Study of an online shopping system

Submitted by

Pragathi Kancharla (1935733) Graduate Student

Sri Vidya Malladi (1925883) Graduate Student

Ujjwal N Shah (1794487) Undergraduate Student

Bharat Darbar (1973214) Graduate student

Computer Science and Management Information Systems

Gildart Haase School of Computer Sciences and Engineering

A **project report** paper submitted in partial fulfillment of the requirements for the course - **Software Engineering CSCI6620.81**

Under the supervision

of

Dr. Yongming Tang

Assistant Professor, Department of Computer Science



(Fall 2020)

You score 92/100. Team responsible for the errors. Excellent!

Contribution

**Everyone contributed for each part in the project in cross checking and correcting others parts.**

**Part A:**

* Problem Statement is done by Everyone.
* Restating the List of Activities is done by Everyone.

**Part B:**

* Analysis Model : Functional model is done by Sri Vidya and Ujjwal
* Dynamic model is done by Pragathi and Bharath.
* Analysis of object models is done by Everyone.

**Part C:**

* Including at least one design pattern for an entity object is done by Everyone.
* Interface specifications: an invariant for each entity class done by Ujjwal.
* Precondition and postcondition for the operations in each entity class done by Ujjwal and Sri Vidya.

**Part D:**

* Including subsystems and software architecture for the system done by everyone.
* Deployment diagram, access control (one of the three: a global access table, an access control list or a capability) done by Pragathi and Bharath
* Strategies for global control and at least one boundary use case done by Pragathi and Bharath.

PART- A

# Problem statement:

Ecommerce has evolved very rapidly from the past decade and it’s been used with a maximum number of people nowadays for all their work to be done. As the demand increased all of them are updating their businesses online. Developing an online site and making it user friendly is a must. So, by using the online developed application one can shop online anywhere and anytime. By comparing different sites for the same product and deciding. In this paper an online shopping like Amazon.com an online shopping system for customershas been taken and described the various functionalities it uses by attaching screenshots.

# what you can do with the system:

Amazon.com is an online shopping system where customers can buy and check about the various products which are there in the site and which have the technologies with the latest trends all over the world that can be viewed and ordered by one click. Moreover the process is reliable, flexible and is user friendly to shop.

One can sign up using an email id or phone number which will be unique for each customer. So, by providing those customers can create their own account to use it further. They can also provide the address they frequently use to deliver and also add the payment details if they don’t want to type them every time it’s their wish.

In the system, one can buy starting from basics to electronic goods, Hardware products, Food, Drinks, Hardware, Home Appliances, Sports, Kids product, Footwear, Jewelry and many more products can be found with a variety of options including with brand and the price limit they want.

There will be seasonal, and festival offers with discounts too which attracts a lot of customers to opt for the online shopping. If they do not want any product and does not satisfy the customer there is an option to cancel the order or return the product within some time limit. Moreover, the tracking system which they provide is the best way to know where their product is present.

So, below are the things that a customer can do on the amazon.com:

* Can sign up and create an account. They can even change the password.
* Can provide address and payment details.
* Customers can search for anything they want to on the search options or by shop by department option.
* Many options of the same product will appear. Customers can choose based on like price, picture, brand, view, rating, time of delivery and so on.
* If we like any option, we can add that item in the cart and search for more options.
* If we decide on the product, we can go to the view cart and delete or keep it for future use or can check out for that product to buy.
* There the address to which the product can be typed or if already saved will deliver to that address.
* Payment options can be provided before or whenever you shop. For some products there will be an option for cash on delivery too.
* After the order, the customer can check my orders and track his product.
* If he decides to cancel the order can cancel too or replace the item within the specified time.
* If after the product is delivered you are not satisfied with the product and can return the product.
* If the customer is satisfied or not satisfied with his product can review the product by giving a rating. So that will be useful for the other users.

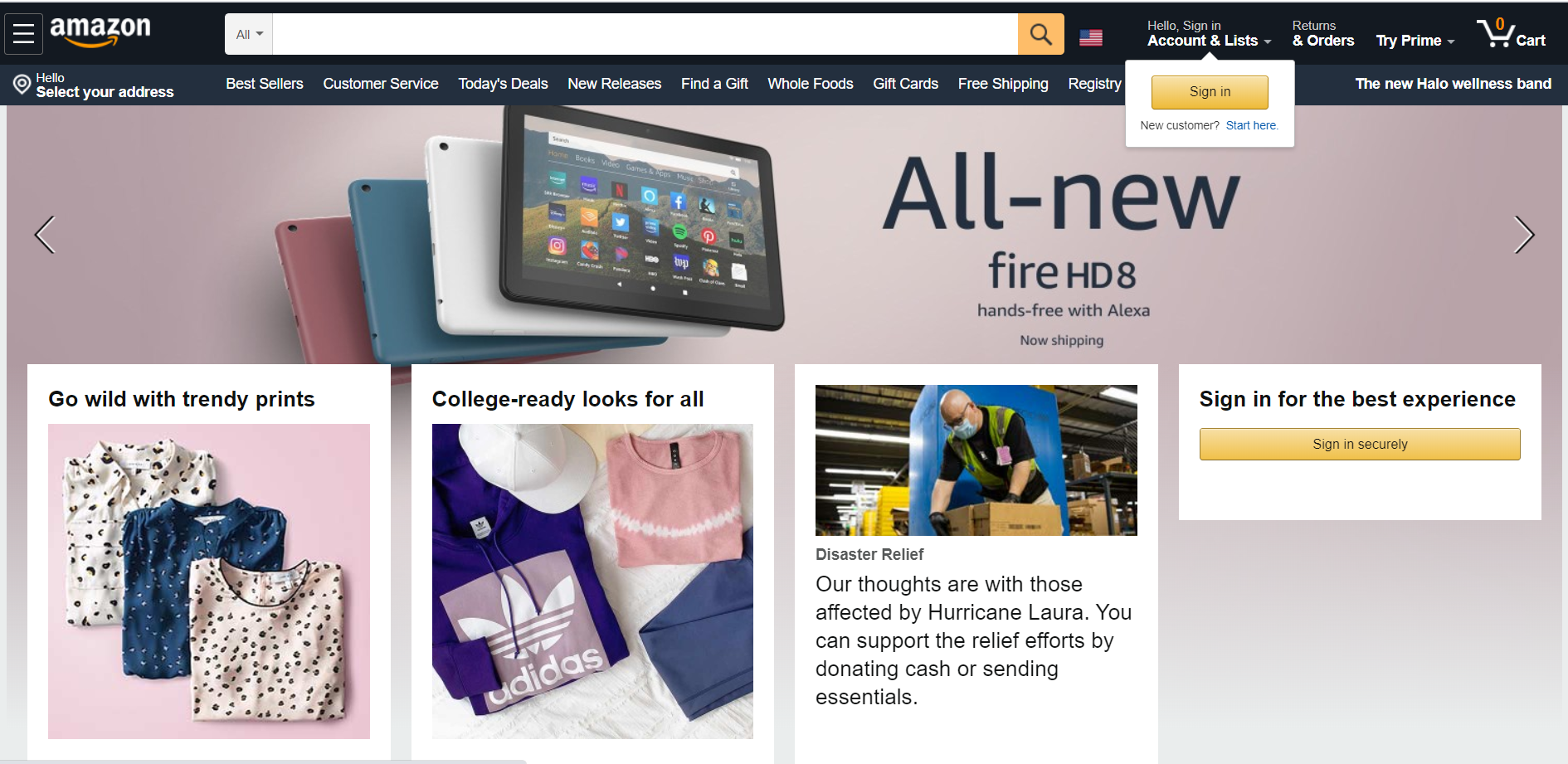
Once the order is given the salesperson in Amazon will do the following:

* They receive the order given by the customer.
* Send those order details to the courier company.
* Send the confirmation email, tracking id, order number to the customer.
* After the delivery they can view the reviews given by the customers and about the services provided by the delivery. To maintain and improve the services.

# List of functionalities with screenshot:

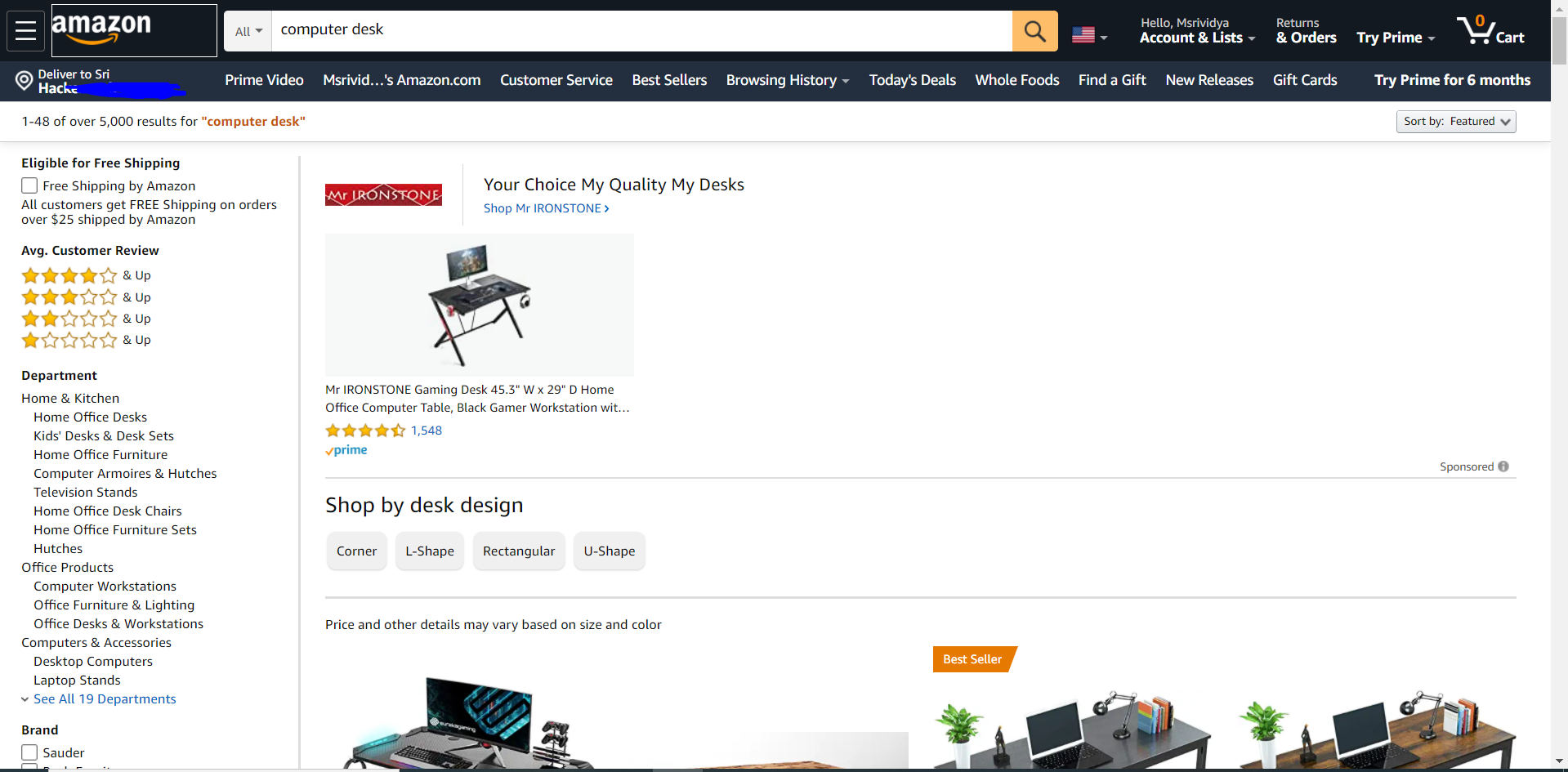
## Home page:

## Below is the home page of the site one can search here or can sign with their details.



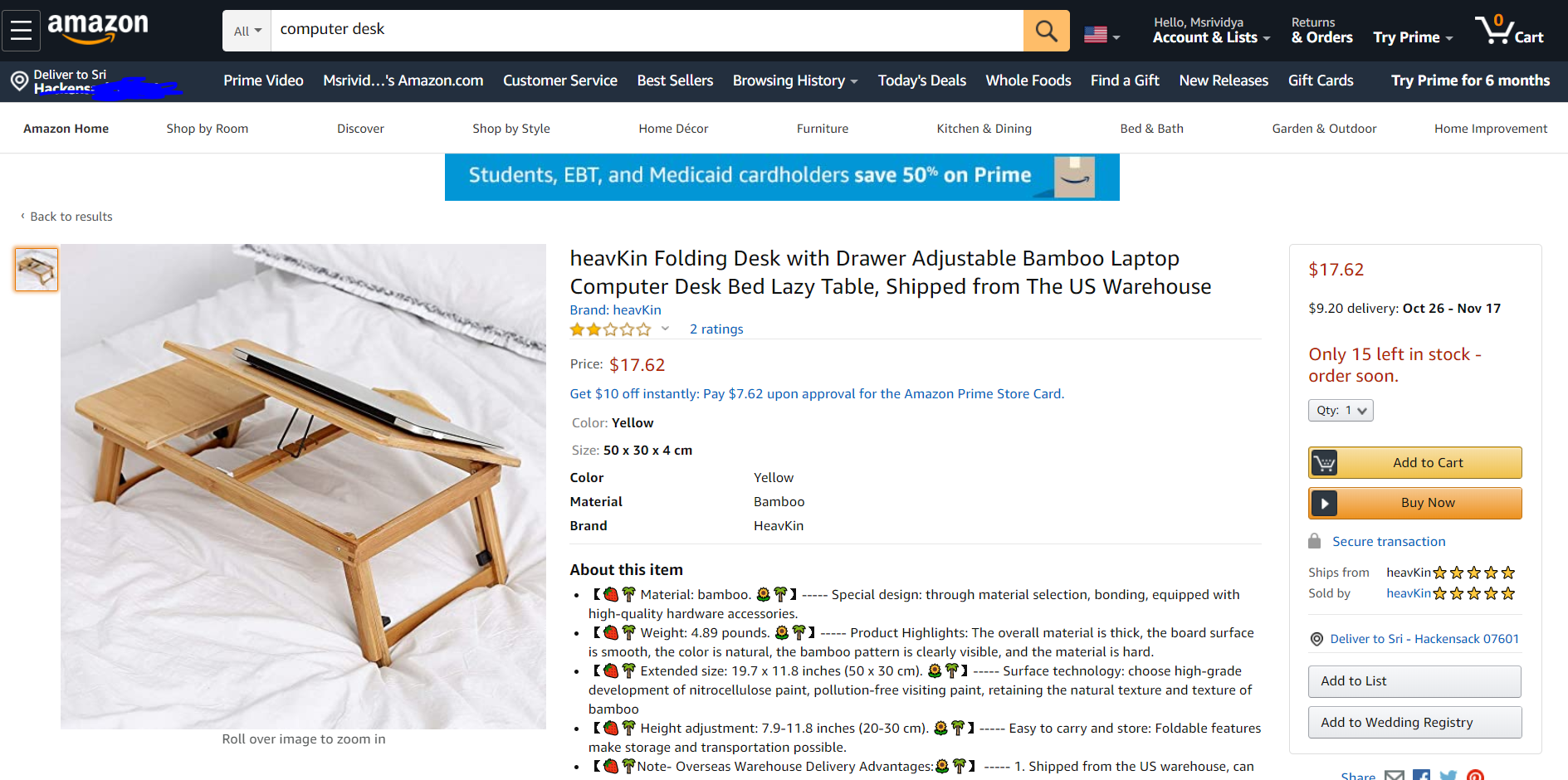
## Search Product:

## Customers can search for any product they need in the search bar as shown in the below screenshot. The search bar also gives automatic options when you type few letters.



## View Product:

## After we search for the product, we want one to view the product full details with the picture, price description, specifications, uses, models, colors, rating, size and so on.

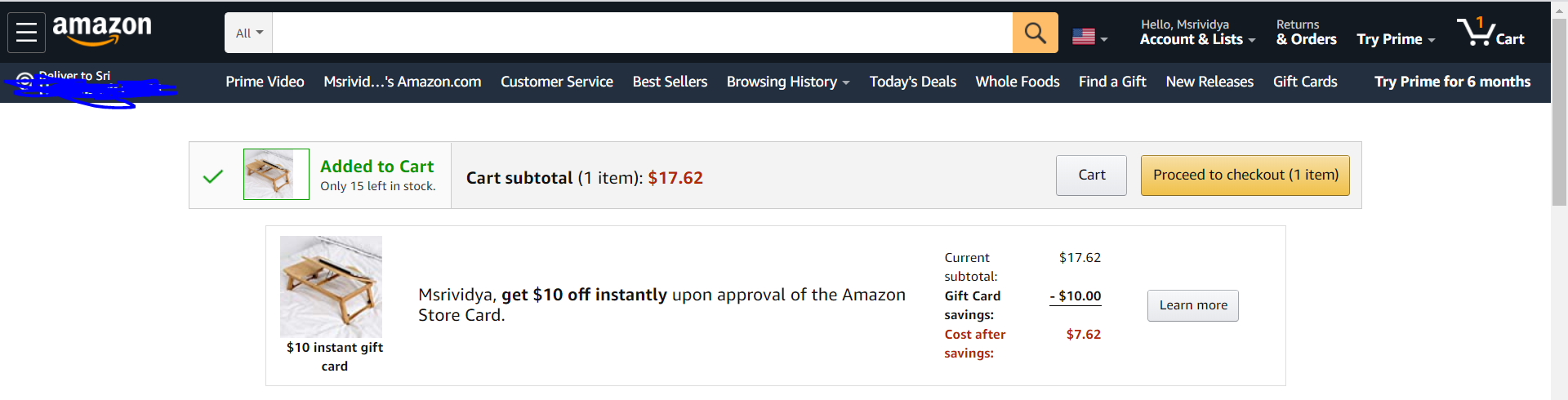


## Add to Cart:

## If we like any product, we can add the product to the cart as below and can delete or proceed to check out. Customers can continue to look for the products after adding the items to the cart and look for those later.

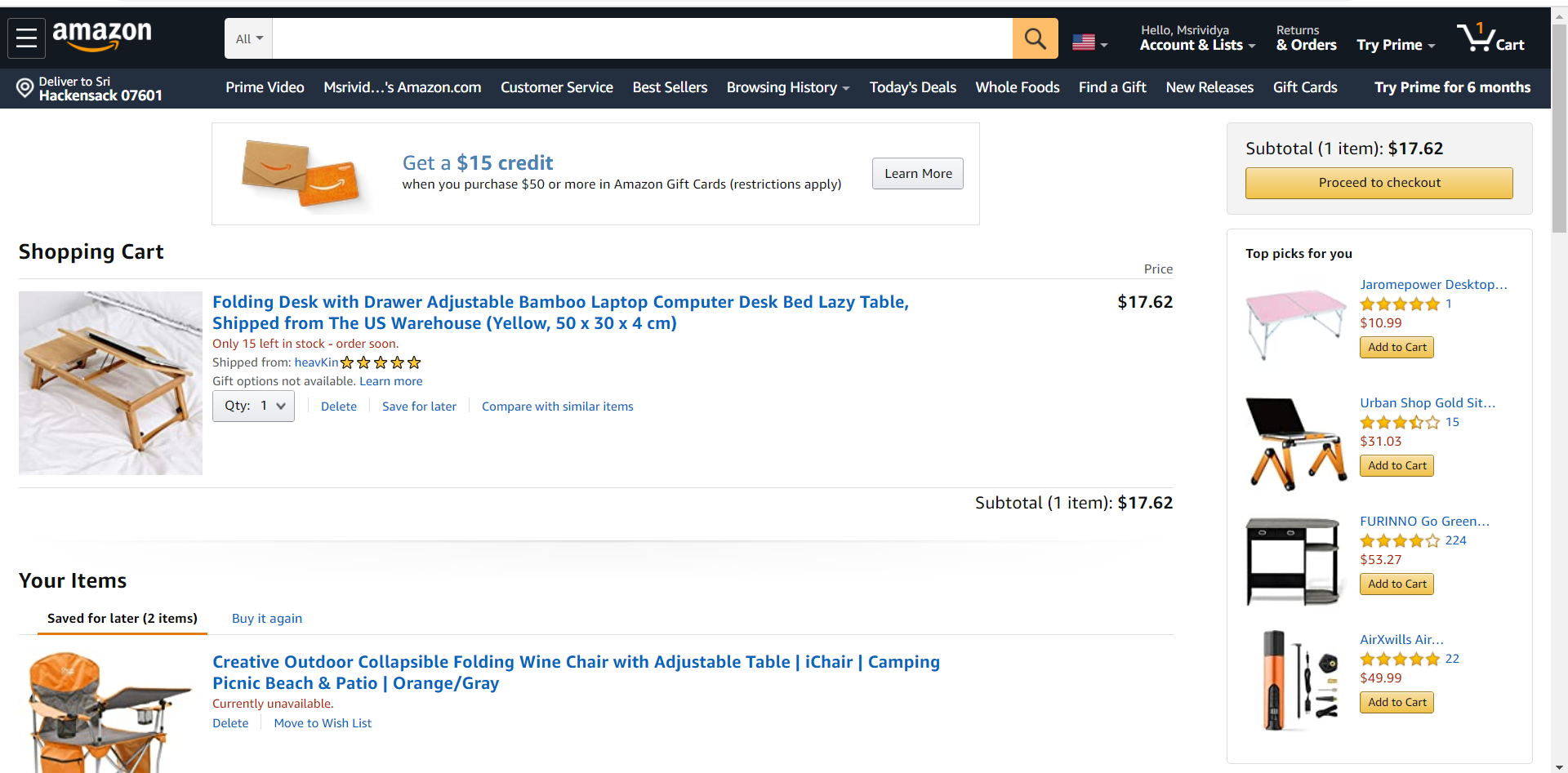
## Buy Now:

## If we like a product and decide to buy there is an option to directly buy the product without adding it to the add cart.



## View Cart:

## If we have added many products, we can view cart and remove or can proceed to pay for both at a time as shown in below.

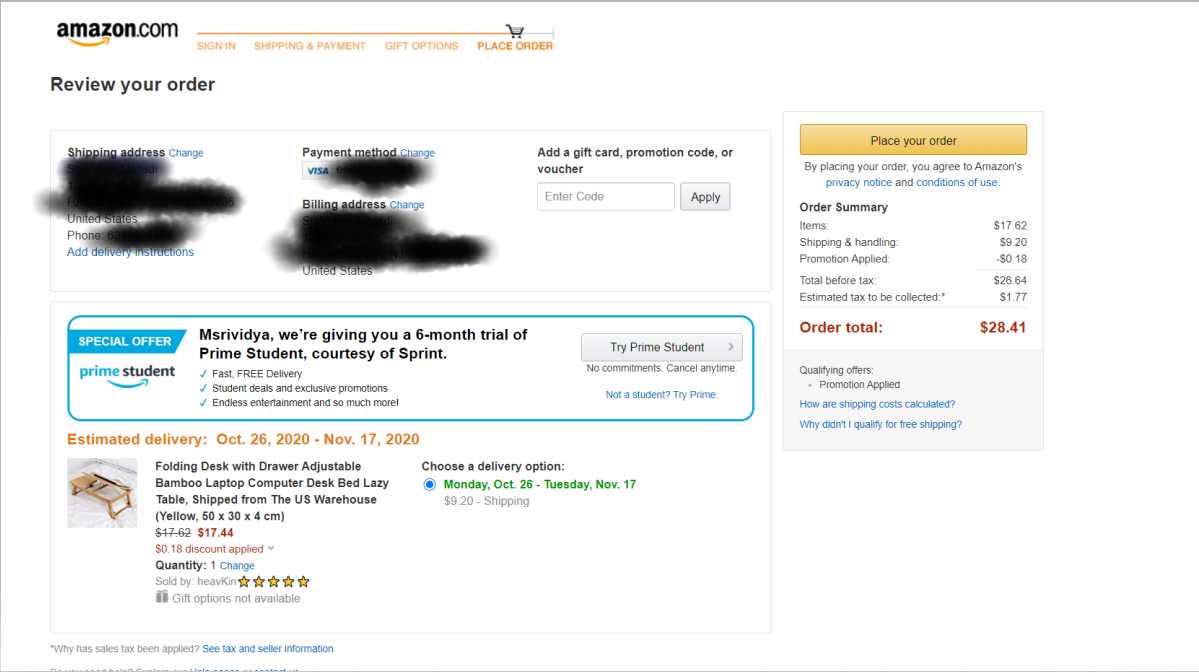


1. Update/ Delete Shopping Cart:

If a Customer wants to add something or increase the number of items of the same product or make any changes to the existing cart. They can update the cart and see the changes, or they can also delete an item from the cart by deleting any product in the shopping cart.

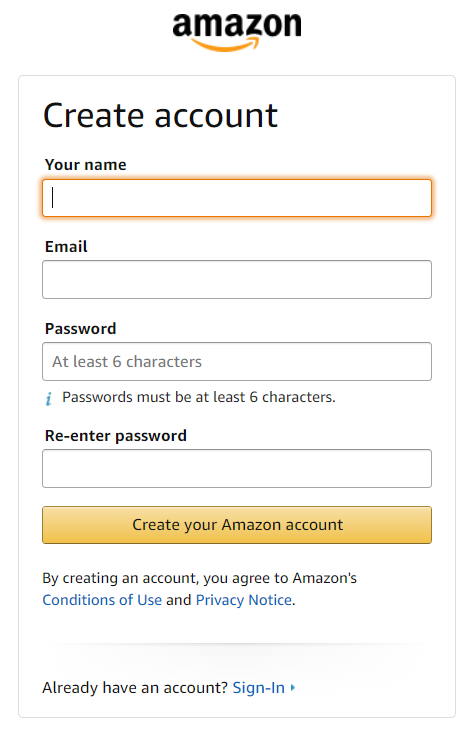
## Check Out:

## Once we have checked out if we already have the payment details it will directly take to this page or we can add the payment details and buy the product. We need to select the shipping address or default so we can keep ours. We can place the delivery options here.



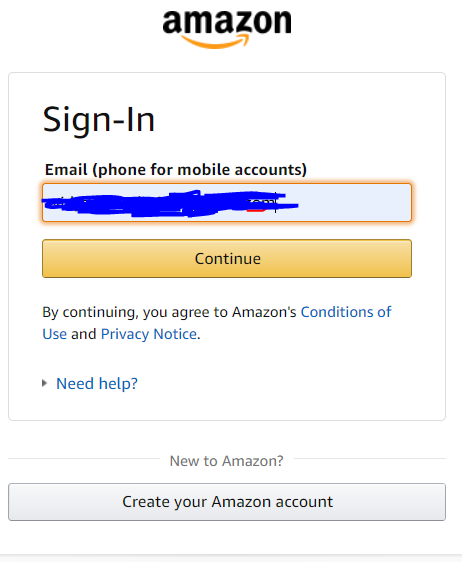
## Sign Up:

## If we already have an account, we can do all the above procedures. If we do not have an account after the procedure to checkout it will redirect to sign-in option. Here they can create a new account for themselves.

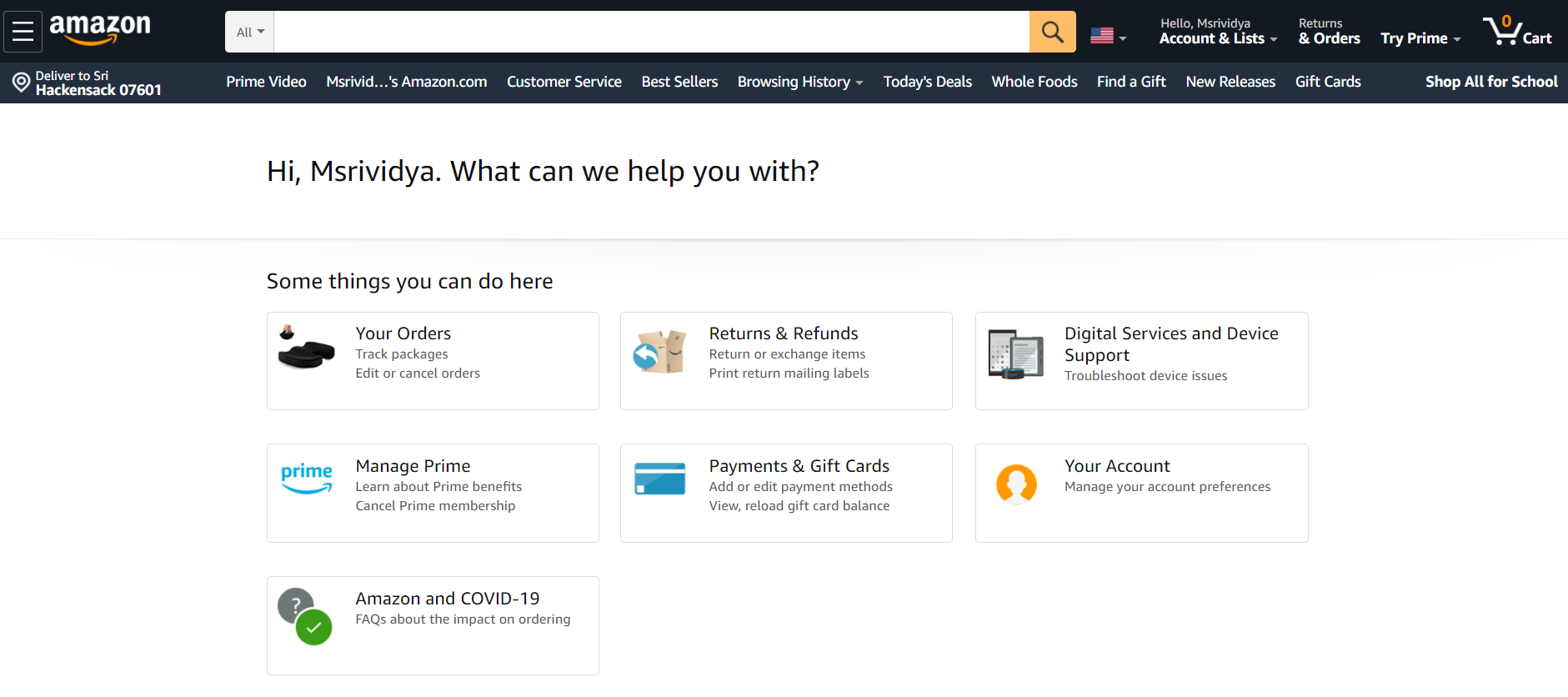


## Login:

## If a customer already has an account, they can directly login before searching for a product by using their username and password. If they forget their password, they get the OTP for the email id or phone number provided while creating the account.

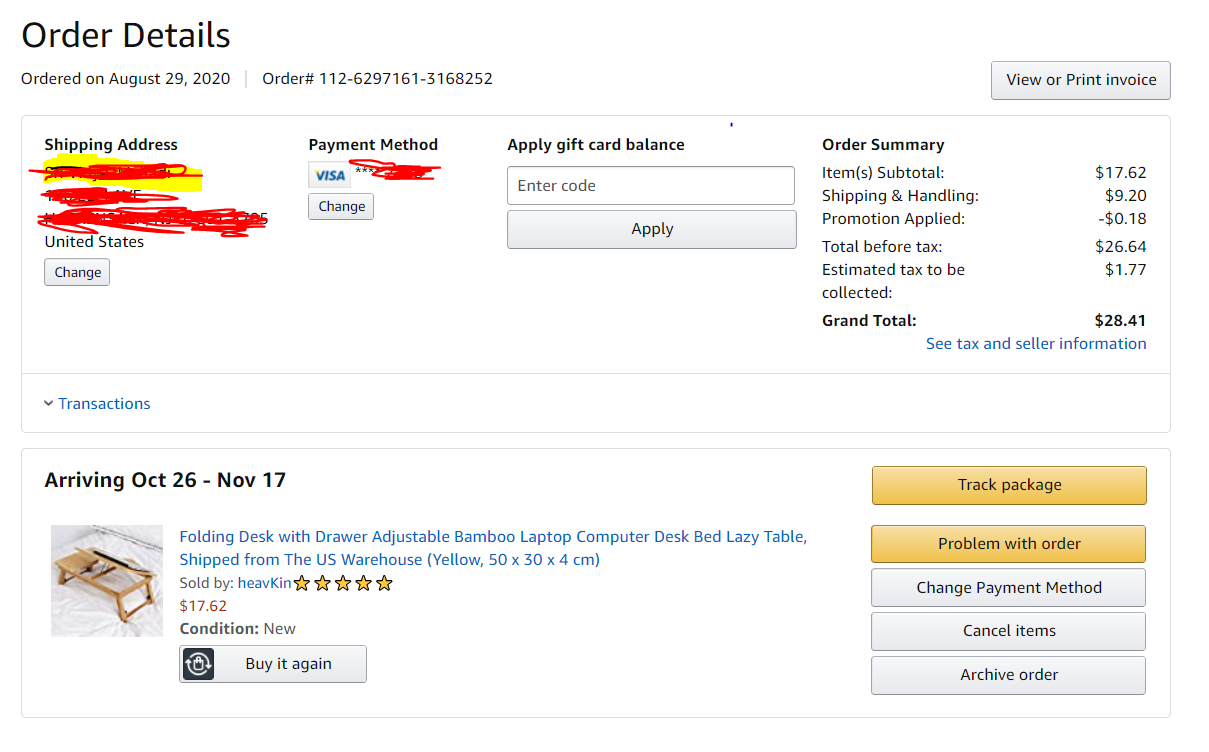


This is the home page for your account



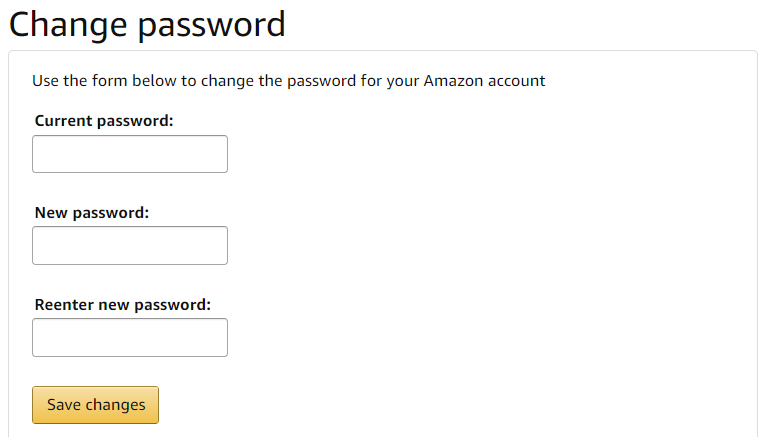
## Order details:

## Here customers can check their orders or change anything if needed.



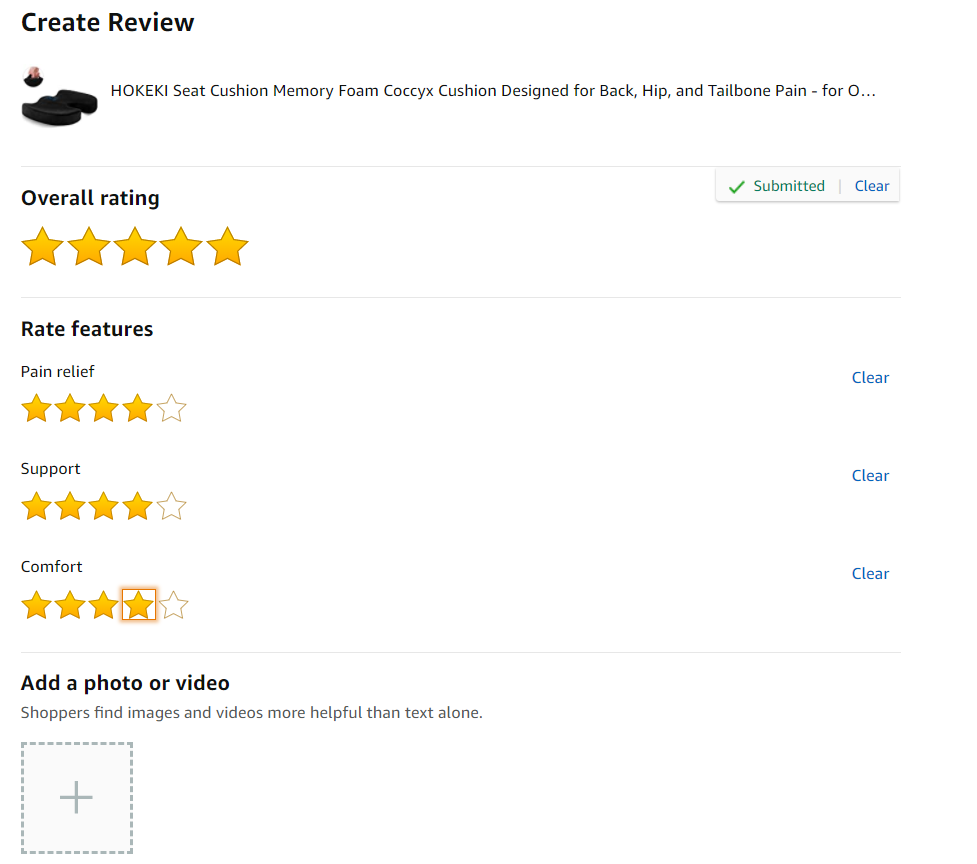
## Change Password:

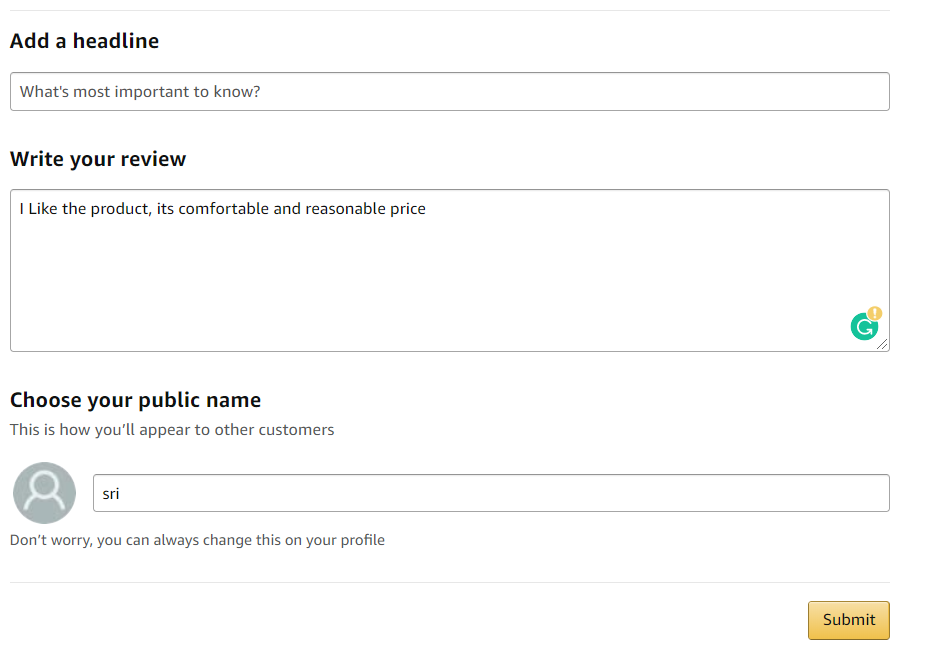
## If a customer wants to change his username or password, they can change in the login and security page in their account.

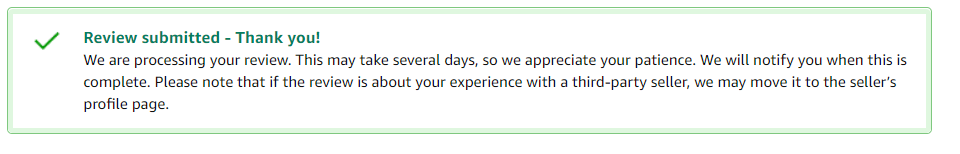


## Review/Rate Product:

After the product is delivered, customers can give their opinion about the product if they like or if they do not like it. This helps the customers and also amazon people to improve their services.

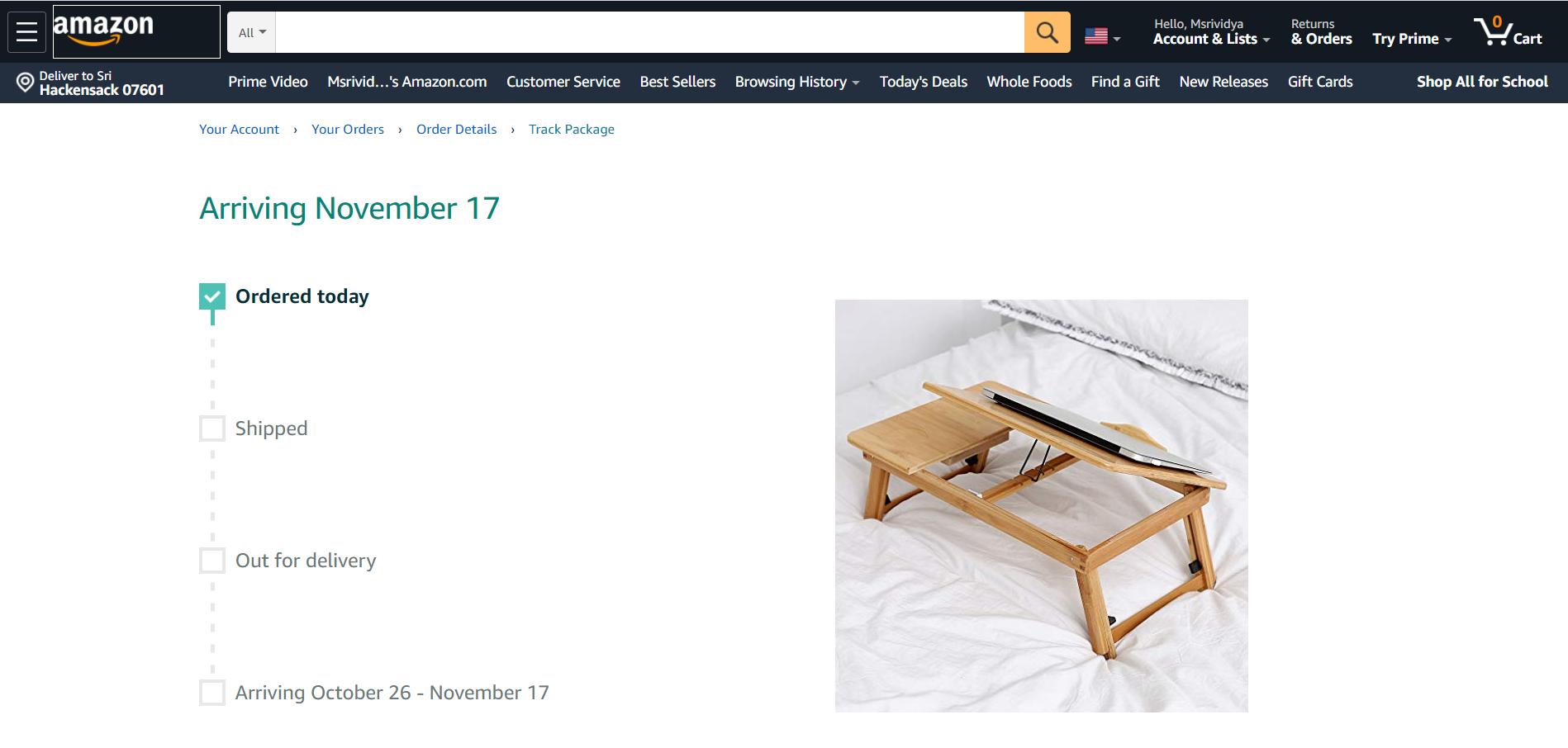






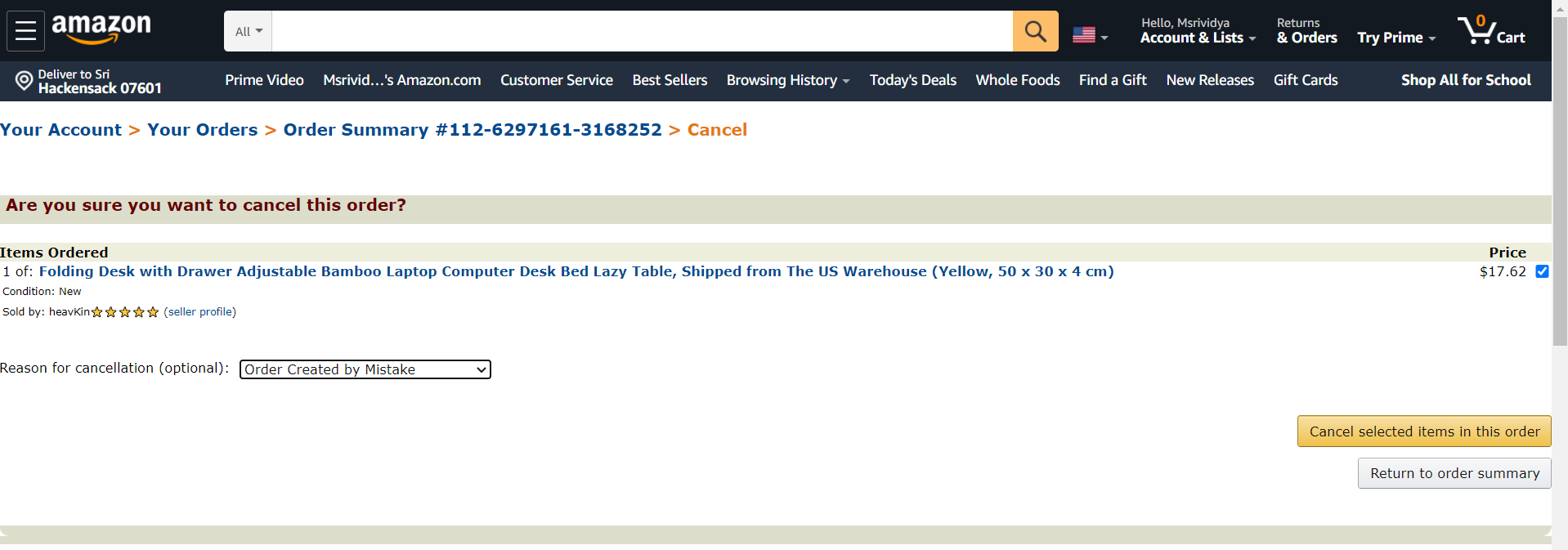
## Track Order:

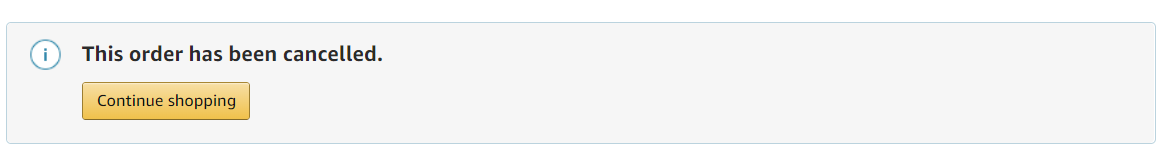
Customers can also track their products after they order by going to the ordered page. There they can see where their product is and if it got shipped, get out for the delivery and so on.



## Modify/Cancel Order:

Customers can modify or cancel the order if they do not like the product, they ordered in the ordered details page. They can modify or cancel the order until the item is shipped after it got shipped, they can return or replace the product if they do not like it.





Part-B

Analysis Model

# Customers are able to search for a product.

## **Functional model:**

**Scenarios:**

Functionality: Customer search for a product by entering keywords in the search bar or through categories of products.

Functionality name: SearchProduct

Example 1: Vidya attempts to search for iPhone by entering **iPhone** in the search bar by clicking on the search icon or by pressing enter.

|  |  |
| --- | --- |
| Scenario name | Found:SearchProduct |
| Participating Actor instances | Vidya:Customer |
| Flow of Events | 1. Vidya enters the keyword “iPhone” in the search bar and press enter or the search icon.  2. Different iPhone models are shown.  Apple iPhone XS(64GB)- $1099.99  Apple iPhone 8(128GB)- $ 849.99  Apple iPhone 7 plus(128GB)- $500.99  3. iPhone 7 plus with grey color is chosen by Vidya. |

Example 2: Now Vidya attempts to search for iPhone by entering **eyephne** in the search bar and pressed enter.

|  |  |
| --- | --- |
| Scenario name | NotFound:SearchProduct |
| Participating Actor instances | Vidya:Customer |
| Flow of Events | 1. Vidya enters eyephne in the search bar and press enter or the search icon.  2. “No results for eyephne” is displayed to Vidya |

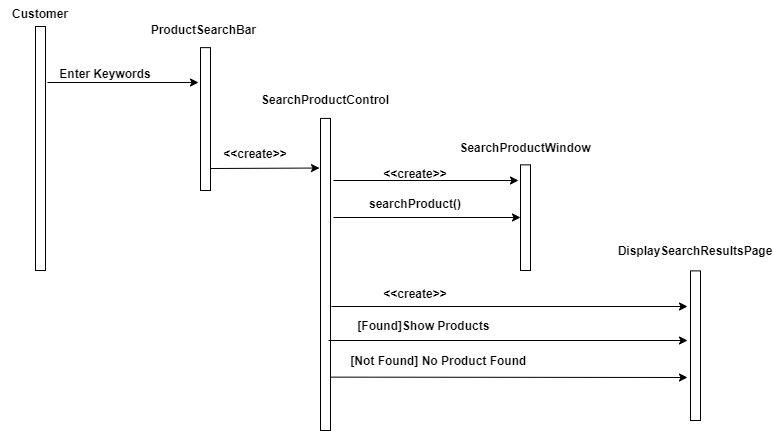
**Initial use cases:**

|  |  |
| --- | --- |
| Use case name | SearchProduct |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customers search for a product by entering the key word in the search bar and press enter or the search icon.    2.List of results or “No results for xx” is displayed to the customer. |
| Entry conditions | Amazon web page has been loaded |
| Exit conditions | Search results page is displayed to the Customer |
| Quality Requirements | Results must be displayed in less than 3 seconds |

**The most refined final use cases:**

|  |  |
| --- | --- |
| Use case name | SearchProduct |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1.Customer activates the Search Product by entering the keyword and clicking on the search icon in the HomePage.    2. SearchProductControl object is created.    3.SearchProductControl gets the keyword and creates Product object and invokes search product operation. //Here supposed to be an entity object, not a boundary object.    4.SearchProductControl receives the list of products and displays the results on DisplaySearchResultsPage else No results found message is displayed. |
| Entry conditions | Amazon web page has been loaded |
| Exit conditions | Search results page is displayed to the user |
| Quality Requirements | Results must be displayed in less than 3 seconds |

## **Dynamic model:**

**Sequence diagram://SearchProductWindow is Product.**

# Customers are able to view a product.

## **Functional model:**

**Scenarios:**

Functionality: Customer view for a product by entering keywords in the search bar or through categories of products.

Functionality name: ViewProduct

Example 1: Pragathi attempts to View for iPhone 7 plus by entering key word in the search bar. View Product displays complete details of the product.

|  |  |
| --- | --- |
| Scenario name | ViewProductDetails:ViewProduct |
| Participating Actor instances | Pragathi:Customer |
| Flow of Events | 1. Pragathi clicked on “Apple iPhone 7 plus(128GB)” in the product page and pressed enter.  2.Complete details of iPhone 7 plus are shown by system as below.  Name: Apple iPhone 7 plus(128GB)  Price: $500.99  Color: Silver  Display: 5.5 inches |

Example 2: View Product displays no details found for the product searched.

|  |  |
| --- | --- |
| Scenario name | NoDetailsFound:ViewProduct |
| Participating Actor instances | Pragathi:Customer |
| Flow of Events | 1. Pragathi clicked on “Apple iPhone 7 plus(128GB)” in the product page and pressed enter.  2.Complete details of iPhone 7 plus are shown by system as below.  Name: Apple iPhone 7 plus(128GB)  Price: $500.99  Color: Silver  Display: 5.5 inches |

**Initial use cases:**

|  |  |
| --- | --- |
| Use case name | ViewProduct |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customers click the “View Product” in the search bar and press enter or the search icon.  2.List of results will be displayed or no details found is displayed. |
| Entry conditions | Amazon web page has been loaded, product is in stock |
| Exit conditions | Product has been successfully viewed or not |
| Quality Requirements | Results must be displayed in less than 3 seconds |

**The most refined final use cases:**

|  |  |
| --- | --- |
| Use case name | ViewProduct |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. A Customer clicks the “View Product” shown in the products page.  2. The ViewProductControl is created by the ViewProductBO.  3.Control object allows the customer to enter product details in ViewProductWindow.//Entering here is not consistent with your initial use case and scenarios.  5. The ViewProductControl gets the details of the product and creates a Product object and invokes the retrieve operation.//Object here is Product.  6. The ViewProductControl receives the details of the products like quantity, size, color etc.  7. The ViewProductControl object displays complete details of a product,ShowProductDetailsWindow. |
| Entry conditions | Amazon web page has been loaded, product is in stock |
| Exit conditions | Product has been successfully viewed or not |
| Quality Requirements | Results must be displayed in less than 3 seconds |

## **Dynamic model:**

**Sequence diagram://For an entity object, use the name from the application.**

# Customers are able to add a product to their shopping carts.

## **Functional model:**

**Scenarios:**

Functionality: Customers can add a product by viewing or pressing the product they like in the search bar or through categories of products.

Functionality name: AddToCart

Example 1: Ujjwal attempts to add a product like iPhone 7 plus by clicking Add to Cart. Product added to cart.

|  |  |
| --- | --- |
| Scenario name | ProductAdded:AddToCart |
| Participating Actor instances | Ujjwal:Customer |
| Flow of Events | 1. Ujjwal searched for “Apple iPhone 7 plus(128GB)” in the view product page.  2. Ujjwal clicks on the “Add to Cart” button.  3. iPhone 7 plus(128GB) is successfully added to the cart. |

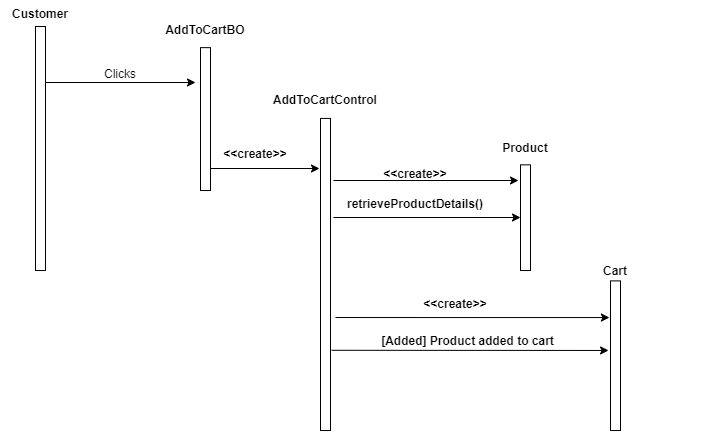
**Initial use cases:**

|  |  |
| --- | --- |
| Use case name | AddToCart |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customers click the “Add To Cart” button from the view product details page he wants to add.  //Not necessary to repeat Click3. Product is added to cart. |
| Entry conditions | Amazon product page has been loaded, product is in stock |
| Exit conditions | Add to cart item is added |
| Quality Requirements | Results must be displayed in less than 3 seconds |

**The most refined final use cases:**

|  |  |
| --- | --- |
| Use case name | AddToCart |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. A Customer clicks the “Add to Cart” shown in the product page.  2. The AddToCartControl is created by the AddToCarttBO.  3. The AddToCartControl gets the product information like size, colour, quality and dimensions from the entity Product.  4.Then AddToCartControl creates a cShoppingCart object and initiates the add product functionality.  5. The AddToCartControl object adds the product to cart. //Missed 6 confirmation |
| Entry conditions | Amazon web page has been loaded, product is in stock |
| Exit conditions | Add to cart item is added |
| Quality Requirements | Results must be displayed in less than 3 seconds |

## **Dynamic model:**

**Sequence diagram:**

# Customers are able to view their shopping carts.

## **Functional model:**

**Scenarios:**

Functionality: Customers can view a product after adding it into the add to cart by viewing or pressing the cart.

Functionality name: ViewShoppingCart

Example 1: Bharat attempts to view a product he added in add to cart like iPhone 7 plus by clicking Cart. Shopping cart displays all the products available in cart

|  |  |
| --- | --- |
| Scenario name | ProductsAvailable:ViewShoppingCart |
| Participating Actor instances | Bharat:Customer |
| Flow of Events | 1. Bharat clicked on the “Cart” icon on the amazon web page.  2. System displays all the items which Bharath has added to cart if they are available in cart as below.  Apple iPhone XS(64GB)- $1099.99  Apple iPhone 8(128GB)- $ 849.99  Apple iPhone 7 plus(128GB)- $500.99 |

Example 2: Bharat attempts to view a product he added in add to cart like iPhone 7 plus by clicking Cart. Shopping cart displays an empty cart or error display.

|  |  |
| --- | --- |
| Scenario name | ProductsNotAvailable:ViewShoppingCart |
| Participating Actor instances | Bharat:Customer |
| Flow of Events | 1. Bharat clicked on the “Cart” icon on the amazon web page.  2. System displays that “ Your shopping cart is empty”. |

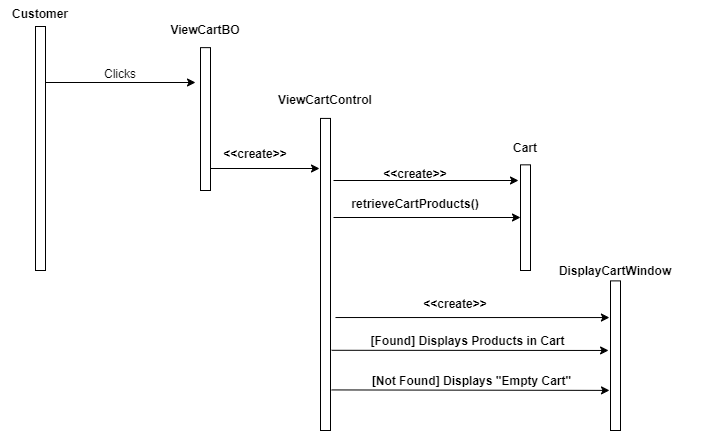
**Initial use cases:**

|  |  |
| --- | --- |
| Use case name | ViewShoppingCart |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customers click the “View Cart” shown in the homepage.  2. Product is viewed in the cart, empty cart is shown. |
| Entry conditions | Amazon web page has been loaded. |
| Exit conditions | Customers can see the list of all products in the shopping cart, empty cart. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

**The most refined final use cases:**

|  |  |
| --- | --- |
| Use case name | ViewShoppingCart |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customer clicks on the “View Cart” shown in HomePage.  2. An object of ViewCartControl is created by the HomePage .  3.The ViewCartControl gets the details of the product available in cart and creates a Cart object and invokes the retrieve operation.  4. The ViewCartControl receives the details of all the products like size, color, quantity, etc.  5. The ViewCartControl object displays all the products available in cart or displays your cart is empty in the DisplayCartWindow. |
| Entry conditions | Amazon web page has been loaded. |
| Exit conditions | Customers can see the list of all products in the shopping cart or an empty cart. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

## **Dynamic model:**

**Sequence diagram:**

# Customers are able to update their shopping carts.

## **Functional model:**

**Scenarios:**

Functionality: Customers can update their shopping cart by adding or removing a product after adding it into the add to cart.

Functionality name: UpdateShoppingCart

Example 1: Vidya attempts to update the shopping cart by adding one more iPhone 7 plus into the cart by clicking add product.

|  |  |
| --- | --- |
| Scenario name | AddProduct:UpdateShoppingCart |
| Participating Actor instances | Vidya:Customer |
| Flow of Events | 1. Vidya increases the quantity of the iPhone 7 plus in the cart by clicking on the add button.  2. iPhone 7 plus is successfully added to the cart and the quantity of the product increased to two. |

Example 2: Now, Vidya attempts to update the shopping cart by deleting the iPhone 7 plus added into the cart by clicking delete product.

|  |  |
| --- | --- |
| Scenario name | DeleteProduct:UpdateShoppingCart |
| Participating Actor instances | Vidya:Customer |
| Flow of Events | 1. Vidya clicks on the delete option to remove the iPhone 7 plus product from the cart.  2. The system successfully deletes iPhone 7 plus is from the cart. |

**Initial use cases:**

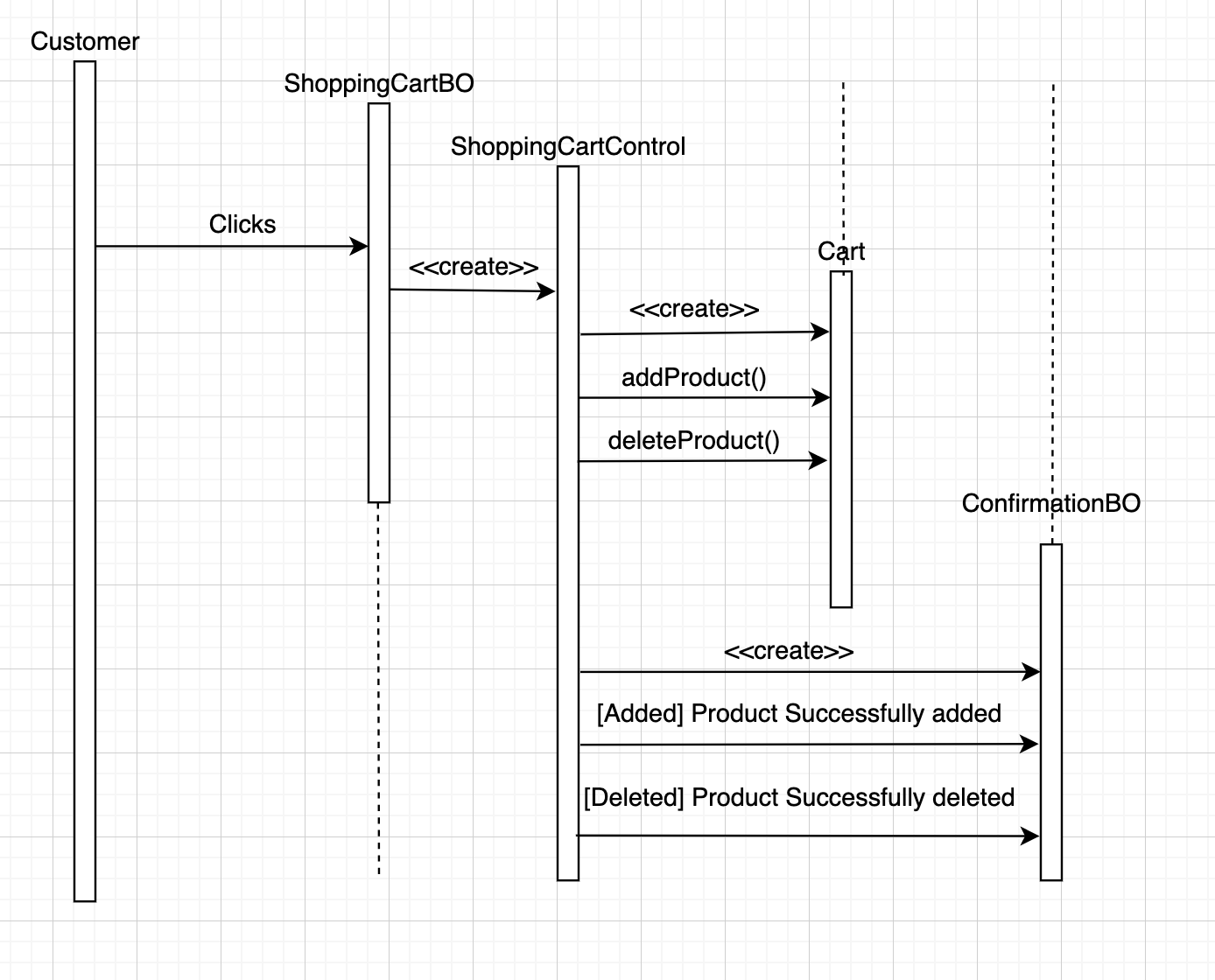
|  |  |
| --- | --- |
| Use case name | UpdateShoppingCart |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customer activates the “Add To Cart” shown in the View cart homepage.//Not consistent with your scenarios. UpdateCart means at the Cart page. Not necessary to do “Add To Cart”  2. Product is successfully updated, either the product is added or deleted. |
| Entry conditions | Amazon web page has been loaded. |
| Exit conditions | Customers can see the list of all products in the shopping cart, empty cart. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

**The most refined final use cases:**

|  |  |
| --- | --- |
| Use case name | UpdateShoppingCart |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. A Customer activates the “Add to cart” shown in ProductsDetailsPage.  2. The ShoppingCartControl object is created by the ShoppingCartBO.  3.The Control object allows the customer to click on the “Add to Cart” button in AddToCartpage. Customer clicks on the Add to cart button.  4. The ShoppingCartControl gets the product information and creates a cart object and initiates the add / delete product functionality.  5. The ShoppingCartControl creates a ConfirmationBO which displays “product successfully deleted” if the product has been removed or “product successfully added” if products have been successfully added to the cart. |
| Entry conditions | Amazon web page has been loaded. At least one product in the cart has to be there. |
| Exit conditions | Customer has updated the shopping cart. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

## **Dynamic model:**

**Sequence diagram:**



# Customers are able to checkout from their shopping carts to buy the products they want. Once checkout is successful, an order summary is displayed to the customers and also is emailed to the customers.

## **Functional model:**

**Scenarios:**

Functionality: Customer wants to checkout for the product he/she wants to buy directly from the products page or from the carts page.

Functionality name: CheckOut

Example 1: Pragathi attempts to CheckOut for iPhone 7 plus product she added in the add to cart.

|  |  |
| --- | --- |
| Scenario name | SuccessfullyCheckedOut:CheckOut |
| Participating Actor instances | Pragathi:Customer |
| Flow of Events | 1. Pragathi activates the checkout function of the system by clicking the button to proceed to checkout.  2. Pragathi enters the shipping details of the product.//Missed entering payment details.  3. The system shows product name: iPhone 7 plus(128GB)  Order number: 12345  Price: $500.99  Color: Silver  Display: 5.5 inches  Shipping address: 106 Elm Ave, Hackensack, NJ, 07601  Delivery time: 6 days |

Example 2: Pragathi attempts to CheckOut for iPhone 7 plus product she added in the add to cart. But the system shows the product can not be delivered.

|  |  |
| --- | --- |
| Scenario name | FailureCheckedOut:CheckOut |
| Participating Actor instances | Pragathi:Customer |
| Flow of Events | 1. Pragathi activates the checkout function of the system by clicking the button to proceed to checkout.  2. Pragathi enters the shipping details of the product.  3. The system shows iPhone 7 Plus(128GB) cannot be delivered. |

**Initial use cases:**

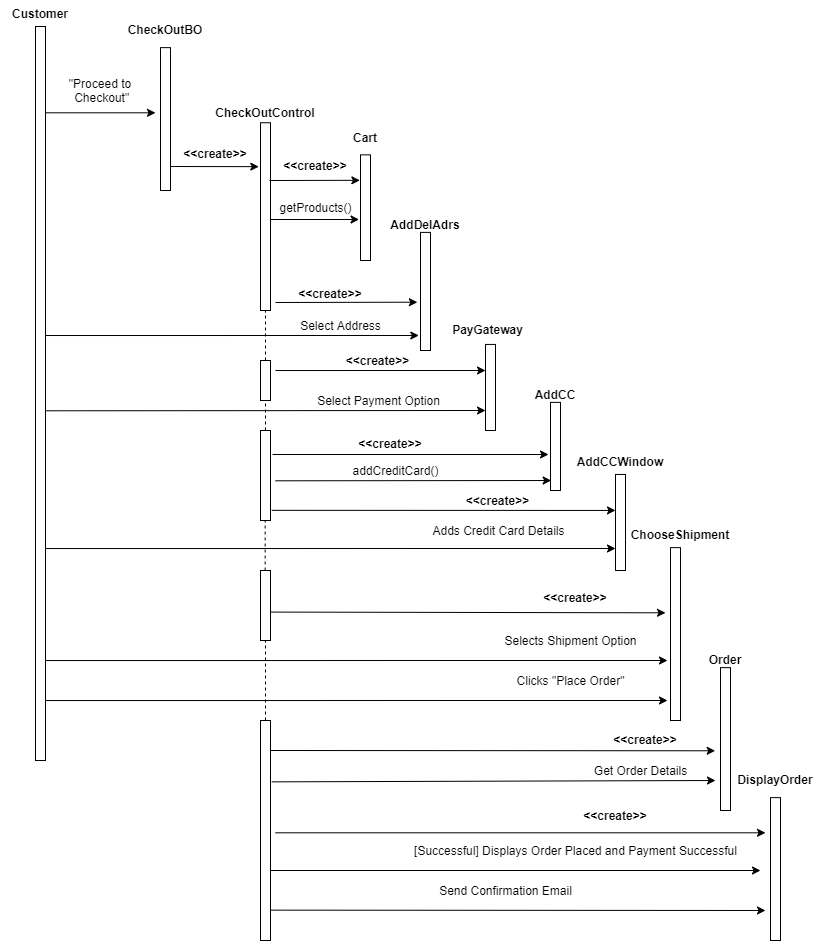
|  |  |
| --- | --- |
| Use case name | CheckOut |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customer activates the CheckOut function.  2. Product is successfully CheckedOut or the Product could not be CheckedOut. |
| Entry conditions | Customer selects the product he/she wants to checkout |
| Exit conditions | Customers will sign-out after successfully buying the product. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

**The most refined final use cases:**

|  |  |
| --- | --- |
| Use case name | CheckOut |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customer activates the “Proceed to Checkout” function shown in AddToCartPage.  2. An object called “CheckOutControl” is created by CheckOutBO.  4. CheckOutControl gets all the products information available in the Cart and creates an AddDelAdrs object and invokes the address selection operation.  5. CheckOutControl shows the saved addresses and displays that to the customer. Customer clicks on the “Select this address” button for selecting the address.  6. CheckOutControl gets the payment methods available and creates a PayGateway object and invokes the payment selection operation.  7. CheckOutControl receives all payment gateway methods and displays that to the customer. Customer clicked on the “Add credit card” link.  8. CheckOutControl creates an AddCC object and invokes adding credit card operation.  9. Control object prompts the user to enter card details in AddCCWindow.  10. CheckOutControl gets all types of shipment methods available and creates a chooseShipment object and invokes the shipment selection operation.  11. CheckOutControl receives the shipment method and displays to the customer.  12. Customer selects the shipment method and clicks on the “place your order” button.  13. CheckOutControl creates an Order object and gets the order status along with order number and expected delivery date with the tracking id.  14. CheckOutControl object displays order details “Payment Success” in DisplayOrderPage.  15. CheckOutControl also emails with a confirmation. |
| Entry conditions | Customer selects the product he/she wants to checkout |
| Exit conditions | Customers will sign-out after successfully buying the product. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

## **Dynamic model:**

**Sequence diagram:**

****

**//AddDelAdrs is supposed to be ShippingAddress, AddCC is supposed to be CreditCard.**

# Customers are able to review a product and give the product a rating (1–5).

## **Functional model:**

Scenarios:**Scenarios:**

Functionality: Customers can review the product they purchased from Amazon.

Functionality name: ReviewProduct

Example 1: Vidya is able to review the product iPhone 7 she purchased on Amazon.

|  |  |
| --- | --- |
| Scenario name | SuccessfulReview:ReviewProduct |
| Participating Actor instances | Vidya:Customer |
| Flow of Events | 1. Vidya clicks on the review button to review the iPhone 7. 2. Then she reviews the iPhone 7 with a rating of 4 stars. |

**Initial use cases:**

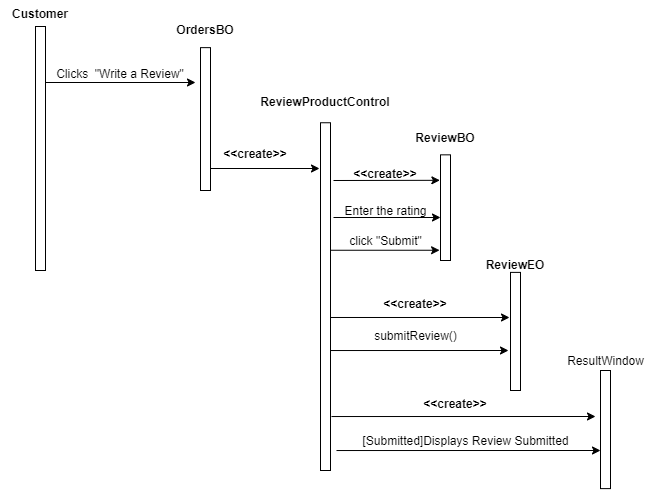
|  |  |
| --- | --- |
| Use case name | ReviewProduct |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customer clicks on the review button of the product to review. 2. Then the customer reviews the product with a rating from 1 to 5. 3. Finally the customer submits the review. |
| Entry conditions | Amazon my orders web page has been loaded. |
| Exit conditions | Customer submitted the review to Amazon |
| Quality Requirements | Submission should be processed in 2 seconds. |

**The most refined final use cases:**

|  |  |
| --- | --- |
| Use case name | ReviewProduct |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customer clicks on the “Write a review” button. 2. The OrdersBO creates the control object ReviewProductControl. 3. The ReviewProductControl creates ReviewBO and takes the customer to write a review of the product. 4. Customers can select the rating they wish to give. 5. Then the customer can submit the review by clicking the “Submit” button on ReviewBO. 6. The ReviewProductControl creates Review and invokes submitReview() operation. 7. ReviewProductControl creates ResultWindow BO and displays “review submitted”. |
| Entry conditions | Amazon my orders web page has been loaded with at least one order. |
| Exit conditions | Customer submitted the review to Amazon |
| Quality Requirements | Submission should be processed in 2 seconds. |

## **Dynamic model:**

**Sequence diagram:**



# Customers are able to sign up for an account.

## **Functional model:**

**Scenarios:**

Functionality: Customers can SignUp to the Amazon website by creating an account for themself. So, they can logIn using their account.

Functionality name: SignUp

Example 1: Ujjwal is able to SignUp into the Amazon web page by giving his details and creating an account.

|  |  |
| --- | --- |
| Scenario name | SignUpSuccessful:SignUp |
| Participating Actor instances | Ujjwal:Customer |
| Flow of Events | 1. Ujjwal activates the SignUp function by clicking on “SignUp” to create a personal account. 2. The webpage asks Ujjwal for details. 3. Ujjwal enters all the details like his   First name: Ujjwal  Last name: Shah  Email Id: [Ujjwal123@gmail.com](mailto:Ujjwal123@gmail.com)  Password: \*\*\*\*\*\*\*\*\*\*  Confirm password: \*\*\*\*\*\*\*\*\*\*  Address: 106 Elm Ave, Hackensack, NJ, 07601  PhoneNo: 1234567891   1. Ujjwal is successfully Signed Up. |

Example 2: Ujjwal is not able to SignUp into the Amazon web page by giving his details.

|  |  |
| --- | --- |
| Scenario name | SignUpFailure:SignUp |
| Participating Actor instances | Ujjwal:Customer |
| Flow of Events | 1. Ujjwal activates the SignUp function by clicking on “SignUp” to create a personal account. 2. The webpage asks Ujjwal for details. 3. Ujjwal enters all the details like his   First name: Ujjwal  Last name: Shah  Email Id: Shah[123@gmail.com](mailto:Ujjwal123@gmail.com)  Password: \*\*\*\*\*\*\*\*\*  Confirm password: \*\*\*\*\*\*\*\*\*\*  Address: 106 Elm Ave, Hackensack, NJ, 07601  PhoneNo: 1234567891   1. Ujjwal SignUp failed and he got a message saying “account is already registered or check your username and password”. |

**Initial use cases:**

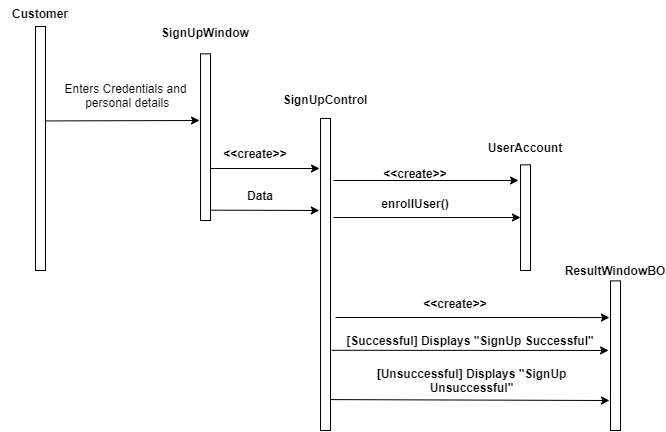
|  |  |
| --- | --- |
| Use case name | SignUp |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customer clicks on the SignUp button on the Homepage. 2. Customers are prompted to enter their details like First name, Last name, Email Id, Password, Confirm password, Address and PhoneNo. 3. Customer is either Signed Up successfully or a SignUp failure page is displayed. |
| Entry conditions | Amazon Web Page for SignUp has been loaded. |
| Exit conditions | SignUp is confirmed of success or failure. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

**The most refined final use cases:**

|  |  |
| --- | --- |
| Use case name | SignUp |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. A SignUpWindow asks the customer to enter his name, email address, contact and address information and asks the customer to create a username and password. After that the customer needs to click the enroll button. //Starts from Customer clicks on the SignUp  2. A SignUpControl is created which gets the data of customers from the SignUpWindow.  3. SignUpControl creates entity object UserAccount and invokes operation EnrollUser to record the First name, Last name, email address, contact and address information.  4. SignUpControl creates a ResultWindowBO. System shows successful signup or signup failed due to the details already existing or incorrect username or password. |
| Entry conditions | Amazon Web Page for SignUp has been loaded. |
| Exit conditions | SignUp is confirmed of success or failure.. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

## **Dynamic model:**

**Sequence diagram:**



# Customers are able to login to their account.

## **Functional model:**

**Scenarios:**

Functionality: Customers can login to the Amazon website by giving his/her userId and password.

Functionality name: LogIn

Example 1: Vidya is able to LogIn into the Amazon web page by giving her username and password

|  |  |
| --- | --- |
| Scenario name | LogInSuccessful:LogIn |
| Participating Actor instances | Vidya:Customer |
| Flow of Events | 1. Vidya clicks on “LogIn” in the Amazon homepage. 2. Then she enters “[vidya12@gmail.com](mailto:vidya12@gmail.com)” as a username. 3. Enters “p@ssword14” as her password. 4. Vidya then clicked on the “LogIn” button. 5. Vidya is now logged in successfully. |

Example 2: Vidya is not able to LogIn into the Amazon web page by giving her username and password an error is displayed.

|  |  |
| --- | --- |
| Scenario name | LogInFailure:LogIn |
| Participating Actor instances | Vidya:Customer |
| Flow of Events | 1. Vidya clicks on “LogIn” in the Amazon homepage. 2. Then she enters “[vidya@gmail.com](mailto:vidya12@gmail.com)” as a username. 3. Enters “pssword14” as her password. 4. Vidya then clicked on the “LogIn” button. 5. Vidya is now shown a LogIn failure page showing either the username or password is wrong. |

**Initial use cases:**

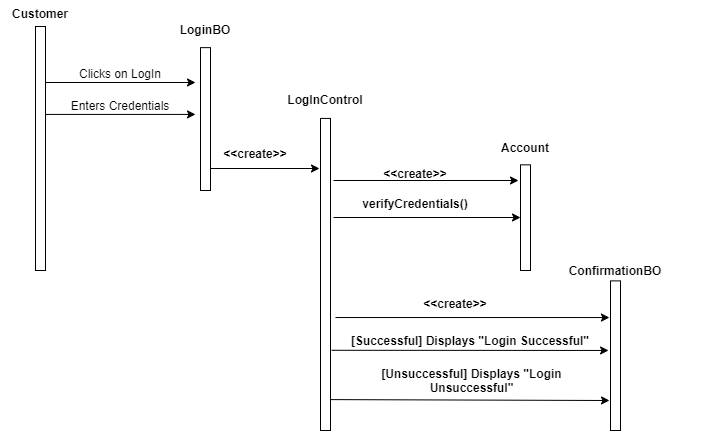
|  |  |
| --- | --- |
| Use case name | LogIn |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customer clicks on the LogIn button on the Homepage. 2. Customers are prompted to enter their username and password. 3. Customer is either LoggedIn successfully or a LoggedIn failure page is displayed. |
| Entry conditions | Amazon Web Page for login has been loaded. |
| Exit conditions | Login is confirmed of success or failure. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

**The most refined final use cases:**

|  |  |
| --- | --- |
| Use case name | LogIn |
| Participating Actor | Initiated by Customer |
| Flow of Events | 1. Customer clicks on the LogIn function in the Amazon homepage.  2. customer enters a username, password and clicks on LogIn contained in the LogInBO.  3. LogInControl is created by LogInBO.  4. Control object creates Account object and invokes the operation to verify the correctness of the username and password.  5. The LogInControl creates a ConfirmationBO that shows the customer “LogIn successful” if your details are valid, else “LogIn unsuccessful” if your details are invalid. |
| Entry conditions | Amazon Web Page for login has been loaded. |
| Exit conditions | Login is confirmed of success or failure. |
| Quality Requirements | Results must be displayed in less than 3 seconds |

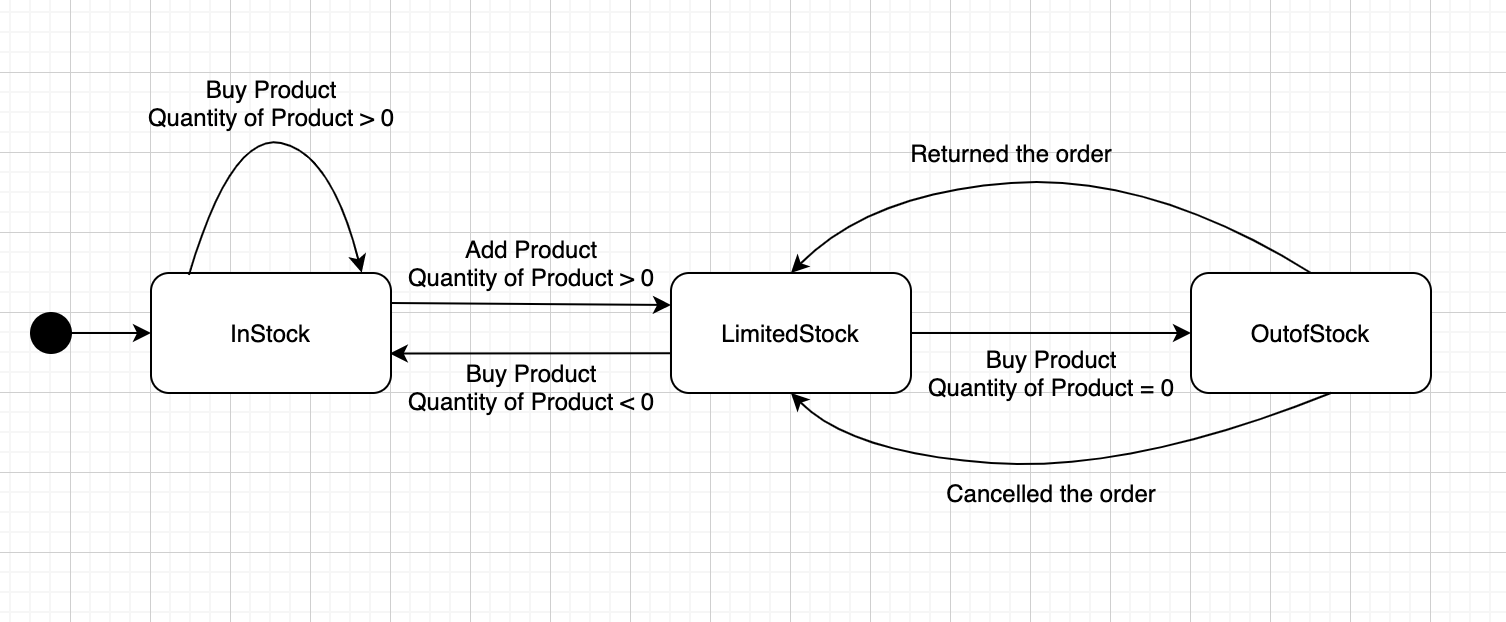
## **Dynamic model:**

**Sequence diagram:**

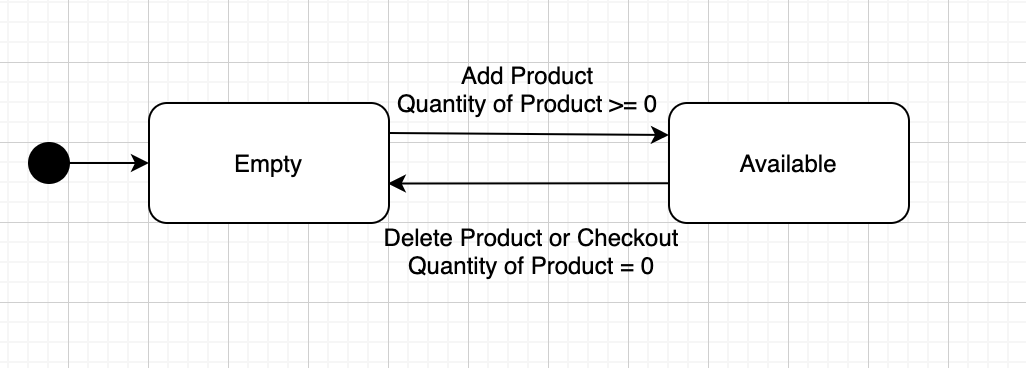


**State diagrams:**

**Product:**

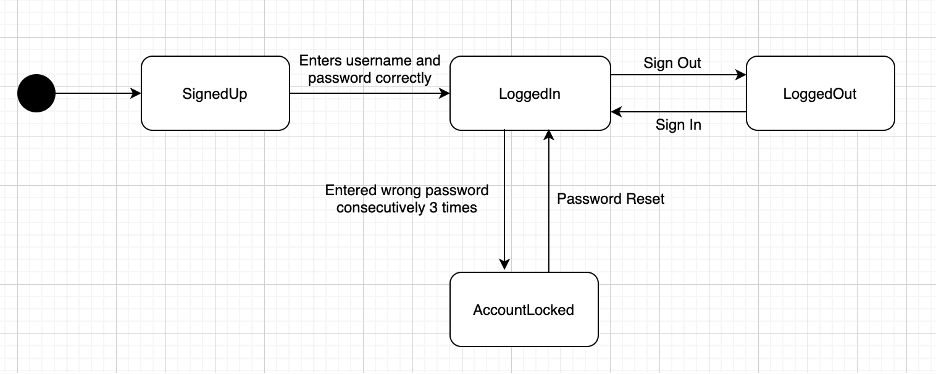
****

**Shopping Cart:**

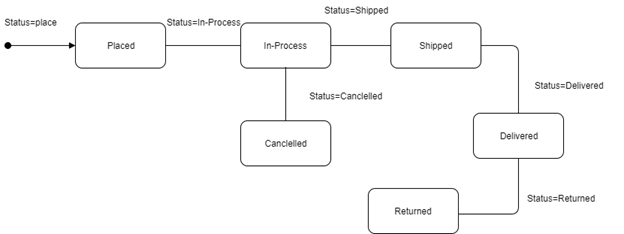
****

**//Available is supposed to be Non-Empty.**

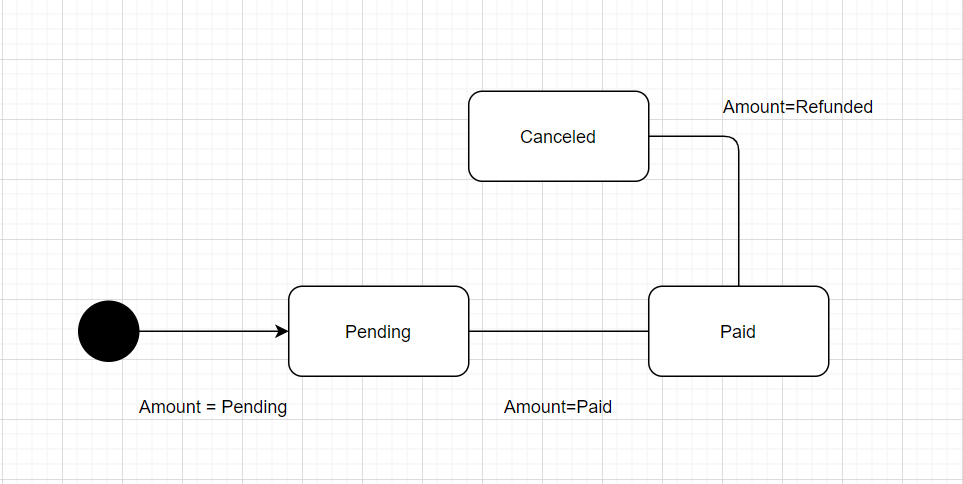
**Account:**

****

**Order:**

****

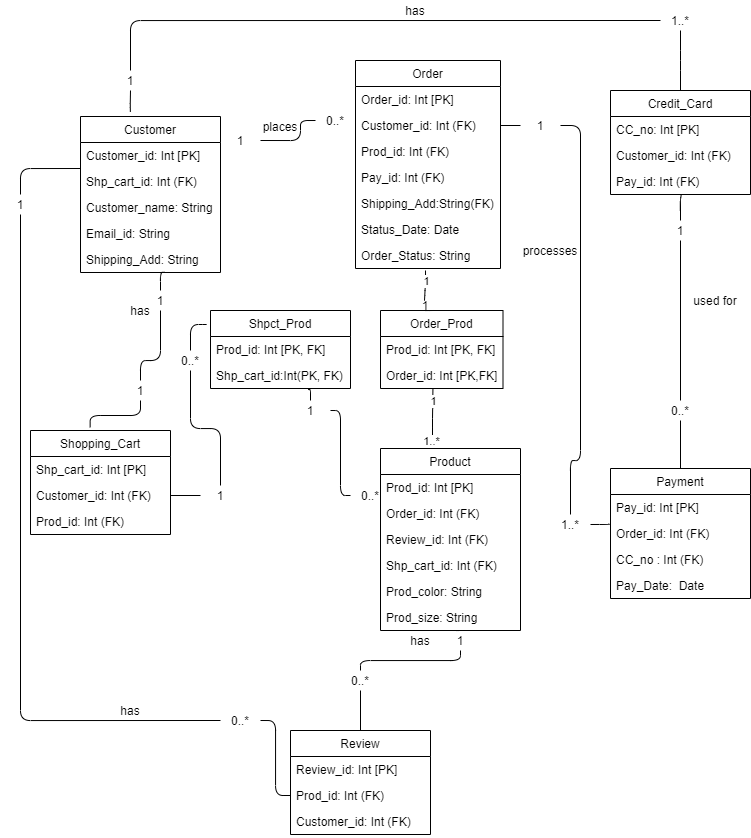
**Payment:**

****

**//Missed transition from Pending to Canceled.**

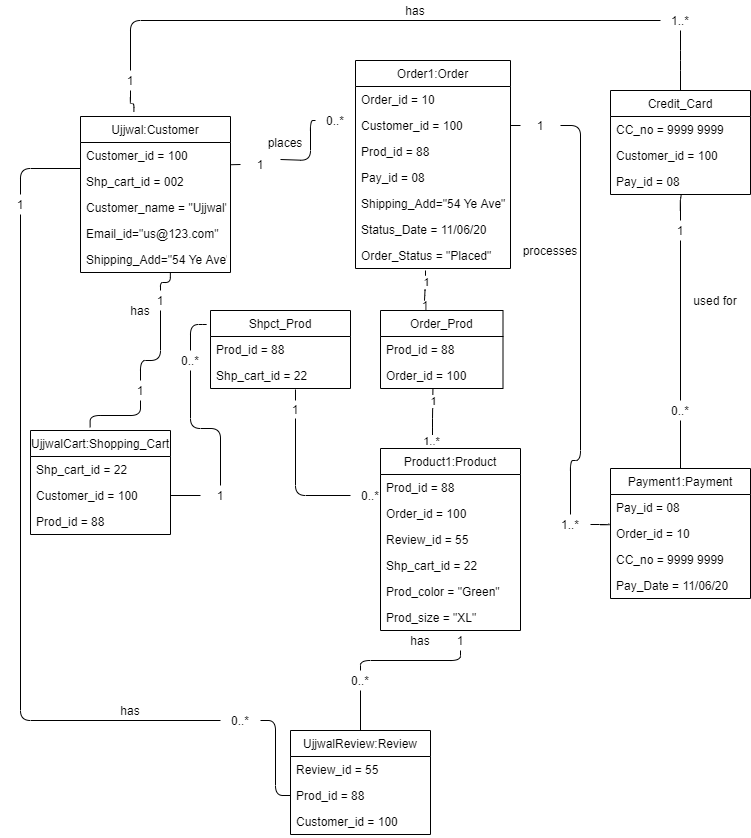
**Analysis object model:**

**Class diagram**:



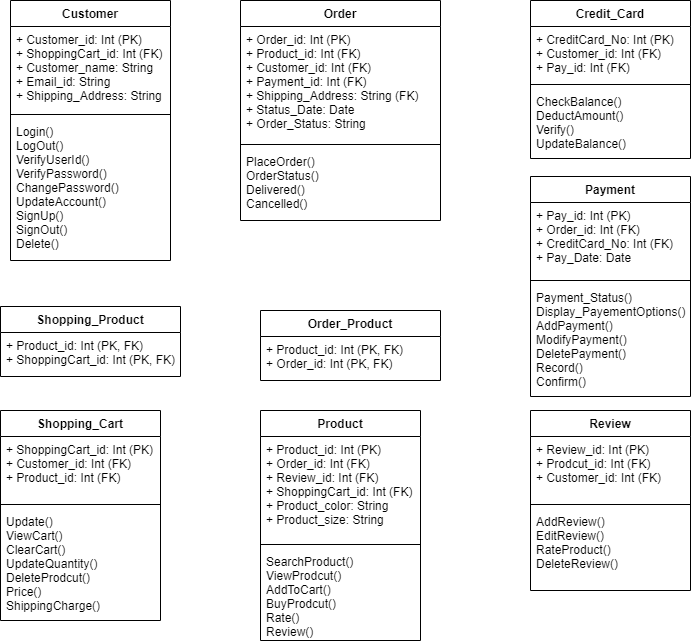
//Prod\_id is not FK in order. Order\_id is not FK in Product. Not necessary because you have class Order\_Prod.

**Object diagram:**



**Attributes and Operations:**

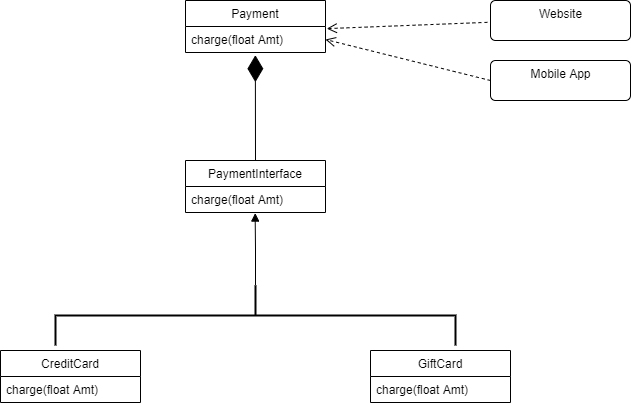
**//Better names are Deliver(), Cancel() for Order.**

****

Part-C

Object Design

**Design Pattern: The operation to be implemented differently is Payment()**

****

**Order:**

|  |
| --- |
| **context** Order **inv**:  Order\_ID !=NULL AND NoOfItems > 0 AND ShipingAddress !=NULL  **context** Order::cancel(String Order\_ID) **pre**:  OrderStatus == “Placed” OR OrderStatus == “InProcess”  **context** Order::cancel(String Order\_ID) **post**:  OrderStatus == “Canceled”  **context** Order::placeOrder(String Order\_ID) **pre**:  PaymentMethod != Invalid() //PaymentMethod is not in your class Order.  **context** Order::placeOrder(String Order\_ID) **post**:  OrderStatus == “Placed” |

**Product:**

|  |
| --- |
| **context** Product **inv**:  Product\_ID !=NULL AND TotalQuantity >= 0  **context** Product::BuyProduct(String Product\_ID) **pre**:  Quantity > 0 AND TotalQuantity >= Quantity //Both TotalQuntity and Quantity are not in your class Product  **context** Product::BuyProduct(String Product\_ID) **post**:  TotalQuantity = TotalQuantity - Quantity  **context** Product::AddToCart(String Product\_ID) **pre**: //AddToCart() is not an operation of Product. It’s an operation of ShoppingCart.  Quantity > 0 AND TotalQuantity >= Quantity  **context** Product::AddToCart(String Product\_ID) **post**:  TotalQuantity = TotalQuantity  **context** Product::DeleteProduct(String Product\_ID) **pre**:  TotalQuantity = 0 AND ProductAvailable = True  **context** Product::DeleteProduct(String Product\_ID) **post**:  ProductAvailable() = False |

**Shopping Cart:**

|  |
| --- |
| **context** Shopping Cart **inv**:  ShoppingCart\_ID !=NULL  **context** ShoppingCart::ClearCart(String ShoppingCart\_ID) **pre**:  NoOfItemsInCart > 0  **context** ShoppingCart::ClearCart(String ShoppingCart\_ID) **pre**:  NoOfItemsInCart = 0  **context** ShoppingCart::DeleteProduct(StringShoppingCart\_ID) **pre**:  Product\_ID !=NULL AND Quantity >= 1  **context** ShoppingCart::DeleteProduct(String ShoppingCart\_ID) **pre**:  Quantity = 0 |

**Review:**

|  |
| --- |
| **context** Review **inv**:  Review\_ID !=NULL AND Prod\_ID !=NULL AND Customer\_ID !=Null  **context** Review::addReview(String Review\_ID) **Pre**:  ReviewCount = ReviewCount AND ProductReviewed = False  **context** Review::addReview(String Review\_ID) **Post**:  ReviewCount = ReviewCount +1 AND ProductReviewed = True  **context** Review::deleteReview(String Review\_ID) **Pre**:  ReviewCount = ReviewCount AND ProductReviewed = True  **context** Review::addReview(String Review\_ID) **Post**:  ReviewCount = ReviewCount -1 AND ProductReviewed = False |

**Payment:**

|  |
| --- |
| **Context** Payment **inv:**  Pay\_id !=NULL AND PaymentAmount>0 AND  Verify(Customer\_id,CreditCard\_No) AND  CheckBalance(CreditCard\_No) AND  CheckStatus(CreditCard\_No) //For an invariant, use only attributes of the class, not operations.  **Context** Payment::Payment\_Status(Pay\_id) **Pre:**  Customer\_id == true AND  Balance >= Min\_Balance AND  Status == Active AND  Creditcard\_No == True  **Context** Payment::Payment\_Status(Pay\_id)**Post:**  Payment\_Status == Paid OR  Payment\_Status==Unpaid  **Context** Payment::AddPayment(Pay\_id) **Pre:**  Customer\_id == true AND  Balance >= Min\_Balance AND  Status == Active  CreditCard\_No == True  **Context** Payment::AddPayment(Pay\_id)**Post:**  Payment\_Status == Paid |

**Customer:**

|  |
| --- |
| **context** Customer **inv**:  Len(Username) == range(6,32) AND  Len(Password) == range(8, 20) AND  Password == (combination of at least one uppercase letter, at least one lowercase letter, at least one number and one special character like @!#$?) AND  NoOf3CFailure >=0 and <=3 AND  len(ContactNo) ==10 AND  Address ! = null AND  EmailAddress == format(email);  **Context** Customer::SignUp(String FName, String LName, String Address, int ContactNo, String EmailAddress, string Username, string Password) **Pre:**  Username != null AND  Password ! = null AND  FName != null AND  LName != null AND  len(ContactNo) ==10 AND  Address ! = null AND  EmailAddress != exist AND  EmailAddress == format(email);  **Context** Customer::SignUp(String FName, String LName, String Address, int ContactNo, String EmailAddress, string Username, string Password) **Post:**  Cust\_status == Active AND Cust\_id != NULL  len(Cust\_id) == 10;  **Context** Customer::UpdateAccount(String FName, String LName, String Address, int ContactNo, String EmailAddress, string Username, string Password) **Pre:**  Cust\_status == Update AND Cust\_id != NULL  **Context** Customer::UpdateAccount(String FName, String LName, String Address, int ContactNo, String EmailAddress, string Username, string Password) **Post:**  beforeedit\_contactNo != afteredit\_ContactNo OR  beforeedit\_Emailaddress != afteredit\_Emailaddress;  **Context** Customer::LogOut(String FName, String LName, String Address, int ContactNo, String EmailAddress, string Username, string Password) **Pre:**  Username\_status == LoggedIn;  **Context** Customer::LogOut(String FName, String LName, String Address, int ContactNo, String EmailAddress, string Username, string Password) **Post:**  Username\_status == LoggedOut;  **Context** Customer::DeleteAccount(String FName, String LName, String Address, int ContactNo, String EmailAddress, string Username, string Password) **Pre:**  Username\_status == Active;  **Context** Customer::DeleteAccount(String FName, String LName, String Address, int ContactNo, String EmailAddress, string Username, string Password) **Post:**  Username\_status == Deleted;  **Context** User::VerifyPassword(Username,Password) **Pre:**  Username == exist AND  Password != NULL;  **Context** User::VerifyPassword(Username,Password) **Post:**  Username\_status == Active;  **Context** User::VerifyUserID(Username,Password) **Pre:**  Username == exist AND  Password == Username.Password;  **Context** Customer::VerifyUserID(Username,Password) **Post:**  Username\_status == Active;  **Context** Customer::ChangePassoword(Username,Password,NoOf3CFailure,Username\_status) **Pre:** NoOf3CFailure == range (0,3) AND  Username\_status == Active;  **Context** Customer::ChangePassoword(Username,Password,NoOf3CFailure,Username\_status, NewPassword) Post: NoOf3CFailure == 0 AND  Username\_status == Active AND NewPassword != NULL AND NewPassword !=Password  **Context** Customer::Login(User\_ID,Password) **Pre:**  User\_ID == true AND  Password == true AND Status == “LoggedOut”  **Context** Customer::Login(User\_ID,Password) **Post:**  Username\_status == “LoggedIn”; |

**Credit Card:**

|  |
| --- |
| **context** CreditCard **inv**:  Creditcard\_No!= NULL AND CreditCard\_NoLenght==16  AND For each characer, CreditCard\_No>=0 AND CreditCard\_No<=9  AND BillingAddress!=NULL AND ExpirationDate>TodayDate AND CVV!=NULL  AND CVV.lenght==3 AND Each char in CVV>=0  AND Each char in CVV<=9 AND  Verify(Creditcard\_No,balance,productprice);  **Context** CreditCard::CheckBalance(Creditcard\_No) **Pre:**  Creditcard\_No == true  **Context** CreditCard::CheckBalance(Creditcard\_No) **Post:**  Credicardstatus= active OR  Credicardstatus= blocked OR  Credicardstatus= low balance  **Context** CreditCard::DeductAmount(Creditcard\_No,Productprice) Pre:  Creditcard\_No == true AND  Balance >= Min\_balance  Balance >= productprice  **Context** CreditCard::DeductAmount(Creditcard\_No,Productprice) Post:  Balance= totalbalance - productprice  **Context** CreditCard::UpdateAmount(Creditcard\_No,productprice) **Pre:**  Creditcard\_No == true AND  Balance >= Min\_balance AND  Balance >= productprice  **Context** CreditCard::UpdateAmount(Creditcard\_No,Productprice) **Post:**  Balance= totalbalance - productprice OR  Balance= totalbalance + productprice |

//All errors in Object Design: (-5)

Part-D

System Design

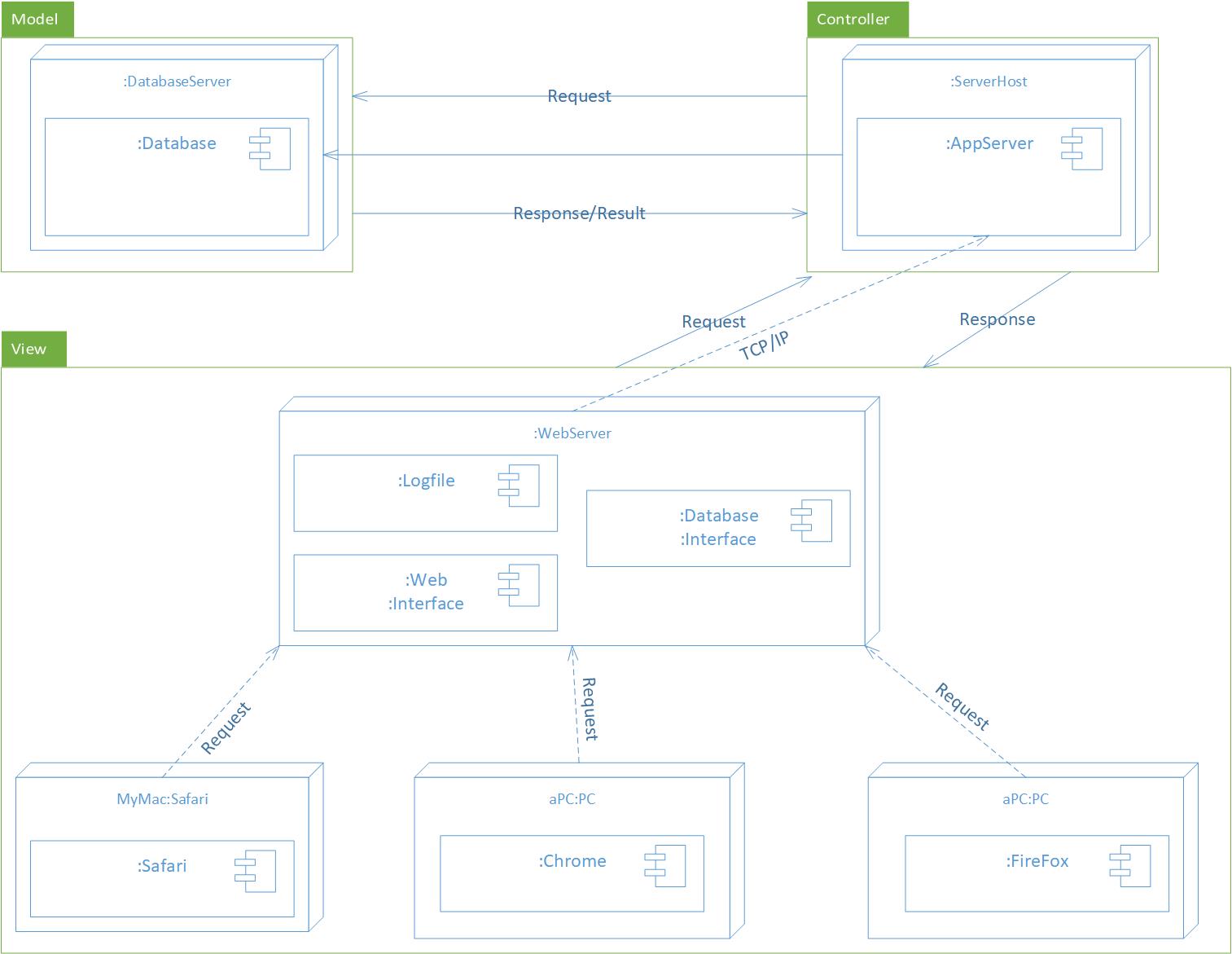
**Software Architecture:**

**System Design**, including subsystems and software architecture for the system; deployment diagram; access control (one of the three: a global access table, an access control list or a capability); strategies for global control; and at least one boundary use case, such as for configuration, start-up, shut-down, or an exception handling.

**We propose a combination of Client-Server Architecture and Model/View/Controller (MVC) architecture**.

**Deployment Diagram:** is used to show the relationship between run time components and the hardware nodes

**Deployment Diagram**



**Access Control**

(Customer, Order, ViewOrderstatus()) //Customers are not allowed to change Order status, deliver.

(Customer, Order, Cancelle())

(Customer, Order, Place())

(Customer, Product, ViewProduct())

(Customer, Product, SearchProduct())

(Customer, Product, Buy())

(Customer, Product, Rate())

(Customer, Product, Review())

//AddToWishlist() is an operation of WishList

(Customer, ShoppingCart, AddToCart())

(Customer, ShoppingCart, RemoveProduct())

(Customer, ShoppingCart, ViewCart())

(Customer, ShoppingCart, ClearCart())

(Customer, ShoppingCart, SaveforLater())

(Customer, ShoppingCart, UpadteQuantity())

(Customer, ShoppingCart, TotalPrice())

(Customer, ShoppingCart, ShippingCharge())

(Customer, Review , SubmitReview())

(Customer, Payment, DisplayPaymentContent())

(Customer, Payment, ViewPayment\_Status())

(Customer, Payment, AddPayment())

(Customer, Payment, DeletePayment()) //Customers may not be allowed to delete/modify any payment.

(Customer, Payment, ModifyPayment())

(Customer, Payment, Record())

(Customer, Payment, Confirm())

(Customer, Payment, Update())

(Customer, Customer, ViewProduct()) //Operations here are not in your class Customer. Must be consistent.

(Customer, Customer, SearchProduct())

(Customer, Customer, Buy())

(Customer, Customer, CheckOut())

(Customer, Customer, UpdateAccount())

(Customer, Customer, AddToWishlist())

(Customer, Account, Login()) //Account is not in your class diagram. Account is Customer.

(Customer, Account, Signup())

(Customer, Account, GuestUser())

(Customer, Account, Delete())

(Customer, Account, ChangeUsername())

(Customer, Account, ChangePassword())

(Customer, Account, UpdateInformation())

(Customer, Credit\_Card, CheckBalance())

(Customer, Credit\_Card, DeductAmount())

(Customer, Credit\_Card, Verify())

(Customer, Credit\_Card, UpdateBalance())

**Strategies for global control:**

Both “Event-driven Control” and “Threads” are the strategies for global control as the application has multi-tasking.

**Boundary Use Case:**

|  |  |
| --- | --- |
| *Use case name* | Exception handling |
| *Participating actor* | Initiated by customer |
| *Flow of events* | 1. Customer login to the online shopping website. 2. Customer search for the product by entering the product name in the search bar. 3. The system throws an exception because of the internal errors. 4. The system handles the exception and sends a generic output which shows the error message. 5. Instead of the exception, the customer could view the error message which is generic. |
| *Entry conditions* | Customer logged in to the online website. |
| *Exit conditions* | Customer is able to view the message displayed. |
| *Quality requirements* | Messages are displayed to the customer which must be generic and within the 2-3 seconds. |

//All errors in System Design: (-3)