Covid Commerce

Design Document

10/27/2020

Version 1

Lucky13

Emma Mickas, Tala Karaki, Mantz Wyrick

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

The purpose of this software design document is to provide the design details of Covid Commerce, an ecommerce website that allows users to buy and sell merchandise. This design document presents the designs used in implementing the project and will give an overview of the project's development. The designs described, follow the requirements specified in the Software Requirements Specifications document prepared for the project. Lastly, this document could be used by developers who want to upgrade or modify the website.

The goal of this project is to create an online store. There will be two types of users: a buyer user (1) and a seller user (2). Both types of users will have an account, and the ability to login and logout of the website. The account will contain fields that can be edited, such as a name, email, password, contact number, address, payment information, and order history. From there, the website will allow users to search and sort items. Every seller user will have a store page, which displays all available merchandise and information, such as price, rating, etc.… Seller users may post items to their store, view their store, update item details, and report users. Buyer users may browse the main store or the store of a seller, add items to an online cart, and make a purchase. After a purchase, buyers can give items a rating, and report a seller. Sellers can also report buyers.

The design document is divided into 6 sections:

Section I includes a description of what was included in this iteration and the motivation behind the implementations included.
Section II includes the overview of the architecture of the project.
Section III includes the design details. More specifically the subsystem design, the data design, and the user interface design.
Section IV includes the progress report for each iteration.
Section V includes the testing plan for the project (iteration 2)
Section VI includes any references used while building design document and the project itself.

Document Revision History

Rev 1.0 10-25-2020 Initial version
Rev 1.1 10-31-2020 Included User Interface Design
Rev 1.2 11-1-2020 Included Progress Report, included database models
Rev 1.3 11-2-2020 Revised all sections, added to subsystem design

II. Architecture Design - Everyone

II.1. Overview

Our architecture follows the Model-View-Controller architecture as a means to separate backend and frontend logic. Our architecture involves more communication between the frontend and backend using a controller than the Client-Server architecture. Because of multiple tables whose information will be updated frequently (items, and seller/buyer user information), we thought it would be the most efficient to have a model to handle that data, a controller to manage the changing of the data and a view to show those changes separately.

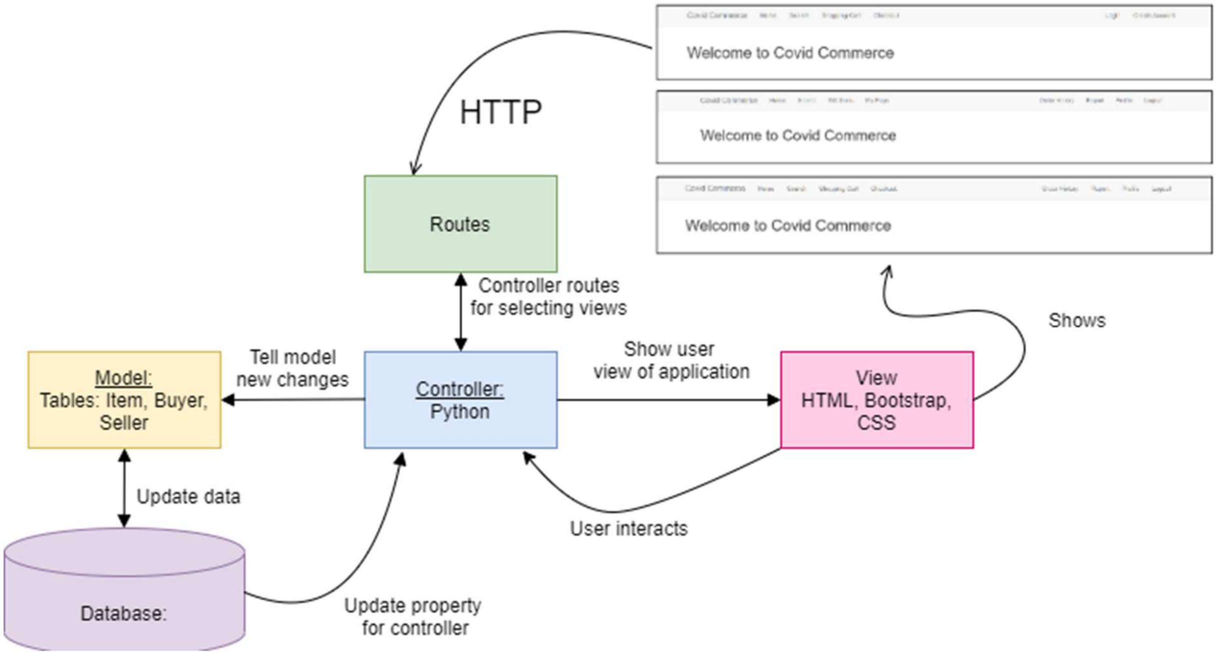
Our architecture includes buyer, seller, and guest subsystems which include the possible functionality of the system based on what kind of user is accessing it. Those who are logged in as buyers have functionality to browse and purchase products, those who are logged in as sellers have functionality to browse, post, and sell items, and those who are not logged in have the functionality to browse existing items (though they cannot sell or purchase items). Also included is a login manager subsystem that handles the registration and logging in of users so that they can access functionalities specific to the account they create/log in with. The buyer/seller operations subsystem handles all the functionalities that pertain to the buying and selling of items through the orders made by buyers and the orders made to sellers. Lastly, the data access subsystem of the architecture will manage the relationships between different models in the data base and their information. Data pertaining to users, orders, and items will be shared throughout the site.

The architecture of our software involves a database containing models for users of the site who want to sell products, models for users of the site who want to purchase products, and models for the items being exchanged. Each of these models have a comprehensive list of attributes that contain all the information associated with an individual instance of those models in order to promote cohesion. Additionally, each model contains no attributes that are dependent upon other models or functions not contained within themselves (except for database(db) relationships) so that they are not reliant on other parts of the system. This helps to limit coupling in the software. The database also contains forms in order to add buyers and sellers in the data-handling model part of the system.

Our architecture next includes python files dedicated to importing the required packages and a python file dedicated to handling the user interaction with data on the site. This makes up the controller part of the system. The controller handles forms that allow for new user registration, and in the future will handle new item registration and item purchasing. This controller will receive user input and update the data in the models of the model part of the system according to that input.

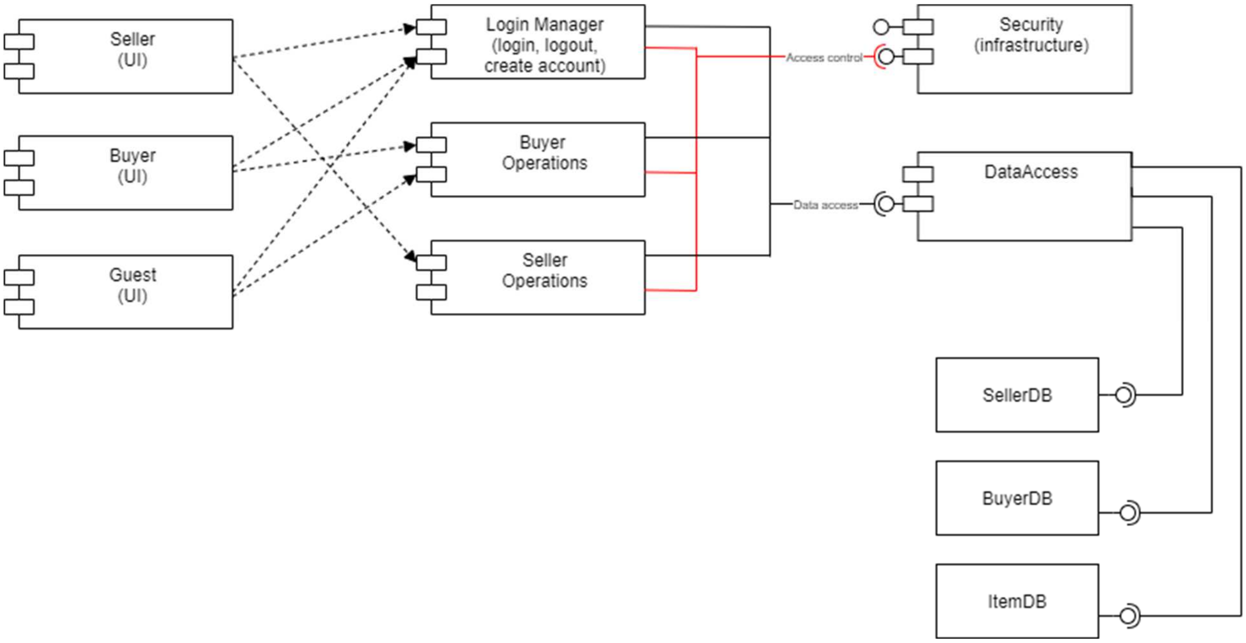
The last part of our architecture is the view part of the system. The view of our system is made through html templates that are built to take information from the controller (which accesses information from the model) and displays that information for a user to see and interact with. Any interactions the user has will then be handled by the controller.

The design of the Model-View-Controller architecture also promotes high cohesion and low coupling as each part of the system is functional on its own and compatible with other versions of other parts of the system. For example, the Model part of the system includes all its own functionality to be paired with multiple different views or controllers. This makes changing and updating the system simple as changing one part will not cause another part to break.


Welcome to Covid Commerce
Welcome to Covid Commerce
Welcome to Covid Commerce
HTTP
Routes
Controller routes for selecting views
Shows
Tell model new changes
Show user view of application
Model:
Tables: Item, Buyer, Seller
Controller:
Python
View
HTML, Bootstrap, CSS
Update data
User interacts
Database:
Update property for controller

III. Design Details

III.1. Subsystem Design



III.1.1. User subsystem

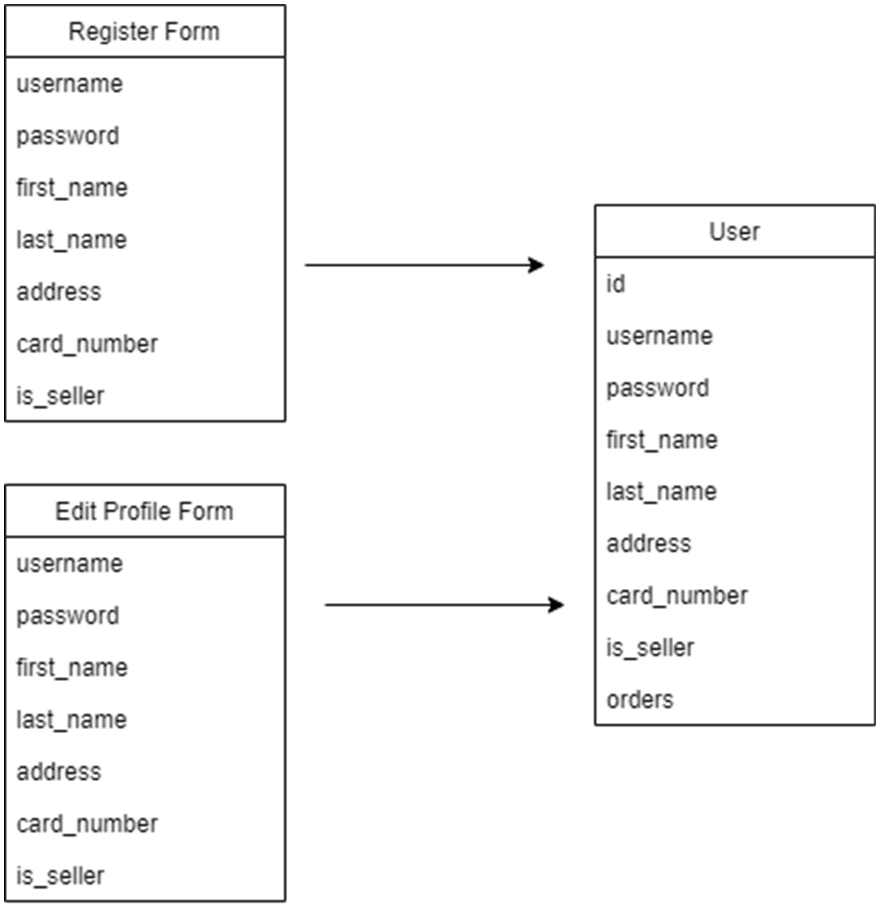
The user subsystem will contain and manage all users in the project: sellers and buyers. The system will offer types of users, and allow a different set of features depending on the type of user.

The role of the subsystem for sellers is to allow them to use features typical of a seller. Seller Users have the responsibility of adding items to the store for the other users to buy. The profile will contain the same attributes as that of a buyer, in which both must provide a name, email, address, and payment information. Seller users also have features that allow them to edit their profile, their items, and report users. The seller subsystem interacts with seller operations, login manager, security, and data access subsystems. It interacts with the seller subsystem because the users are adding items into the store and selling them. The user login manager subsystem is also involved because the users must have active profiles and the attributes are constantly being modified. The data access is subsystem is also heavily involved because these users can add items, remove items, and change their properties. There is an interdependency between the sellers and buyers because the inventory of the items will be directly impacting both.

The role of the user subsystem for buyers is to allow them to use features typical of a buyer. The buyer subsystem will have more capable features than the guest subsystem. The responsibility of the buyer subsystem is to separate the logged in user from a seller and a guest since they will all have different features associated with them. Some features of the buyer that differ from a guest include giving a review, viewing order history, reporting, view/editing profile, and logging out. The buyer subsystem will interact with the login manager, buyer operations, data access, and security subsystems. The buyer subsystem interacts with the login manager because the buyer can log out of the current buyer account. It also interacts with the buyer operations because the majority of the buyer operations and capable features come from the buyer operations subsystem. The buyer subsystem also interacts with the data access subsystem actively because when a user buys an item, gives a review, or reports a seller all that information goes to each appropriate database and updates it. There is an interdependency between the sellers and buyers because the inventory of the items will be directly impacting both.

Methods	URL	Description
register()	http://127.0.0.1:5000/register	Allows user to create either a seller or buyer account. All fields must be filled in and correctly.
edit_user_profile()	http://127.0.0.1:5000/edit_user_profile	Allows user to edit their existing profile that they are logged into.

- (in iteration -2) Provide your class level design for the subsystem. You should include a UML class diagram visualizing your class level design. In addition, explain each class in detail, specify and explain their methods.



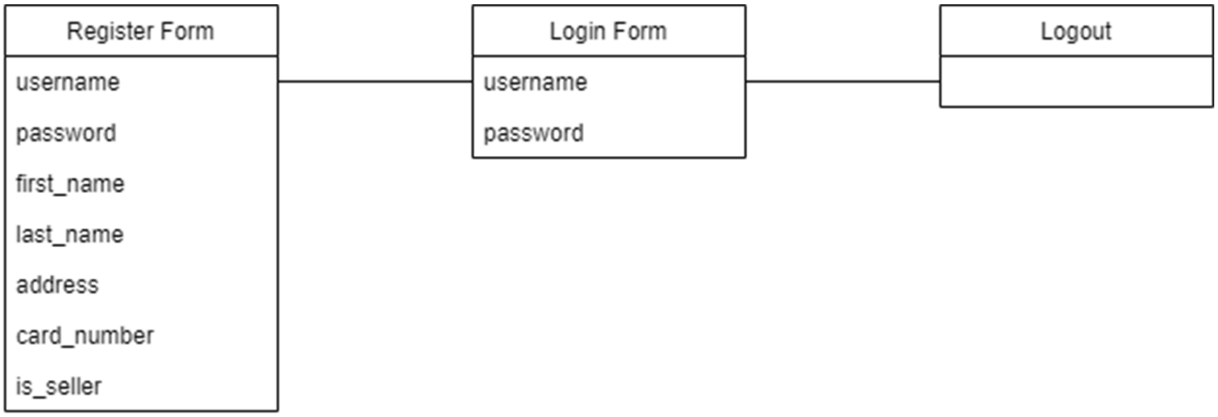
III.1.2. Login Manager subsystem

The role of the login manager is to allow users to create an account, login, and logout. The main responsibilities of this subsystem is to only allow one user to be logged in at a time, have unique user names and emails no matter if the account is for a buyer or seller, and require all fields are correctly filled in.

The login manager interacts with the seller, buyer, guest, and security subsystems. It interacts with the guest subsystem because a guest can create an account or login. A seller and buyer can logout. The login manager interacts heavily with the security subsystem because the users are required to put in a password so they can sign into their accounts. There are no subsystems that are interdependent with the login manager.

Methods	URL	Description
register()	http://127.0.0.1:5000/register	Allows user to create either a seller or buyer account. All fields must be filled in and correctly.
login()	http://127.0.0.1:5000/login	Allows user to login to created seller or buyer account.
logout()	http://127.0.0.1:5000/index	Allows user to logout of the currently logged in account.

- (in iteration -2) Provide your class level design for the subsystem. You should include a UML class diagram visualizing your class level design. In addition, explain each class in detail, specify and explain their methods.



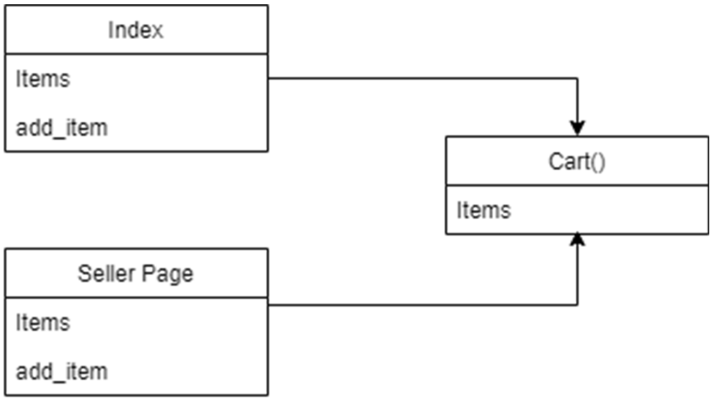
III.1.3. Buyer Operations subsystem

The role of the buyer operations subsystem is to allow a buyer perform operations on the website when they are logged in as a buyer. The responsibilities include searching, adding item to cart, viewing cart, checking out, viewing order history, writing a review, viewing/editing profile, reporting a seller, and logging out of the account. These operations will be different depending on if the user is a buyer or guest.

The buyer operations subsystem will interact with buyer, guest, and data access. The buyer operations subsystem interacts with the buyer and guest subsystems because both subsystems can use the features in the buyer operations subsystem. Changes made in the buyer operations subsystem will not affect the guest or buyer subsystems, however. The buyer operations subsystem actively interacts with the data access subsystem because whenever an item is bought it changes the data access subsystem's appropriate database. There are no interdependencies to other subsystems.

Methods	URL	Description
index()	http://127.0.0.1:5000/index	"Homepage" that displays items for sale.
seller_page(seller_id)	http://127.0.0.1:5000/seller_page/<seller_id>	Displays items only sold by the specific seller.
cart()	http://127.0.0.1:5000/cart	Displays all items added by the buyer to the shopping cart

- **(in iteration -2)** Provide your class level design for the subsystem. You should include a UML class diagram visualizing your class level design. In addition, explain each class in detail, specify and explain their methods.



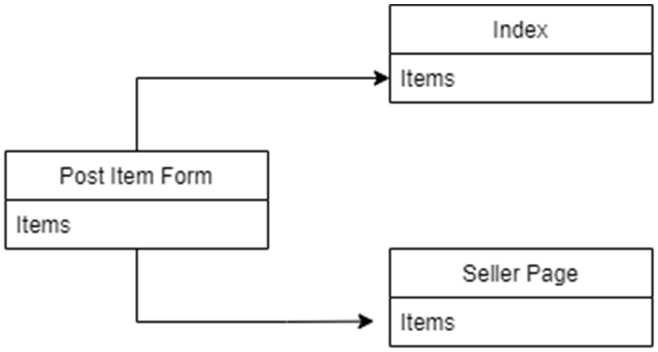
III.1.4. Seller Operations subsystem

The role of the seller operations subsystem is to allow a seller perform operations on the website when they are logged in as a seller. The responsibilities include searching, adding item, editing item, viewing/editing seller page, view order history, viewing/editing profile, reporting a buyer, and logging out of the account.

The seller operations subsystem will interact with seller, data access, and security subsystems. The seller operations subsystem interacts with the seller subsystems because the seller subsystem can use the features in the seller operations subsystem. Changes made in the seller operations subsystem will not affect the guest or seller subsystems, however. The seller operations subsystem actively interacts with the data access subsystem because whenever an item is added it changes the data access subsystem's appropriate database. It also interacts with the security when the user logins into a seller account. There are no interdependencies to other subsystems.

Methods	URL	Description
index()	http://127.0.0.1:5000/index	“Homepage” that displays items for sale.
seller_page(seller_id)	http://127.0.0.1:5000/seller_page/<seller_id>	Displays items only sold by the specific seller.
post_item(item_id)	http://127.0.0.1:5000/item/<item_id>	Allows seller to post items to the store

- **(in iteration -2)** Provide your class level design for the subsystem. You should include a UML class diagram visualizing your class level design. In addition, explain each class in detail, specify and explain their methods.



III.1.5. Security subsystem

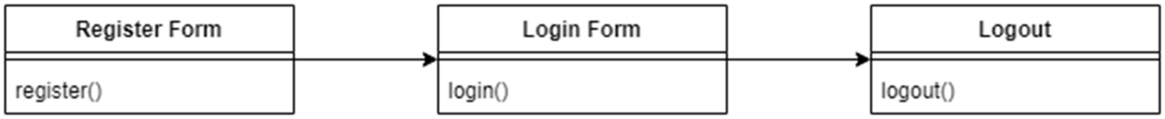
The role of the security subsystem is to keep user information safe. The responsibilities include hashing all passwords, saving and protecting user information.

The security subsystem interacts with seller, buyer, guest, login manager, buyer operations, and seller operations subsystems. For seller, buyer, and guest the security subsystems interact by allowing the subsystems’ users to login and logout while protecting the user information. The login manager subsystem interacts heavily with the security subsystem since all the users must go through the login manager for the security subsystem to verify the user. It interacts with the buyer and seller operations subsystems by requiring passwords for some of those operations to be usable. There are no interdependencies to other subsystems.

Methods	URL	Description
register()	http://127.0.0.1:5000/register	Allows user to create either a seller or buyer account. All fields must be filled in and correctly.

login()	http://127.0.0.1:5000/login	Allows user to login to created seller or buyer account.
logout()	http://127.0.0.1:5000/logout	Allows user to logout from account

- (in iteration -2) Provide your class level design for the subsystem. You should include a UML class diagram visualizing your class level design. In addition, explain each class in detail, specify and explain their methods.



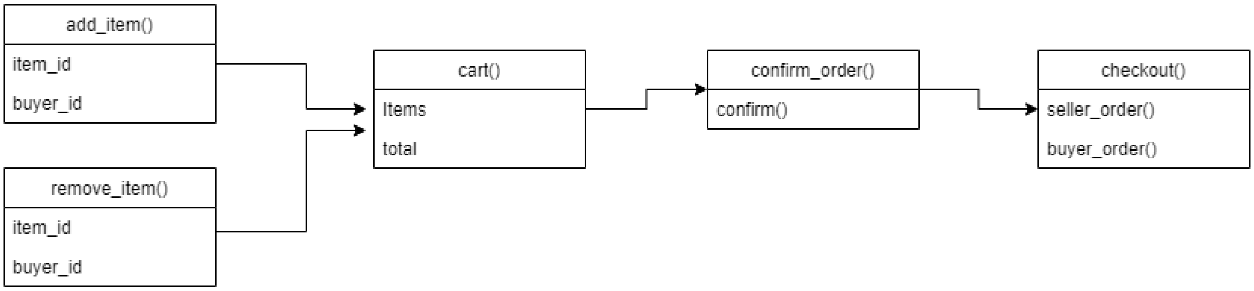
III.1.6. Data Access subsystem

The role of the data access subsystem is to control and allow changes to the databases. The changes to the databases will occur when items are added, bought, when an item is reviewed, and when a user has been reported.

The data access subsystem interacts with the buyer, seller, guest, login manager, buyer operations, and seller operations subsystems. All of these subsystems are not affected when the data access subsystem changes. The buyer, seller, and guest subsystems interact with the data access subsystem when an item is either bought or added. It interacts with the login manager when a new account is created and thus added to the appropriate database. The buyer and seller operations subsystems interact with data access subsystem when any operation taken will update the databases. There are no interdependencies to other subsystems.

Methods	URL	Description
add_item(item_id)	http://127.0.0.1:5000/add_item	Adds the specified item to the current_user's order.
Remove_item(item_id)	http://127.0.0.1:5000/remove _item	Removes item from the user's order
Cart()	http://127.0.0.1:5000/cart	List of all items in the user's shopping cart
checkout()	http://127.0.0.1:5000/checkout	Places the current_user's cart as an order.
Confirm_order()	http://127.0.0.1:5000/confirm_order	Confirm the order before checkout and placing the final order

- **(in iteration -2)** Provide your class level design for the subsystem. You should include a UML class diagram visualizing your class level design. In addition, explain each class in detail, specify and explain their methods.



III.2. Data design

1. Item: db.Model representing products that can be sold and purchased on the site.
    a. Schemas:
        i. id: A unique integer representing the specific item in the Item table.
        ii. item_name: A string representing the name of an item being sold and purchased. It does not have to be unique as some items may have the same name, but different sellers, descriptions, or traits.
        iii. item_description: A string representing a description of the item as entered by the seller of the item.
        iv. seller_id: An integer that is a ForeignKey to the id of the seller user who posted the item.
        v. price: A double that shows the price of an item which is selected by the seller who posts it.
        vi. posted_on: A timestamp representing the date that this item was posted by the seller.
        vii. num_available: An integer representing how many of the item is currently available.
        viii. initial_supply: An integer representing how many of the item was initially available when it was posted.
        ix. num_purchased: An integer recording how many of the item have been purchased.
        x. reviews: A one to many relationship between the Item and the reviews from the review table about the item.
        xi. bought_in: A many to many relationship that items have with orders in which they were purchased by buyers. Many items can be purchased in many different orders.
        xii. supplied_in: A many to many relationship that items have with orders in which they were sold by seller. Many items can be sold in many different orders.
2. User: db.Model and UserMixin type that represents a user that sells products on the site.
    a. Schemas:
        i. id: A unique integer representing the specific seller in the Seller table.

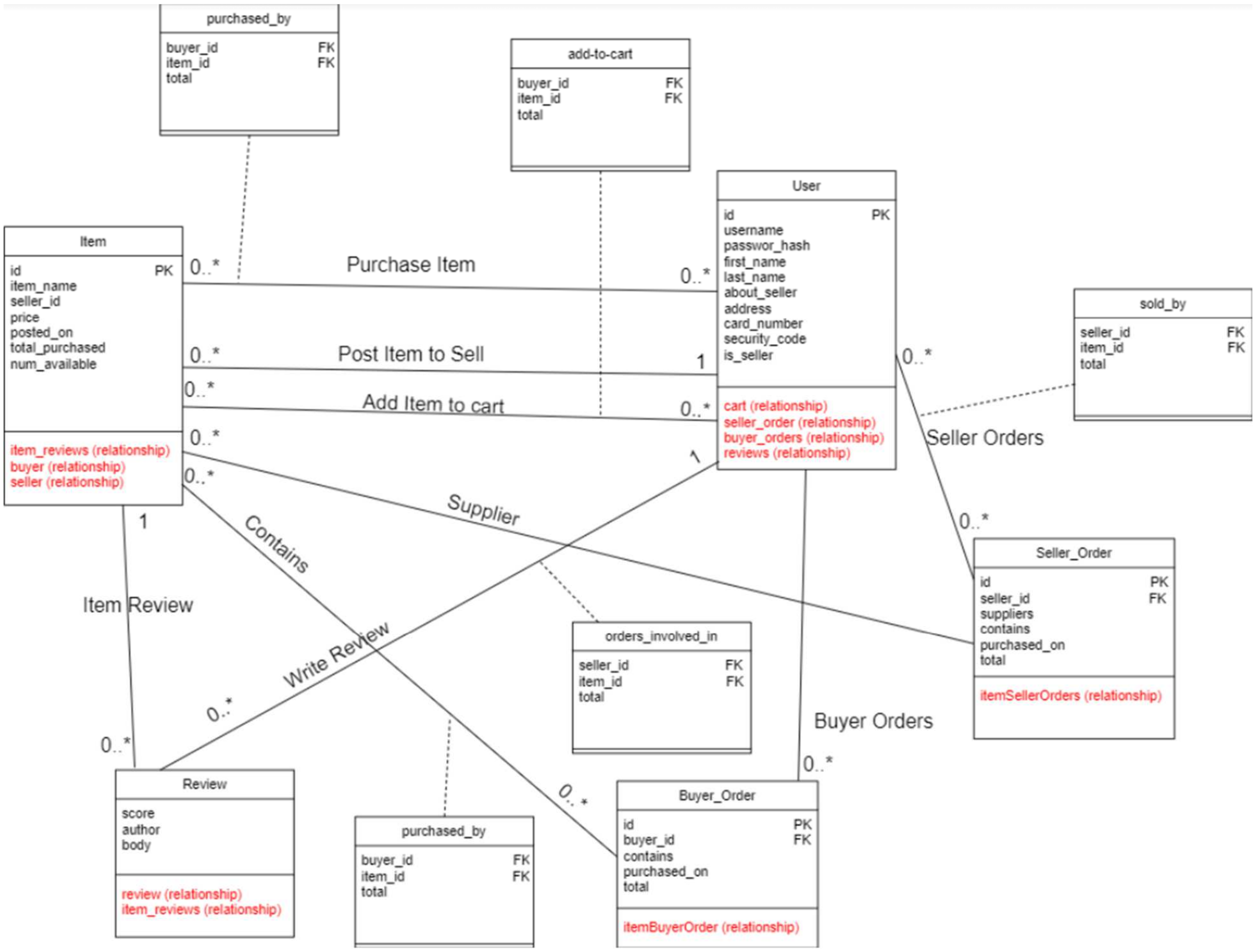
ii. username: A unique string representing the username of the seller. This is what will be visible to other users browsing the site when they see an item sold by the seller. The username is also used to log in to the site.
iii. email: A unique string representing the email of the user.
iv. password_hash: A string representing the password that the user will use to log in to the site.
v. first_name: A string representing the first name of the seller user.
vi. last_name: A string representing the last name of the seller user.
vii. about_seller: A string that a seller can add when they edit their account. This string is an opportunity for the seller to give information about themselves for potential customers of them.
viii. address: A string representing the address of the seller user for shipping and billing purposes.
ix. card_number: A string representing a credit or debit card number of the seller user for shipping and billing purposes.
x. security_code: A string representing a credit or debit card security code of the seller user for shipping and billing purposes.
xi. items: A one to many relationship between a seller user and the items from the Item table that they have posted to sell. The supplier keyword is the backref to access the seller from the Seller table that posted an item.
xii. in_cart: A many to many relationship between a buyer user an the items that are currently inside of their cart.
xiii. orders_to: A one to many relationship between the seller user and the orders from the Seller_Order table containing items that have been purchased in an order from the seller.
xiv. orders_by: A one to many relationship between the buyer user and the orders from the Buyer_Order table containing items that have been purchased in an order by the buyer.
xv. orders_involved_in: A many to many relationship between the supplier users in any orders (users who had any amount of items ordered in a buyer order) and the orders that they were involved in.
xvi. is_seller: A boolean field that is set to true to be used for customization of the website depending on whether the current user is a buyer or a seller.
xvii. item_reviews: A one to many relationship between the seller user and all of the reviews from the Review table left on items that the seller supplies.
xviii. reports_on: A one to many relationship between the seller user and any reports made about them.
xix. reports_by: A one to many relationship between the buyer user and any reports that they made about a seller.

b. Functions:
    i. __repr__(self): A method to get information about the user in the python shell.
    ii. set_password(self, password): A method to set the password of the user using password hashing from werkzeug.security.
    iii. get_password(self, password): A method to retrieve the password of the user using password hashing from werkzeug.security.
    iv. check_password(self, password): A method to check if a given password matches the user’s password using password hashing from werkzeug.security.

v. add_to_cart(self, item): A method to add an item to the user’s cart. It will check to make sure that the item does not exist in the cart to avoid duplicates.

vi. remove_from_cart(self, item): A method to remove an item from the user’s cart. It will check to make sure that the item does exist in the cart to avoid errors.

3. Seller_Order: db.Model that represents an order made to a seller including the items within that order that were purchased from that seller.
    a. Schemas:
        i. id: A unique integer representing the specific order in the Seller_Order table.
        ii. seller_id: A foreign key between the Seller_Order and the seller from the Seller table that is selling items in the order.
        iii. contains: A one to many relationship between a Seller_Order and the items from the Item table that were purchased in the order.
        iv. purchased_on: A timestamp for the date that the order was made on.
        v. total: A price of the items that were contained in this order.
        vi. buyer_id: A foreign key between the Seller_Order and the buyer from the Buyer table that purchased the order
    b. Functions:
        i. add_to_order(self, item): A method to add an item to the order. It will check to make sure that the item does not exist in the order to avoid duplicates.
4. Buyer_Order: db.Model that represents an order made by a buyer including all the items within that order that were purchased by that buyer.
    a. Schemas:
        i. id: A unique integer representing the specific order in the Buyer_Order table.
        ii. buyer_id: A foreign key between the Buyer_Order and the buyer from the Buyer table that purchased the order
        iii. contains: A one to many relationship between a Buyer_Order and the items from the Item table that were purchased in the order.
        iv. purchased_on: A timestamp for the date that the order was made on.
        v. Suppliers: A many to many relationship between the order and the suppliers (sellers) of all of the items in the order.
        vi. total: A price of the items that were contained in this order.
    b. Functions:
        i. add_to_order(self, item): A method to add an item to the order. It will check to make sure that the item does not exist in the order to avoid duplicates.
        ii. add_supplier(self, seller): A method to add a seller to the list of suppliers associated with the order. It will check to make sure that the seller does not already exist in the order to avoid duplicates.
5. Review: db.Model that represents a review left on an item that was ordered by a buyer.
    a. Schemas:
        i. score: An integer corresponding to the score of the item given by a buyer.
        ii. author: A relationship between the buyer writing the review and the review itself.
        iii. body: A string stating the actual review of the item given by the buyer.
        iv. item_id: A relationship between the review and the item it pertains to.


purchased_by
buyer_id FK
item_id FK
total
add-to-cart
buyer_id FK
item_id FK
total
User
id PK
username
passwor_hash
first_name
last_name
about_seller
address
card_number
security_code
is_seller
cart (relationship)
seller_order (relationship)
buyer_orders (relationship)
reviews (relationship)
Item
id PK
item_name
seller_id
price
posted_on
total_purchased
num_available
item_reviews (relationship)
buyer (relationship)
seller (relationship)
sold_by
seller_id FK
item_id FK
total
Purchase Item
0..*
0..*
Post Item to Sell
0..*
1
Add Item to cart
0..*
0..*
0..*
0..*
Supplier
Contains
1
1
Item Review
Write Review
0..*
0..*
0..*
Seller Orders
0..*
Seller_Order
id PK
seller_id FK
suppliers
contains
purchased_on
total
itemSellerOrders (relationship)
orders_involved_in
seller_id FK
item_id FK
total
Buyer Orders
0..*
0.. *
Review
score
author
body
review (relationship)
item_reviews (relationship)
purchased_by
buyer_id FK
item_id FK
total
Buyer_Order
id PK
buyer_id FK
contains
purchased_on
total
itemBuyerOrder (relationship)

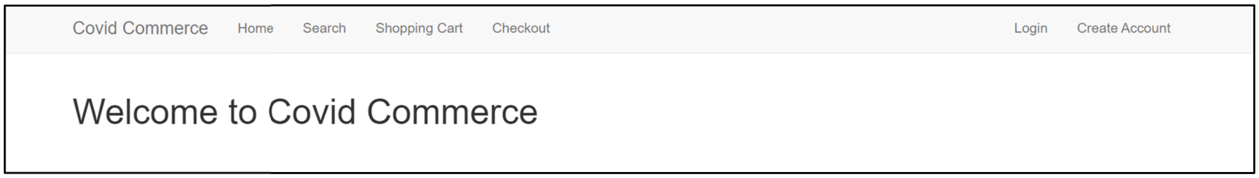
III.3. User Interface Design

The user interface of iteration 1 contains a login, logout, and register account page. These pages match use-cases #1-4, for creating a buyer/seller account, as well as login and logout. The navigation bar is an element in the user interface that will enable the remaining use-cases to be accessed. Depending on the current user (guest /seller /buyer), the navigation bar will display different pages to be accessed by the user.

1. ***The Navigation Bar:***
   The navigation bar will allow guest users who have not logged in to browse through items, add items to a shopping, and checkout. The option to login or create an account is also available.

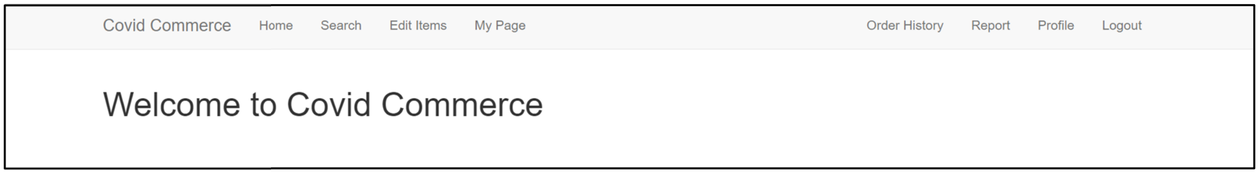
   Seller and buyer users can access an order history page to monitor all recent transactions, a report page to report other users, a profile page to update personal details, and a logout page. Buyer users will have the same access to browsing as guest users: search, shopping cart, and checkout. Seller users share the same browsing functionalities, for the main store and their personal page. Instead of adding items, they have an edit item page to update their merchandise, and a personal page to monitor their store.

Guest Users →



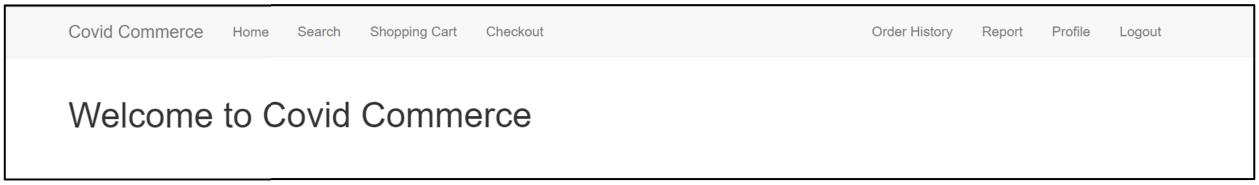
Welcome to Covid Commerce

Seller Users →



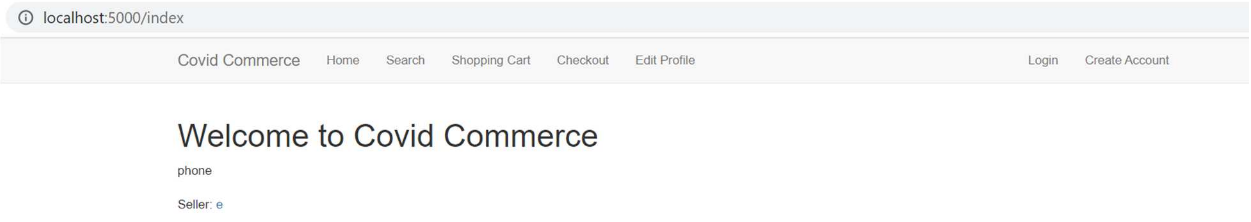
Welcome to Covid Commerce

Buyer Users →



Welcome to Covid Commerce

2. ***Home Page:***
   The home page is the main page of store, that is the first page users see when they access the site.

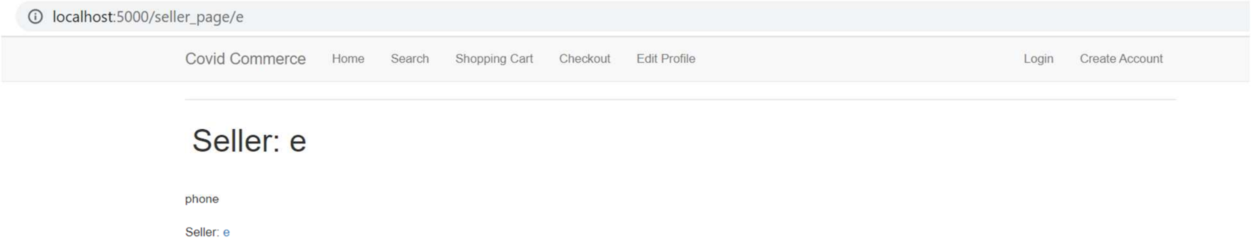

localhost:5000/index

Welcome to Covid Commerce

phone

Seller: e

3. ***Seller Page:***
   Every seller has their own page, in which a list of all their items are displayed. This page can be accessed by selecting the seller name under each item, or entering the URL with the seller username.

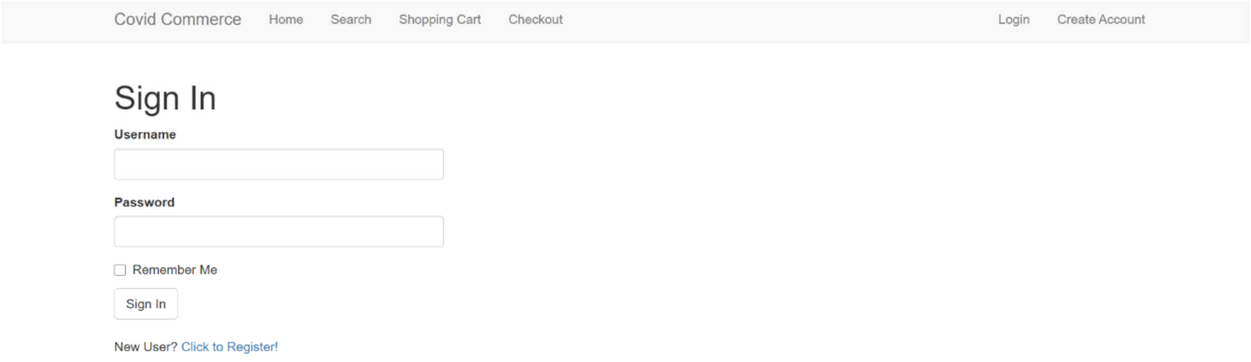

localhost:5000/seller_page/e

Seller: e

phone

Seller: e

4. ***Login Page:***
   The login page is the same for all users, where only a username and password is required.



Sign In

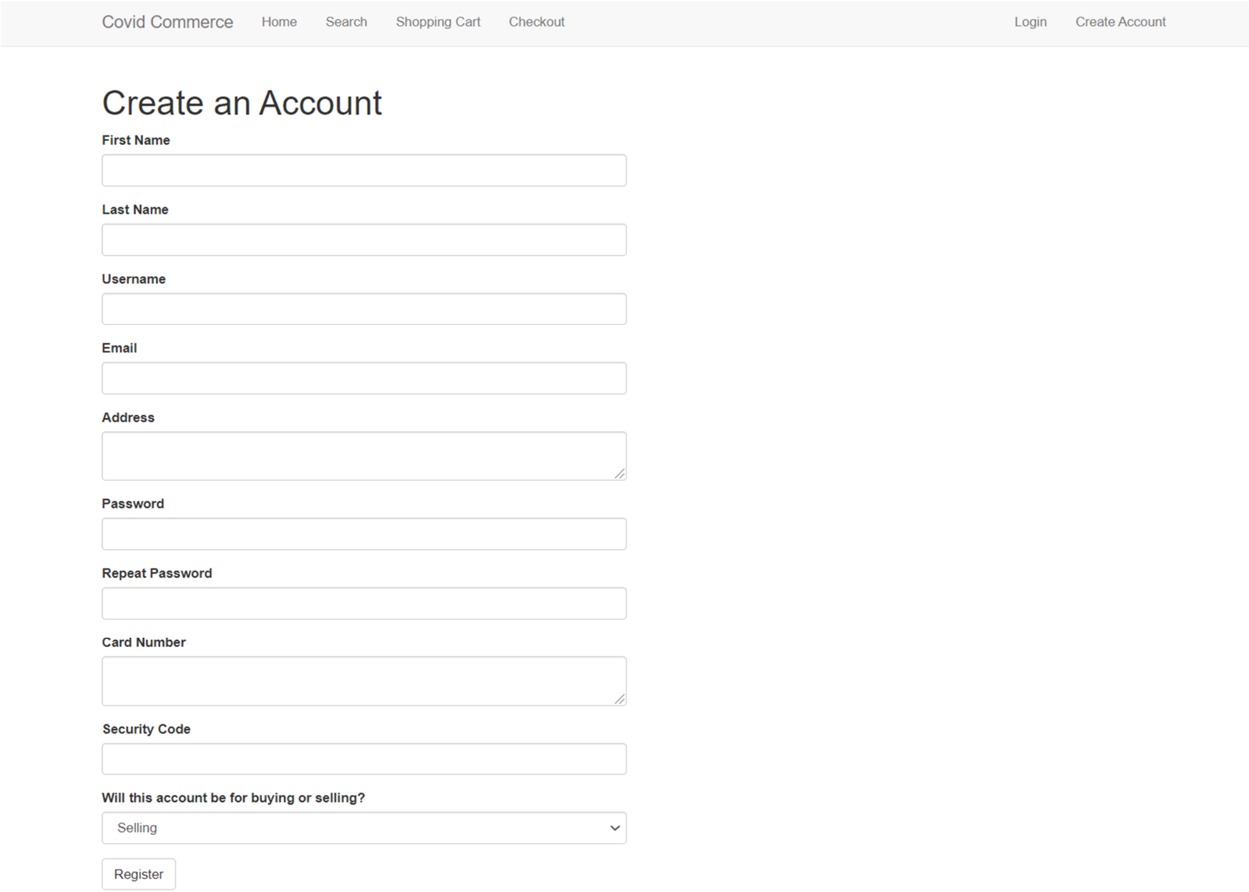
Username

Password

□ Remember Me

New User? Click to Register!

5. ***Register New Users:***
   Users can create a new account by accessing this page, either through the “Create account” option on the nav-bar, or from the login page. The required fields are the same for all users, and the user must select which type (buyer/seller) of account this will be.



Create an Account

First Name

Last Name

Username

Email

Address

Password

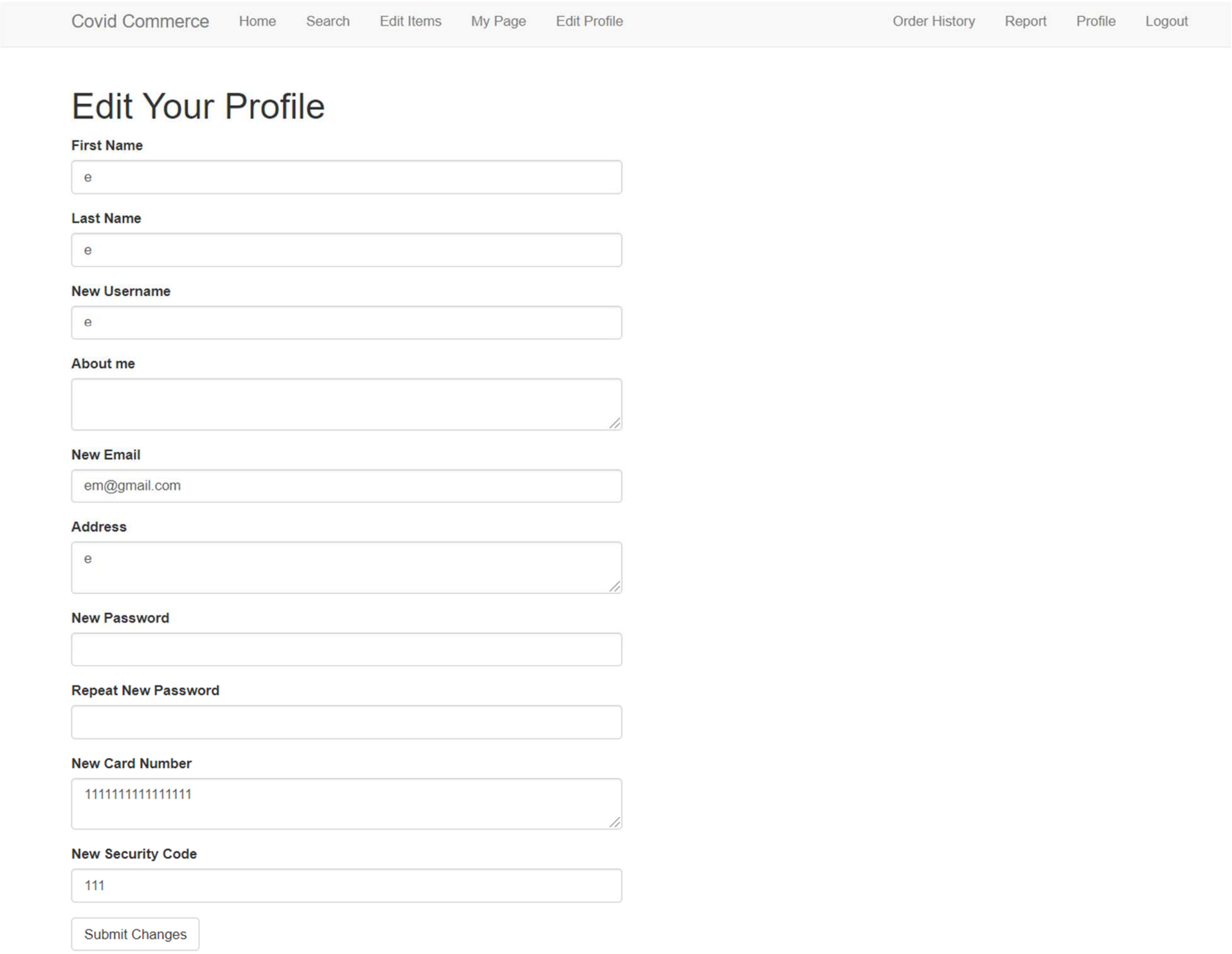
Repeat Password

Card Number

Security Code

Will this account be for buying or selling?
Selling

6. ***Edit User Profile:***
   Users can edit their profile attributes by selecting the Profile option in the navbar. This is available for all registered users (buyers and sellers).



Edit Your Profile

First Name: e

Last Name: e

New Username: e

About me:

New Email: em@gmail.com

Address: e

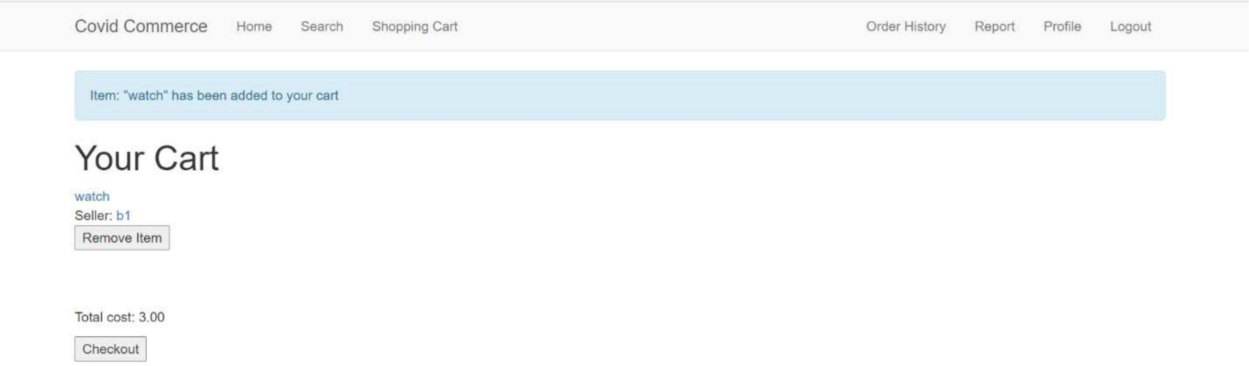
New Password:

Repeat New Password:

New Card Number: 1111111111111111

New Security Code: 111

7. ***Shopping Cart:***
   Users can view and remove the items in their cart



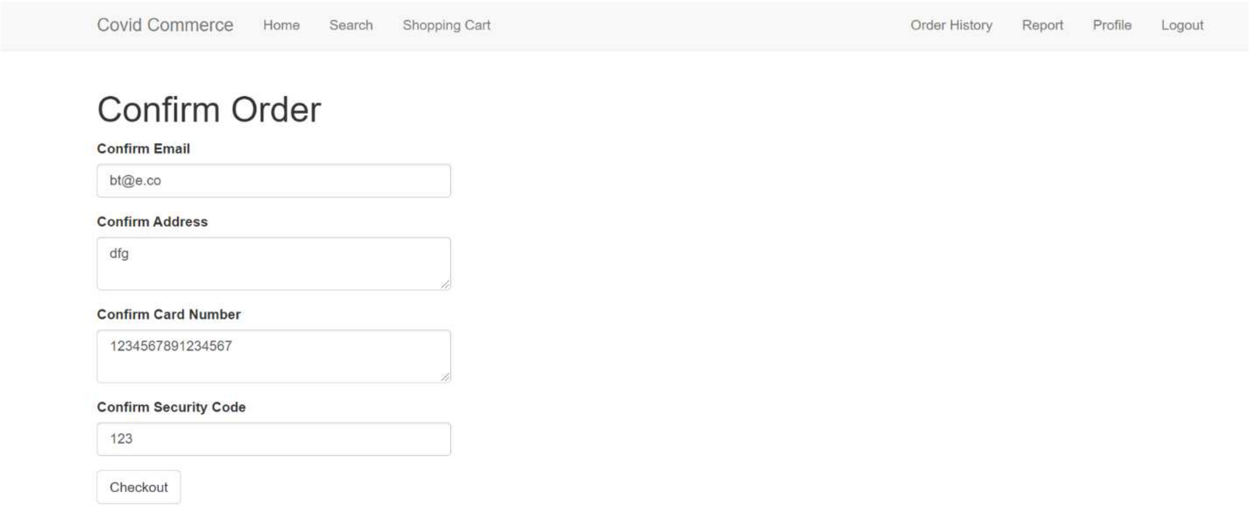
Item: "watch" has been added to your cart

Your Cart

watch
Seller: b1

Total cost: 3.00

8. ***Confirm Order:***
   Before checkout, a confirm order page will be displayed .



Confirm Order

Confirm Email
bt@e.co

Confirm Address
dfg

Confirm Card Number
1234567891234567

Confirm Security Code
123

IV. Progress Report

For iteration one we wanted to get a good base on the users and how the users will interact. We created a navigation bar that allows the user to travel to different parts of the website with ease. For iteration one the navigation bar links that currently have function include Covid Commerce, Home, Login, Create Account, and Logout. Along with the navigation bar we implemented the login, create account, and logout features that allow users to do all those actions, as seen in the user interface design section. Along with these features we created a basic design for the sellers' pages that cannot be accessed for the moment. All these features have been designed with a basic bootstrap styling. In future iterations we will style the pages to be styled better/appealing.

For iteration two we started to implement more of the buyer and seller use cases. Along with adding the additional functionality for the users we changed the navigation bar to make it a more realistic navigation bar. Functions that we included for the seller include posting an item to sell, viewing their order history of which of their items have been sold and when. A feature that both the seller and buyer can access similar to the order history is to get a more detailed look at any particular item up for sale. Features that we added for the buyer include adding a item that is for sale to their personal shopping cart, viewing their order history of the items they bought, having access to their shopping cart so the see what all is in their cart, checkout, and remove item from cart. Similar to iteration one we focused on the functionality of these features and will in the future make them more appealing. Currently they are styled with enough details to get the important aspects of each feature across the board.

V. Testing Plan

Unit tests will be written using the flask unittest framework. Methods being tested will include user registration/creation, item posting/creation, and the manipulation of items through different user carts and orders. These tests will create instances of the Item, Buyer_Order, Seller_Order, and User, then check attributes of the instances using assertions in order to ensure that each one was successfully created. After writing the tests, the tests will be runnable whenever changes are made in order to ensure that no new implementation has interfered with the ability to create an instance of a model.

Some functional testing will occur making use of the flask unittest framework in order to test relationships between different models. Tests will create temporary items, a temporary buyer with a buyer order, and a temporary seller with a seller order, in order to avoid having to reimplement new instances of these objects each time to automate the testing. After the creation of these instances, the tests will attempt to add items to the orders, and then check that the items exist in the orders, and that the orders are properly associated with their respective buyers and sellers (this will be done through assertions using buyer and seller attributes that should match attributes associated with each order), in order to ensure the correct relationships in the database. Other functional testing will occur through the pytest framework, mainly focusing on routes that were not associated with database models. We will use the pytest framework to test the registration, order history, and checkout methods, which largely depend on relationships and loading the correct information from the database, which can be tested by asserting that the correct responses are invoked by certain requests (attempting to purchase a sold out item should result in failure to add the item to cart, attempting to register with a duplicate email should result in failure to register), and by asserting that after the checkout route, the correct items will be found in order history, and the correct suppliers will be found in a previous order. Once again, these tests will be runnable whenever modifications are made to the project in order to test that previous features have not been interfered with when changes are made. This will help to maximize testing without taking a lot of manual testing.

UI testing will occur manually. We will attempt to test each use case of the project, as well as each html of the project, and view the User Interface throughout the use cases for the correct results. The User interface should reflect the information being recorded in the data base, which is tested through both unit testing and functional testing. We will have to review manually whether the html templates properly display the information from the database and all its routes and models.

VI. References

Dutoit Bernd Bruegge and Allen H. Object Oriented Software Engineering Using UML, Patterns and Java, 3rd Edition [Book]. - [s.l.] : Prentice Hall, 2010.

Grinberg Miguel The Flask Mega-Tutorial [Online] // miguelgrinberg.com. - https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world.