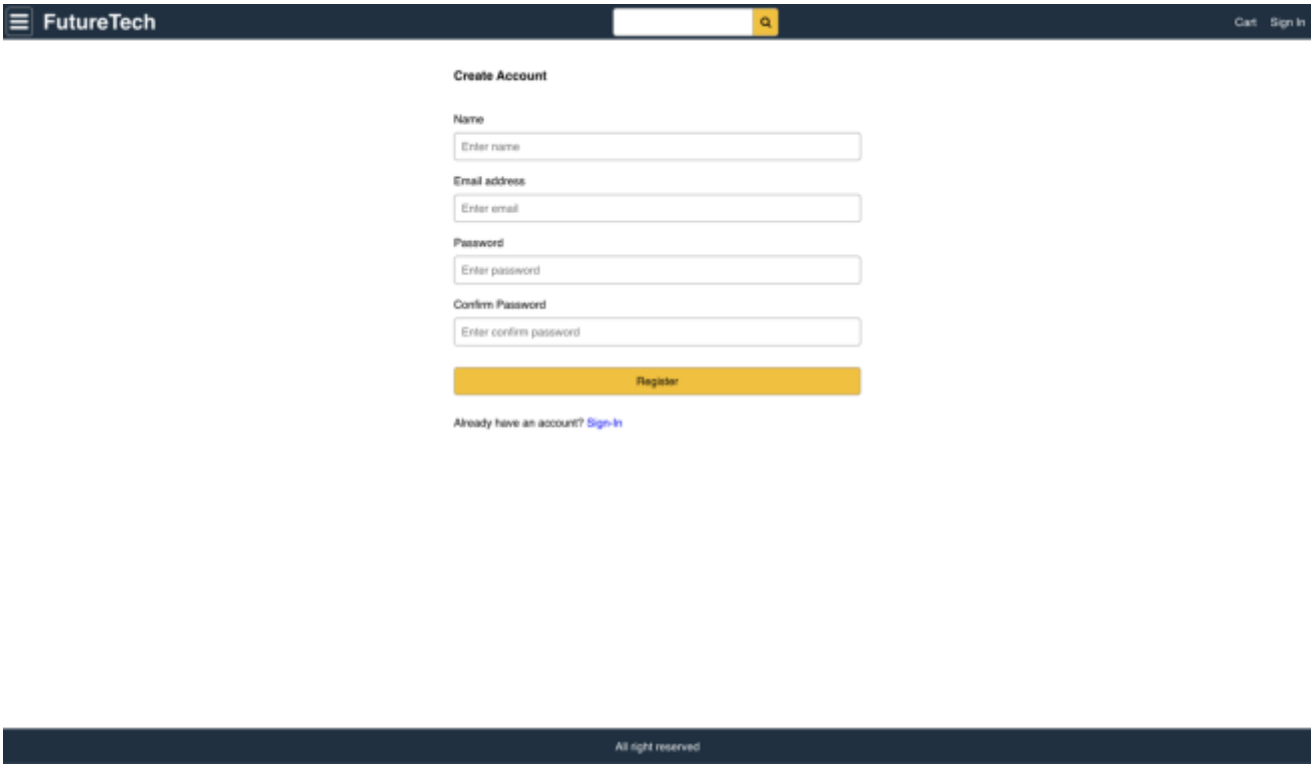
Sign in Page



Add to Cart Page



Sign up Page



6. Group Meetings:

Date: February 20, 2021

Participants: All group members

Agenda:

1. Introduce each other.
2. We decided to use ReactJS, HTML and CSS as our main structure for this project.
3. Come up with a main web page sketch.

Problem:

1. The name of our team/project remained undecided.

Date: March 28, 2021

Participants: All group members

Agenda:

1. Discussed what we need to do for the Phase I report.
2. Divided the work between each member.

Date: April 01, 2021

Participants: All group members

Agenda:

1. Come up with the Project and Team name.
2. Check up on everyone's status.
3. Help the member who is having problem with their parts.

Date: April 02, 2021

Participants: All group members

Agenda:

1. Put all the diagrams, and details on the report

Date: April 18, 2020

Participants: All group members

Agenda:

1. Created the skeleton of our project.
2. Connected to the DB.

Date: April 23, 2021

Participants: All group members

Agenda:

1. Worked on the Phase II report.
2. Divide the work between each member.

Date: April 25, 2021

Participants: All group members

Agenda:

1. Checked on the update on each member work.

Date: April 26, 2021

Participants: All group members

Agenda:

1. Put all the diagrams, code, and figures into the report.

7. Github:

GitHub repository link for our project: https://github.com/tenzin1308/CSc-322-Project