Design Document
Methods and Tools in SW Development

I. Group Information

Group Number: 7

Group Member names/netIDs:

- Jeffrey Chancellor
  - jlc1541
- Robert Dilworth
  - rkd103
- Vishwa Patel
  - vnp39
- Connor Sparkman
  - cps260

What classes are you going to have? Explain why.

Given the provided requirements, we propose that eight classes will be necessary to satisfy the project's demands. The classes are as follows: (A) Users; (B) Cart; (C) Order_history; (D) Storage; and (E) Inventory_Items, where the subsequent objects—(1) Headphones, (2) PCs, (3) Smartphones, and (4) Gaming_Consoles—would comprise the supplies for purchase.

The User class is necessary as it serves to centralize and identify a customer's cart. Specifically, a hypothetical Cart class would presumably link an Inventory_Item's primary key with that of a User's primary key. This relation, when queried based on a particular User's primary key, would produce the complete list of items they would like to purchase. Moreover, the log-in feature mentioned in the requirement could be implemented by simply checking whether the customer's input (username and password) exists in the User table. Namely, if the username 'John_Doe' existed in the User table, then the program could operate accordingly, prompting for the user's password. Otherwise, the program could output an error message, indicating that the customer provided invalid credentials (and printing the number of remaining log-in attempts). In the context of this project, the terms 'user' and 'customer' are synonymous, referencing the User class and the related User table that will externally store its contents. A user object would serve as a wrapper encapsulating fields such as username, password, payment_information, etc.

As mentioned in the User class explanation, a Cart class is necessary given the project's end goal, to produce an e-commerce store. The cart class would house the primary keys (or identifying names or numerical labels) of a user object and an item object. This will allow for the program to pinpoint a customer and his or her related collection of items.

Due to the similarities of the data stored in both the Cart and Order_History classes, we believe that it would be prudent to create an additional object called Storage. The Cart and

Order_History classes will invoke an object of type Storage to house the client's current cart status and their finalized order (once they elect to checkout). As a disclaimer, every occurrence of the word 'list' can be read as a dynamic array, given the nature of modifying a customer's cart, order_history, etc.

Based on the wording of the project's requirements (namely the phrase "assume each category item [will] be handled independently"), our store's inventory will not consist of a singular inventory class but four distinct item categories. The categories will reflect the types of items that will be up for sale in our e-commerce store. In that way, the inventory will be comprised of the item types: Headphones, PCs, Smartphones, and Gaming_Consoles. Given the limited scope of the project, limiting our store's repertoire is a necessary set to ensure the feasibility of deploying the software and the subsequent shop.

The last class our program will possess is an Order_History object. This class will functionally mirror the operations of the proposed Cart class, storing a customer's user_id and cart's status prior to checkout (i.e., the item_ids associated with the user_id), to serve as a ledger of the customer's purchase history. This class is necessary as the cart is inherently volatile. Particularly, the requirements dictate that the customer should be able to add and remove items from their cart. Moreover, when a customer checks out, the program should "remove the items from the user's shopping cart." For this reason, having an additional class to store this data is an integral part of the e-commerce store. While not going into the specifics, the program (and by extension the store) could push the state of the cart to this class before processing the user's payment information and clearing their shopping cart.

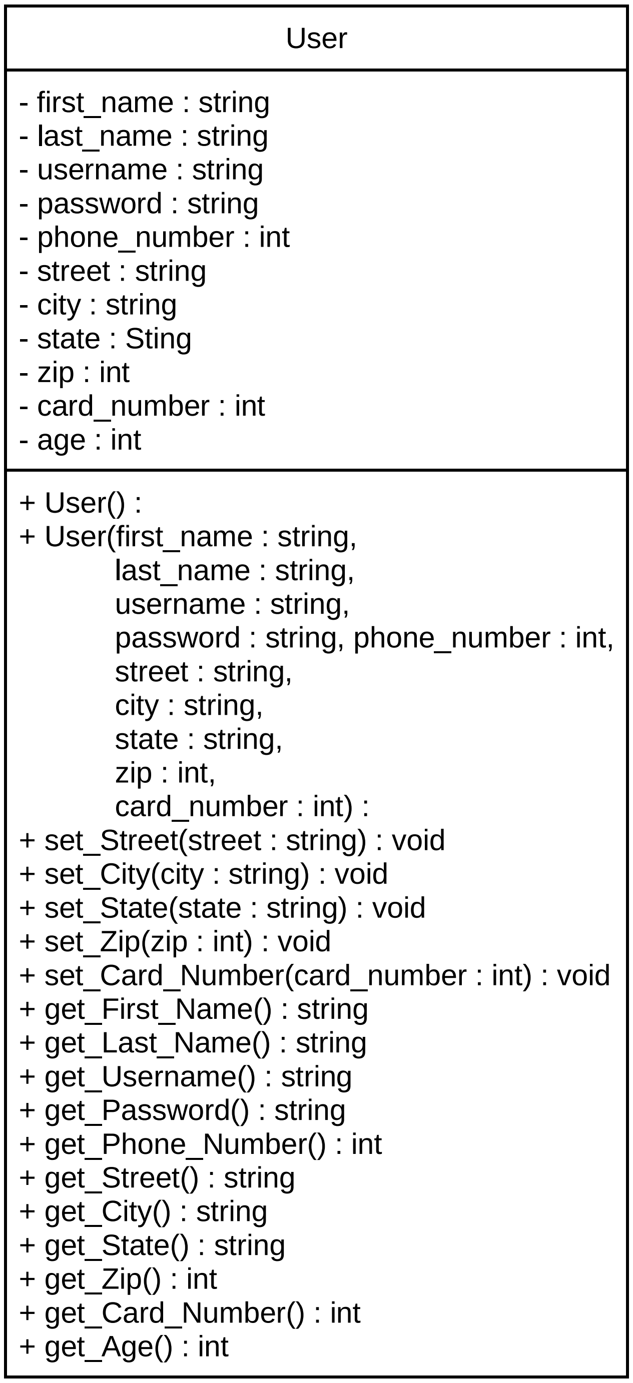
The following bulleted list outlines the classes that will exist in our program.

- **Users**
    - The customer will be considered logged-in if they exist in the user table, and they provide the necessary credentials.
- **Order_History**
    - This is a derived relation that will store (A) the user_id and (B) cart status prior to checkout (i.e., the fields stored in the cart) to serve as a ledger of the customer's purchase history. I propose this table given that the cart is volatile.
- **Cart**
    - Self-explanatory
- **Storage**
    - Houses a list (a python dictionary) and a variable storing the total number of elements within said list
- **Inventory_Items** (Propossed Items)
    - *Headphones*
        - (item_id, no_in_stock, make, model, ear_placement, color, manufacturer, connectivity, noise_cancellation, battery_capacity {35 hr}, range, price)
    - *PCs*
        - (item_id, no_in_stock, make, model, processor, RAM, device_id, product_id, system_type{64-bit, 32-bit}, model_type{traditional, touch,

    two-in-one}, operating_system, dimensions{length, width}, storage_size {500 GB, etc.}, price, release_date, battery_capacity {15 hr})

- *Smartphones*
    - (item_id. no_in_stock, make, model, network_capability {4G, 5G, etc.}, network_carrrier, dimensions {length, width}, screen_resolution {800x1792}, memory_card_type {SD, mirco SD, mini SD}, operating_system, charger_type {USB-C, USB-Micro-B, etc.} color, storage_size {32 GB}, price, release_date)
- *Gaming Consoles*
    - (item_id, no_in_stock, make, model, brand, price, backward_compatibility, DVD_player, cartridge_type, color, internet_connectivity, handheld, online_services, FPS, screen_resolution, release_date, storage_size {1 TB})

II. Detailed Class Diagrams



```
User
----------------------------------------
- first_name : string
- last_name : string
- username : string
- password : string
- phone_number : int
- street : string
- city : string
- state : Sting
- zip : int
- card_number : int
- age : int
----------------------------------------
+ User() :
+ User(first_name : string,
        last_name : string,
        username : string,
        password : string, phone_number : int,
        street : string,
        city : string,
        state : string,
        zip : int,
        card_number : int) :
+ set_Street(street : string) : void
+ set_City(city : string) : void
+ set_State(state : string) : void
+ set_Zip(zip : int) : void
+ set_Card_Number(card_number : int) : void
+ get_First_Name() : string
+ get_Last_Name() : string
+ get_Username() : string
+ get_Password() : string
+ get_Phone_Number() : int
+ get_Street() : string
+ get_City() : string
+ get_State() : string
+ get_Zip() : int
+ get_Card_Number() : int
+ get_Age() : int
```

- User() – a null constructor that creates an empty user object with string values equivalent to “” and integer variables corresponding to 0.
- User(…) – a constructor that populates a user object’s private fields (first_name, last_name, username, password, phone_number, street, city, state, zip, card_number, and age)
- set_Street(…) – updates the street field of a user object

- set_City(…) – updates the city field of a user object
- set_State(…) – updates the state field of a user object
- set_Zip(…) – updates the zip field of a user object
    - Note that the particular setters are present due to the requirements stipulations (A) can “edit the shipping information” and (B) can “edit the payment information. Moreover, it is atypical for a system to allow for a user, once their account has been created, to alter their age and name. Granted, it is debatable whether it is necessary to include the functionality to allow the customer to change his or her log-in credentials. We simply do not account for that feature as it is not explicitly stated in the requirements.
- set_Card_Number – updates the card_number field of a user object
- get_First_Name() – returns the field first_name
- get_Last_Name() – returns the field last_name
- get_Username() – returns the field username
- get_Password() – returns the field password
- get_Phone_Number() – returns the field phone_number
- get_Street() – returns the field street
- get_City() – returns the field city
- get_State() – returns the field state
- get_Zip() – returns the field zip
- get_Card_Number() – returns the field card_number
- get_Age() – returns the field age

Cart
- shopping_cart : Storage
+ checkout_Cart(username : string, order_history_obj : Order_History) : bool + view_Cart(username : string) : string + add_to_Cart(username: stiring, itemID : int, no_item : int) : bool + remove_from_Cart(username : string, itemID : int, no_item : int) : bool

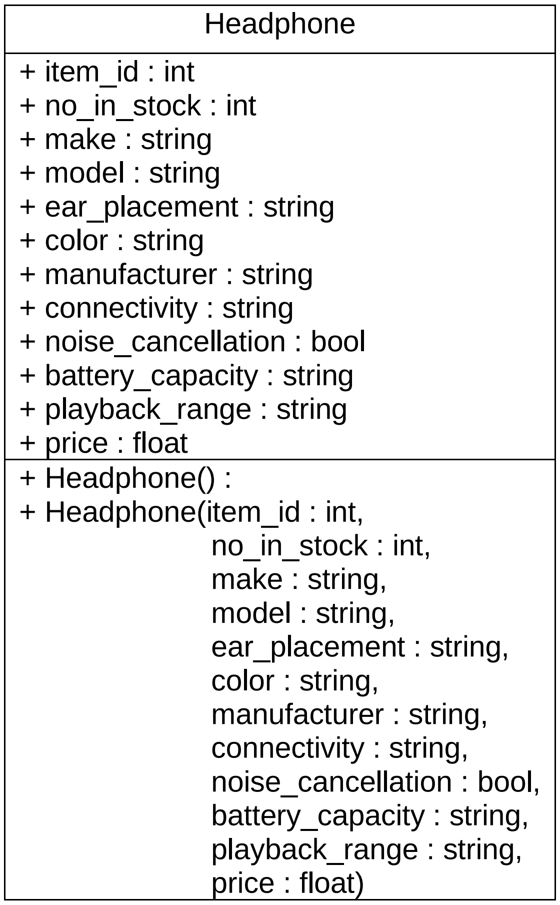
- checkout_Cart(…) – using the provided username to identify the customer of interest, this function will (A) remove the items from the user's shopping cart, (B) edit the stock information to lower the present amount, and (C) push the finalized order (the current state of the cart prior to checkout) to an instance (object) of the Order_History class; this function passively encompasses the requirement "add an order to the user's order history," as it would be foolhardy to give the user said functionality when he or she has not yet paid for all of the items in their cart
- view_Cart(…) – displays the elements of the cart associated with a given customer's username
- add_to_Cart(…) – appends a new entry—comprised of a customer's username, the item's ID, and the total number of items (of the ID specified prior) that the user would like to purchase—into the cart
- remove_from_Cart(…) – discards an existing entry, deleting its contents from the cart

Storage
- list{username : string, item_id : int, no_item : int} : - size : int
+ view_Storage(username : string) : string + add_to_Storage(username: stiring, itemID : int, no_item : int) : bool + remove_from_Storage(username : string, itemID : int, no_item : int) : bool

- list – a class attribute that will store a list, a python dictionary to be exact, of the customer’s username, the items identification number, and the number of units the customer would like to purchase
- size – a class attribute that will house the total number of items in the list; could be considered a count of the items in the record
- view_Storage(…) – outputs the contents of the list for a particular customer based on their username
- add_to_Storage(…) – adds a new element to the list populating the fields with the passed parameters
- remove_from_Storage(…) – eliminates an element from the list based on information loaded into the method

Order_History
- record[] : Storage
+ view_Order_History() : string

- record – a nested list (the outer list, a typical python list, contains instances of Storage objects, or python dictionaries); the indexing of the outer list represents the order number, whereas the indexing of the inner list indicates the items that made up the user’s cart before checkout
- view_Order_History() – outputs the elements that comprise the user’s order history, using dual indexing
    - Take note that our design does not explicitly allow for the Order_History object to append a finalized cart order to its record. We believe it is unintuitive to give the object such access when the customer may or may not have paid for the items in their cart (i.e., they checked out).



```
Headphone
---------------------------------------------
+ item_id : int
+ no_in_stock : int
+ make : string
+ model : string
+ ear_placement : string
+ color : string
+ manufacturer : string
+ connectivity : string
+ noise_cancellation : bool
+ battery_capacity : string
+ playback_range : string
+ price : float
---------------------------------------------
+ Headphone() :
+ Headphone(item_id : int,
            no_in_stock : int,
            make : string,
            model : string,
            ear_placement : string,
            color : string,
            manufacturer : string,
            connectivity : string,
            noise_cancellation : bool,
            battery_capacity : string,
            playback_range : string,
            price : float)
```

- Headphone() – a null constructor that creates an empty Headphone object with string values equivalent to “”, integer variables corresponding to 0, and Boolean fields set to False
- Headphone(…) – a constructor that populates a Headphone object’s fields (item_id, no_in_stock, make, model, ear_placement, color, manufacturer, connectivity, noise_cancellation, battery_capacity, playback_range, and price)
    - Note that the requirements do not specify that inventory items must have the capability to alter their fields, our implementation will treat the inventory item categories as data types

```
PC
------------------------------------
+ item_id : int
+ no_in_stock : int
+ make : string
+ model : string
+ processor : string
+ RAM : int
+ device_id : int
+ product_id : int
+ system_type : string
+ model_type : string
+ operating_system : string
+ length : float
+ width : float
+ storage_size : string
+ price : float
+ release_date : string
+ battery_capacity : string
------------------------------------
+ PC() :
+ PC(item_id : int,
        no_in_stock : int,
        make : sting,
        model : string,
        processor : string,
        RAM : int,
        device_id : int,
        product_id : int,
        system_type : string,
        model_type : string,
        operating_system : string,
        length : float,
        width : float,
        storage_size : string,
        price : float,
        release_date : string,
        battery_capacity : string)
```

- PC() – a null constructor that creates an empty PC object with string values equivalent to "", integer variables corresponding to 0, and float fields set to 0.0
- PC(…) – a constructor that populates a PC object's fields (item_id, no_in_stock, make, model, processor, RAM, device_id, product_id, system_type, model_type, operating_system, length, width, storage_size, price, release_date, and battery_capacity)

Smartphone
+ item_id : int
+ no_in_stock : int
+ make : string
+ model : string
+ network_capability : string
+ network_carrier : string
+ length : float
+ width : float
+ screen_resolution : string
+ memory_card_type : string
+ operating_system : string
+ charger_type : string
+ color : string
+ storage_size : string
+ price : float
+ release_date : string
+ Smartphone() :
+ Smartphone(item_id : int,
no_in_stock : int,
make : string,
model : string,
network_capability : string,
network_carrier : string,
length : float,
width : float,
screen_resolution : string,
memory_card_type : string,
operating_system : string,
charger_type : string,
color : string,
storage_size : string,
price : float,
release_date : string)

- Smartphone() – a null constructor that creates an empty Smartphone object with string values equivalent to “”, integer variables corresponding to 0, and float fields set to 0.0
- Smartphone(…) – a constructor that populates a Smartphone object’s fields (item_id, no_in_stock, make, model, network_capability, network_carrrier, length, width, screen_resolution, memory_card_type, operating_system, charger_type, color, storage_size, price, and release_date)

```
Gaming_Console
--------------------------------------------
+ item_id : int
+ no_in_stock : int
+ make : string
+ model  : string
+ brand : string
+ price : float
+ backward_ compatibility : bool
+ DVD_player : bool
+ cartridge_type : string
+ color : string
+ internet_connectivity : bool
+ handheld : bool
+ online_services : bool
+ FPS : int
+ screen_resolution : string
+ release_date : string
+ storage_size : string
--------------------------------------------
+ Gaming_Console() :
+ Gaming_Console(item_id : int,
                 no_in_stock : int,
                 make : string,
                 model : string,
                 brand : string,
                 price : float,
                 backward_compatibility : bool,
                 DVD_player : bool,
                 cartridge_type : string,
                 color : string,
                 internet_connectivity : bool,
                 handheld : bool,
                 online_services : bool,
                 FPS : int,
                 screen_resolution : string,
                 release_date : string,
                 storage_size : string)
```

- Gaming_Console() – a null constructor that creates an empty Gaming_Console object with string values equivalent to “”, integer variables corresponding to 0, float fields set to 0.0, and Booleans preset to False
- Gaming_Console(…) – a constructor that populates a Gaming_Console object’s fields (item_id, no_in_stock, make, model, brand, price, backward_compatibility, DVD_player, cartridge_type, color, internet_connectivity, handheld, online_services, FPS, screen_resolution, release_date, and storage_size)

III. Menu Information

Before login:

- Login
- Create Account
- Exit Program

After login:

- Inventory Information
    - Go back
    - View Inventory Items
        - Headphones
            - Go back
            - View all Headphones
            - Add Headphone to Cart
            - Remove Headphone from Cart
        - PCs
            - Go back
            - View all PCs
            - Add PC to Cart
            - Remove PC from Cart
        - Smartphones
            - Go back
            - View all Smartphones
            - Add Smartphone to Cart
            - Remove Smartphone from Cart
        - Gaming Consoles
            - Go back
            - View all Gaming Consoles
            - Add Gaming Console to Cart
            - Remove Gaming Console from Cart
- Cart Information
    - Go back
    - View Cart
    - Add Item to Cart
    - Remove Item from Cart
    - Checkout
        - **Adds Order to Order History (Amongst Other Things)**
- Order History
    - Go back
    - View Order History
- User Profile
    - Go back
    - Edit Shipping Information

    - Edit Payment Information
    - Delete Account
- Logout
    - Go back
    - Sign Out
- Exit Program

Does your menu cover all requirements given? If not, explain why certain requirements don't have a distinct menu option?

Despite not directly allowing the user to "add an order to…[his or her] order history," the 'menuing' above should adequately address the project's requirements. Specifically, there does not exist a distinct menu option for adding a collection of items from a user's inventory to his or her order history because that functionality would be counterproductive. Intuitively, a system should add a batch of items to a user's order history if and only if he or she has checkout and paid their final balance. Due to this assumption, our design is structured in such a way so that the requirement (add an order to the user's order history) is satisfied during the checkout process.

Additionally, the 'menuing' allows the user to add/remove items in two locations: first, they navigate the store's inventory and second, when they access the cart directly.

IV. Information Storage

How is your group storing information?

Due to the nature of the project, our team feels that it would be better to store our client's information in a *database* rather than an external file.

Include one of these lines of questioning based on your storage schema:

- If a database, what kind of database?
    - How many database tables will you have?

Given our exposure to databases such as MariaDB, we will forgo using non-relational and object-oriented databases in favor of the familiar *relational* variant. In that way, we will store seven tables. The tables are: User, Cart, Order_History, Headphones, PCs, Smartphones, and Gaming_Consoles.

What information are you going to store in each (table / file depending on schema)?

- *Users*
    - *first_name*
    - *last_name*
    - *username*
    - *password*
    - *phone_number*
    - *street*
    - *city*
    - *state*
    - *zip*
    - *card_number*
    - *age*
- *Cart*
    - *cart_id*
    - *username*
    - *item_id*
    - *no_item*
- *Order_History*
    - *order_id*
        - **a nested list with fields similar to the Cart table**
        - *username*
        - *item_id*
        - *no_item*
- *Headphones*
    - *item_id*
    - *no_in_stock*

    - *make*
    - *model*
    - *ear_placement*
    - *color*
    - *manufacturer*
    - *connectivity*
    - *noise_cancellation*
    - *battery_capacity*
    - *range*
    - *price*
- *PCs*
    - *item_id*
    - *no_in_stock*
    - *make*
    - *model*
    - *processor*
    - *RAM*
    - *device_id*
    - *product_id*
    - *system_type*
    - *model_type*
    - *operating_system*
    - *length*
    - *width*
    - *storage_size*
    - *price*
    - *release_date*
    - *battery_capacity*
- *Smartphones*
    - *item_id*
    - *no_in_stock*
    - *make*
    - *model*
    - *network_capability*
    - *network_carrier*
    - *length*
    - *width*
    - *screen_resolution*
    - *memory_card_type*
    - *operating_system*
    - *charger_type*
    - *color*
    - *storage_size*
    - *price*

    - *release_date*
- *Gaming_Consoles*
    - *item_id*
    - *no_in_stock*
    - *make*
    - *model*
    - *brand*
    - *price*
    - *backward_compatibility*
    - *DVD_player*
    - *cartridge_type*
    - *color*
    - *internet_connectivity*
    - *handheld*
    - *online_services*
    - *FPS*
    - *screen_resolution*
    - *release_date*
    - *storage_size*