

**Faculty of Engineering & Technology**

**Electrical & Computer Engineering Department**

**Linux LaboratoryPython Project – E-commerce System**

**Prepared by:**

Lana Thabit 1200071

Jana Herzallah 1201139

**Instructor**: Dr.Mohammad Jupran

**Assistants**: Eng.Ahed Mafarjeh, Eng.Tariq Zedan

Section: 3,

Date: 9/9/2023

# Abstract

In the world of e-commerce, our project focuses on building a versatile online shopping platform. This platform caters to both administrators and shoppers, offering a range of features designed to streamline the online shopping experience. It includes tools for managing products and users, ensuring a smooth interaction between buyers and sellers. The project is guided by principles of structured programming, making it user-friendly and efficient while adhering to best practices for input validation

**Table of Contents**

Table of Contents

[Abstract 2](#_Toc145145919)

[Procedures 8](#_Toc145145920)

[1. Load Products from "products.txt" 10](#_Toc145145921)

[2. Load Users from "users.txt" 12](#_Toc145145922)

[3. Validate User ID and Print Access Denied Messages 20](#_Toc145145923)

[4. Create system main menu 24](#_Toc145145924)

[1.Add Product (Admin-Only): 26](#_Toc145145928)

[2.Place an Item on Sale (Admin-Only): 30](#_Toc145145931)

[3.Update Product (Admin-Only): 34](#_Toc145145932)

[4.Add a New User (Admin-Only): 40](#_Toc145145933)

[5.Update User (Admin-Only): 44](#_Toc145145934)

[6.Display All Users (Admin-Only): 49](#_Toc145145937)

[7.List Products (Admin and Shopper): 51](#_Toc145145938)

[8.List Shoppers (Admin-Only): 55](#_Toc145145942)

[9.Add Product to the Basket (Shopper-Only): 58](#_Toc145145944)

[10.Display Basket (Shopper-Only): 62](#_Toc145145947)

[11.Update Basket (Shopper-Only): 65](#_Toc145145949)

[12.Place Order (Shopper-Only): 69](#_Toc145145953)

[13.Execute Order (Admin-Only): 72](#_Toc145145956)

[14.Save Products to a File (Admin-Only): 80](#_Toc145145962)

[15.Save Users to a Text File (Admin-Only): 83](#_Toc145145965)

[16.Exit: 86](#_Toc145145967)

[Concusion 87](#_Toc145145920)

**Table of figures**

[Figure 1:Product class 9](#_Toc145145778)

[Figure 2:User class 9](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145779)

[Figure 3:ECommerceSystem class 10](#_Toc145145780)

[Figure 4:EcomeerceModule module 10](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145781)

[Figure 5:declare list of products objects 11](#_Toc145145782)

[Figure 6:declare read\_products\_file to fill the product list 12](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145783)

[Figure 7:invoke read\_users\_file to fill our products list 13](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145784)

[Figure 8:declare list of users objects 13](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145785)

[Figure 9:declare read\_products\_file to fill the users list 14](#_Toc145145786)

[Figure 10: filling the users baskets 15](#_Toc145145787)

[Figure 11:display products function 15](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145788)

[Figure 12::display users function 16](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145789)

[Figure 13: products.txt file inputs 17](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145790)

[Figure 14:product list output on compiler 18](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145791)

[Figure 15:users.txt file inputs 19](#_Toc145145792)

[Figure 16:user list output on compiler 20](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145793)

[Figure 17: print welcome statment for user(admin/shopper) 21](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145794)

[Figure 18:Access control of users 22](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145795)

[Figure 19:print welcome message for user (admin) 22](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145796)

[Figure 20:available admin choice 23](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145797)

[Figure 21:unavailable admin choice 23](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145798)

[Figure 22:print welcome message for user (shopper) 24](#_Toc145145799)

[Figure 23:available shopper choice 24](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145800)

[Figure 24:unavailable shopper choice 25](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145801)

[Figure 25:display main menu 26](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145802)

[Figure 26: function to display menu for multiple times 26](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145803)

[Figure 27:choice 1 handling 27](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145804)

[Figure 28:add product function 28](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145805)

[Figure 29:choice 2 handling 31](#_Toc145145806)

[Figure 30:place product on sale function 33](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145807)

[Figure 31:when product is on sale 33](#_Toc145145808)

[Figure 32:when product is not on sale 34](#_Toc145145809)

[Figure 33:choice 3 handling 36](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145810)

[Figure 34:Update\_product function 38](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145811)

[Figure 35:update product example 39](#_Toc145145812)

[Figure 36: update product output 40](#_Toc145145813)

[Figure 37:choice 4 handling 41](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145814)

[Figure 38:add user function 42](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145815)

[Figure 39: validate user id function 43](#_Toc145145816)

[Figure 40:validate role function 43](#_Toc145145817)

[Figure 41:validate active function 43](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145818)

[Figure 42:before add a user 44](#_Toc145145819)

[Figure 43: after user added 44](#_Toc145145820)

[Figure 44:choice 5 handling 46](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145821)

[Figure 45: update user function 47](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145822)

[Figure 46update user file function 47](#_Toc145145823)

[Figure 47:output of update user function 48](#_Toc145145824)

[Figure 48:choice 6 handling 50](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145825)

[Figure 49:display users function 50](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145826)

[Figure 50:display users output 51](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145827)

[Figure 51: list products function 53](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145828)

[Figure 52:choice 7 handling 53](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145829)

[Figure 53:other functions used to display products 54](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145830)

[Figure 54:list products output 55](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145831)

[Figure 55:choice 8 handling 56](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145832)

[Figure 56: List all shoppers function 57](#_Toc145145833)

[Figure 57:other functions used in display shoppers function 57](#_Toc145145834)

[Figure 58:List shoppers output 58](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145835)

[Figure 59:choice 9 handling 60](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145836)

[Figure 60:add to basket function 61](#_Toc145145837)

[Figure 61:add to basket output 62](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145838)

[Figure 62:choice 10 handling 63](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145839)

[Figure 63:display basket function 64](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145840)

[Figure 64:display basket output. 65](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145841)

[Figure 65:choice 11 handling 66](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145842)

[Figure 66:update basket function 68](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145843)

[Figure 67:update basket output 69](#_Toc145145844)

[Figure 68:choice 12 handling 70](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145845)

[Figure 69:place order function 71](#_Toc145145846)

[Figure 70:place order output 72](#_Toc145145847)

[Figure 71:choice 13 handling 74](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145848)

[Figure 72:execute order function 76](#_Toc145145849)

[Figure 73:check notification function 77](#_Toc145145850)

[Figure 74:the file before execute the order 78](#_Toc145145851)

[Figure 75:the file after execute the order 78](#_Toc145145852)

[Figure 76:unaccepted order 79](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145853)

[Figure 77:displaying shopper notification 80](#_Toc145145854)

[Figure 78:choice 14 handling 81](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145855)

[Figure 79:save products to a file function 82](#_Toc145145856)

[Figure 80: save products to file output 83](#_Toc145145857)

[Figure 81:choice 15 handling 84](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145858)

[Figure 82:choice 15 handling 85](#_Toc145145859)

[Figure 83:save users to file function 85](#_Toc145145860)

[Figure 84:save users to file output 86](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145861)

[Figure 85::choice 16 handling 87](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145862)

[Figure 86:Exiting program output 87](file:///C:\Users\ss\Desktop\Linuxr.docx#_Toc145145863)

**Theory**This comprehensive E-commerce System project aims to develop a robust online shopping platform, catering to the diverse needs of both administrators and shoppers. The project encompasses a wide array of features and functionalities, including product and user management, shopping cart operations, order processing, and data persistence. It adheres to the principles of object-oriented programming (OOP), modular code organization, and meticulous input validation to ensure an efficient, user-friendly, and maintainable system.

The core aspects of this project can be summarized as follows:

User Roles and Authentication: The system differentiates between two distinct user roles: administrators, responsible for managing products and users, and shoppers, who engage in the shopping experience. User authentication is enforced through unique user IDs, and unauthorized access to admin or shopper-specific functions is prevented with clear 'Access Denied' messages.

Data Loading and Validation: At system launch, data is loaded from external text files ("products.txt" and "users.txt"). The system meticulously validates this data, ensuring its accuracy and integrity. Any invalid input, be it regarding dates, integers, or strings, prompts clear error messages for user feedback.

Product Management: Admins can perform various product-related tasks, such as adding new products, placing items on sale with discounted prices and expiration dates, and updating existing product details. These actions are carried out using intuitive input forms.

User Management: Admins are equipped to manage users, adding new members to the system and updating their information. For each user, additional attributes like the shopping basket (a dictionary of selected products and quantities) and order status are maintained, allowing for seamless shopping experiences.

User and Product Listing: Both admins and shoppers can list products based on various criteria, including all available products, items on sale, specific product categories, and product names. Admins can also list shoppers based on criteria like having items in their basket or having unprocessed orders.

Shopping Cart Operations: Shoppers can add products to their shopping baskets, view the contents of their baskets, and perform actions such as clearing the basket, removing specific items, or updating item quantities.

Order Processing: Shoppers can initiate the order process, while admins have the ability to execute these orders. Executing an order reduces product inventory accordingly and clears items from the shopper's basket.

Data Persistence: Admins can save both product and user data to external text files, offering a form of data persistence.

Exit Option: The system provides a clean exit option for both admins and shoppers, ensuring that the user is prompted to save any unsaved data before terminating the software.

Throughout the project, an emphasis is placed on clean and well-structured code, meaningful variable names, and extensive comments for code clarity. Modularity is achieved through the use of functions and OOP concepts, allowing for easier code maintenance and scalability. This E-commerce System project endeavors to create a user-friendly, efficient, and versatile online shopping platform that aligns seamlessly with the evolving landscape of e-commerce.

# Procedures

To start our system, we've built a strong foundation using three important classes and one special module. These components work together to make our project do all the cool stuff it can do. The Product class keeps track of product details and helps us add, change, and show products in our system. The User class takes care of things related to users, like making and updating user profiles, and handling shopping baskets and orders. We also have the E-commerce System class, which acts like the project's manager. It handles tasks like reading data from files, checking if users are allowed to do certain things, and doing important actions like putting products on sale and processing orders. To make sure our project can work with files properly, we've added a special module (the Ecommerce module) that's really good at handling files. All of these pieces fit together to make sure our E-commerce System works smoothly and does all the things it's supposed to do.

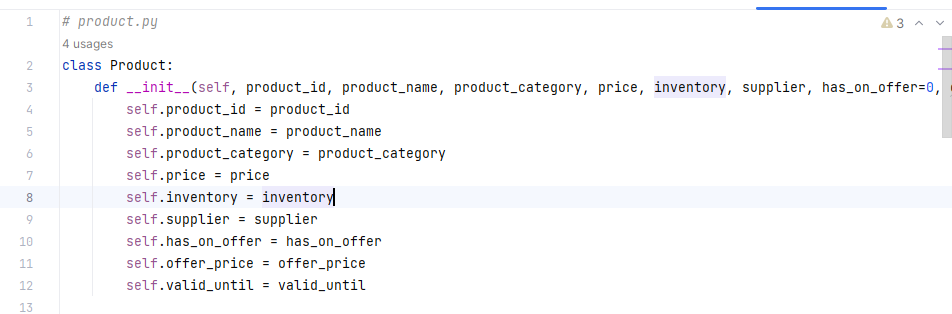


Figure 1:Product class

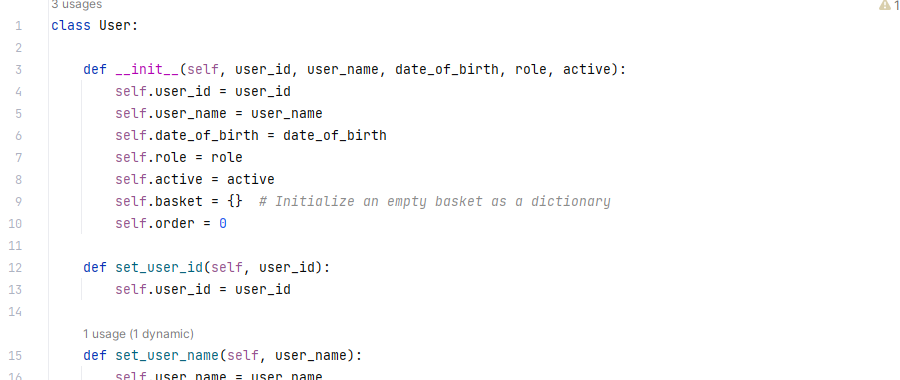


Figure 2:User class

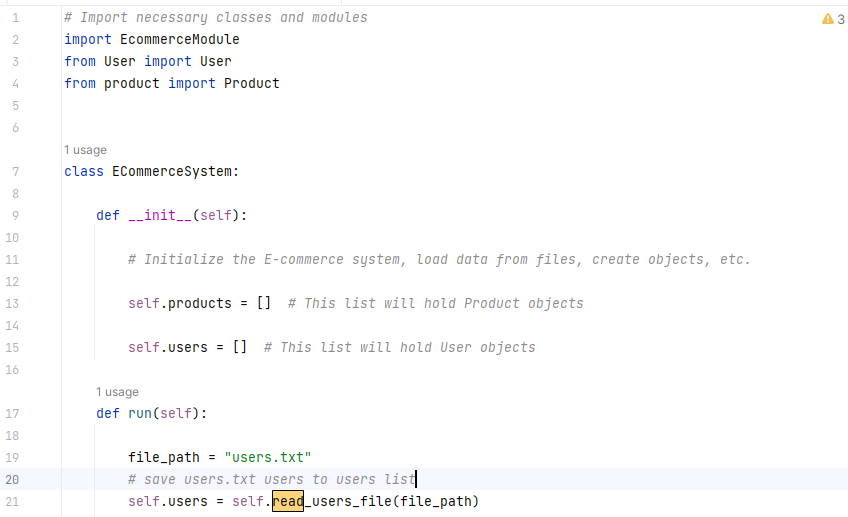


Figure 3:ECommerceSystem class

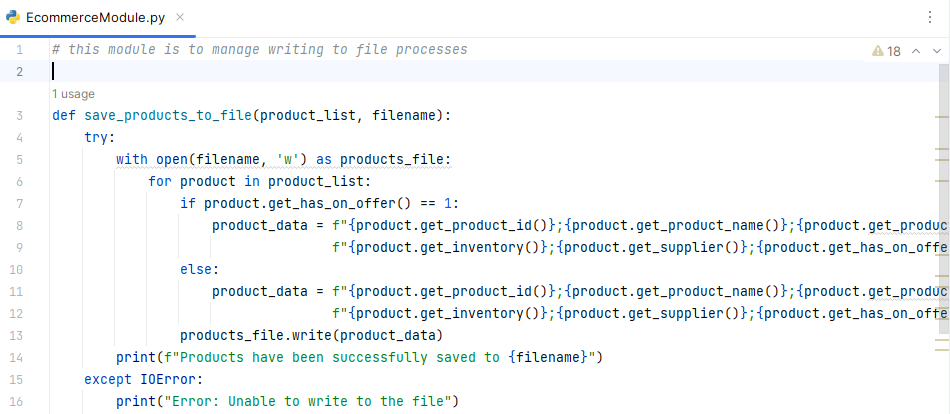


Figure 4:EcomeerceModule module

At first we began our project by starting the task of importing data from external files. Our initial objective was to extract information about the products from the 'products.txt' file carefully checking their structure for accuracy. At the time we obtained user data from 'users.txt' making sure it matched our predefined format perfectly. These first steps formed a base on which we built the functionality of our Ecommerce System.

Initial Steps for System Launch:

## Load Products from "products.txt"

To load products from the "products.txt" file, we followed a careful step-by-step approach. We created an empty list to hold the product information. Then, we opened the file and read its content line by line. For each line, we checked if it had enough information (at least seven pieces) and if it did, we extracted important details like product ID, name, category, price, how many were in stock, who supplied it, and if it was on sale. If it was on sale, we also noted the sale price and how long the sale lasted.

We used all this information to create a special "Product" object that represented the product, and we added this object to our list of products. If any line in the file didn't have the right information, we let the user know with an error message. This careful process ensured that our E-commerce System got all the correct details it needed from "products.txt" so it could manage products effectively.

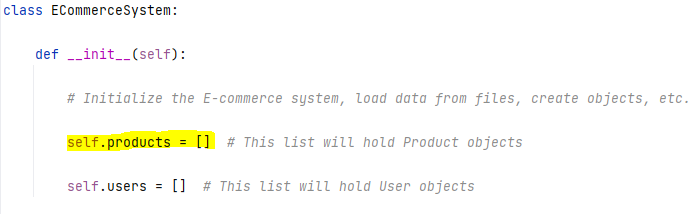


Figure 5:declare list of products objects

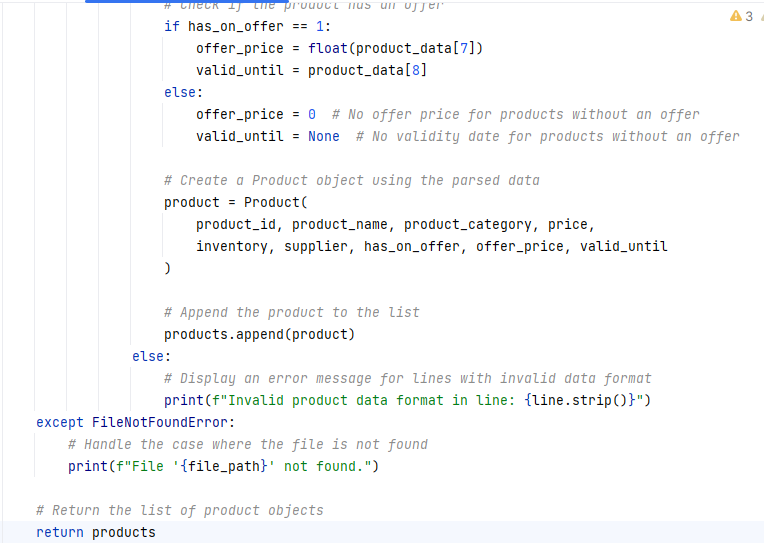


Figure 6:declare read\_products\_file to fill the product list

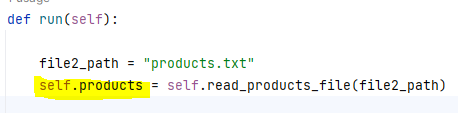


Figure 7:invoke read\_users\_file to fill our products list

NOTE: The run function includes our lists (products,users) and the project minu

## Load Users from "users.txt"

In the process of getting user data from the "users.txt" file, we took careful steps. First, we made an empty list to keep track of user information in our system. Then, we opened the file and looked at what was inside line by line.

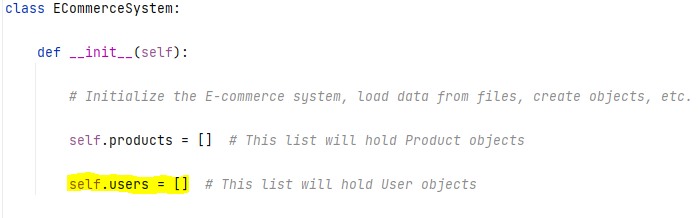
For each line, we made sure it had exactly seven pieces of information, just like we wanted. These pieces included things like the user's ID, name, birthdate, whether they were an admin or a shopper, whether they were active, what they had in their shopping basket, and if they placed an order.

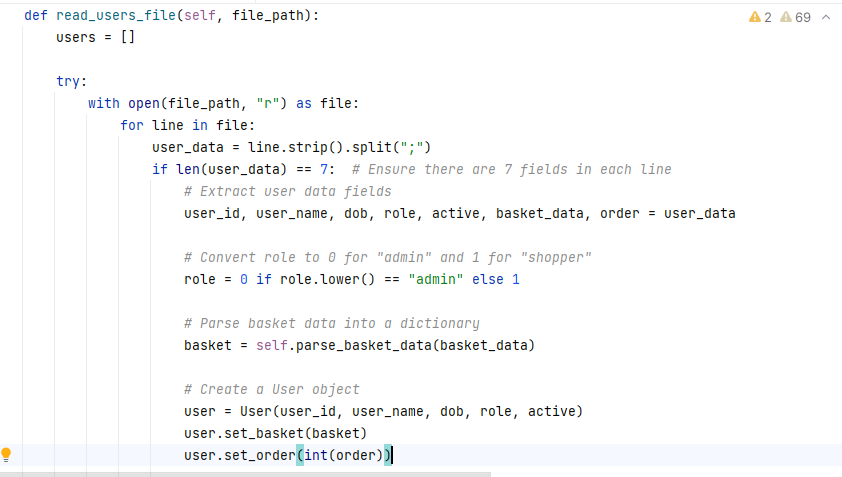
We paid special attention to whether a user was an admin or a shopper and turned that into a number. Admins became 0, and shoppers became 1.

We also handled the shopping basket data, which was organized in a specific way. We changed this data into a dictionary so we could understand what products users had in their baskets.

Then, we made User objects with all this information for each user and put these objects in our list. If we found any problems with how the data was written, we told the user with an error message.

Figure 8:declare list of users objects





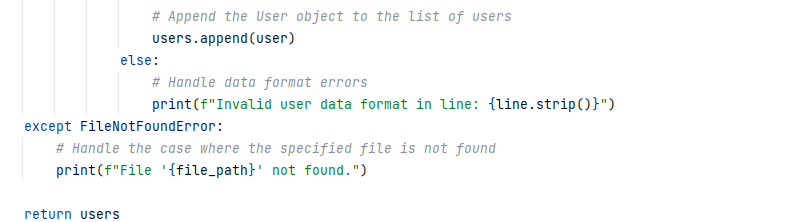


Figure 9:declare read\_products\_file to fill the users list

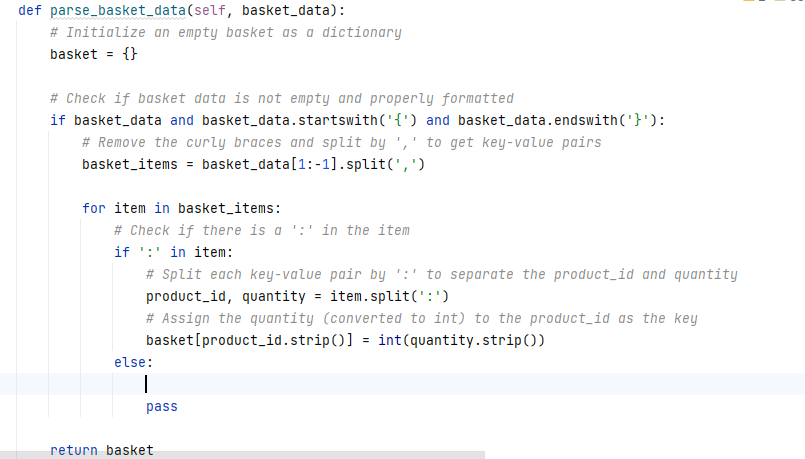


Figure 10: filling the users baskets

* The outputs of users and products loading codes:

We build two functions to display the outputs of the files lists.

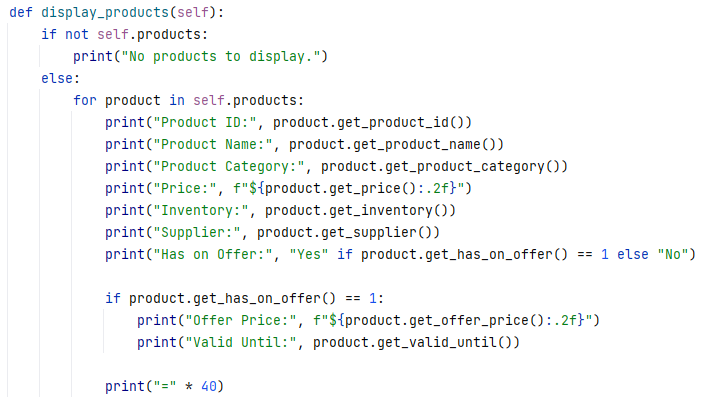


Figure 11:display products function

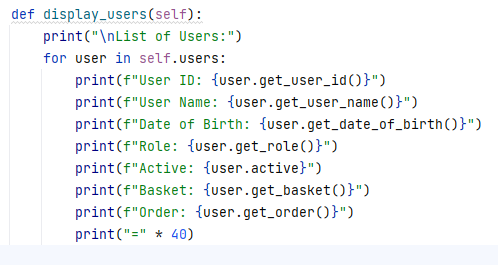


Figure 12::display users function

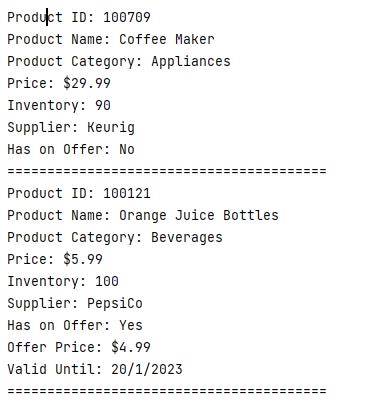
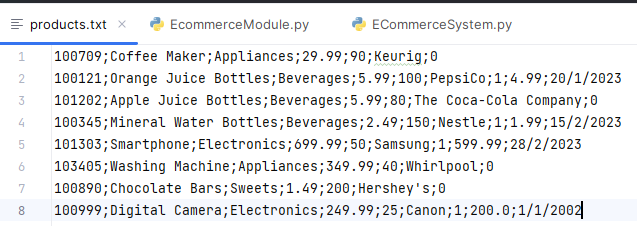
 

Figure 13: products.txt file inputs



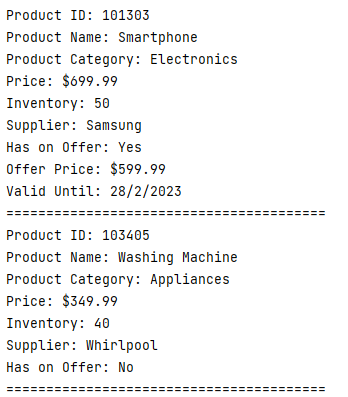
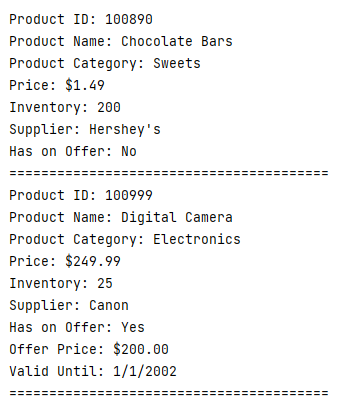
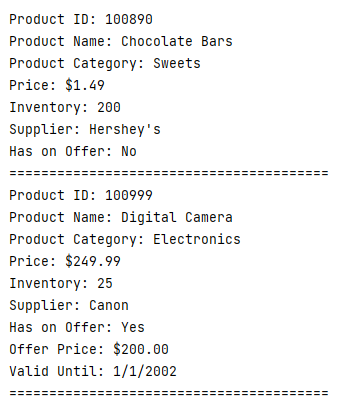


Figure 14:product list output on compiler

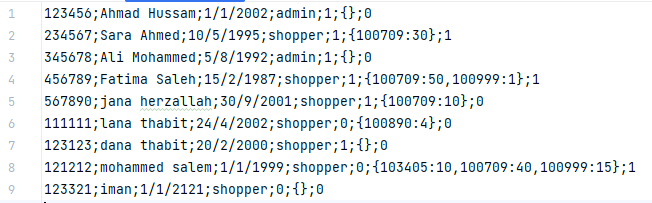
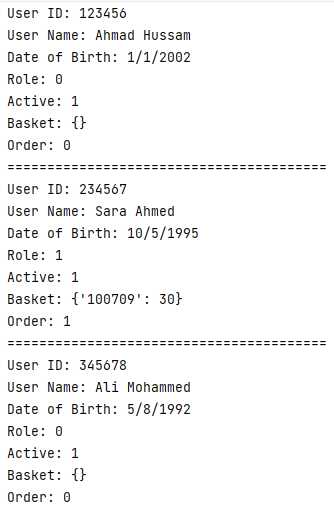


Figure 15:users.txt file inputs





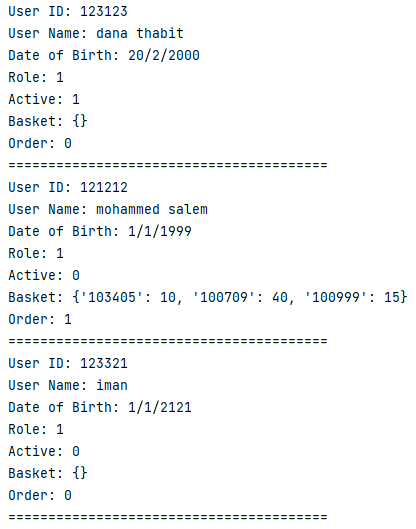


Figure 16:user list output on compiler

## Validate User ID and Print Access Denied Messages

the code belwo was responsible for user authentication and access control within a program. It initiated by requesting the user to input their user ID, which was used for user validation. The code then attempted to find a user with the provided ID using the self.find\_user\_by\_id(user\_id) method. If a matching user was found, the user's role and name were retrieved. However, if no user with the given ID was found, the user variable remained None, signifying an invalid or non-existent user.

Access control was implemented based on the user's role. If the user was identified as a shopper (role 1), they were restricted to accessing specific menu options (options 7-12,16). Any attempt to select an unauthorized option resulted in an "Access Denied" message being printed, and the program continued to prompt the user for input.

Conversely, if the user was an administrator (role 0), they had a different set of permitted menu options (options 1-6 and 13-16). Similar to shoppers, administrators also received an "Access Denied" message if they attempted to access an unauthorized option.

The access control logic ensured that users could only interact with features and options appropriate for their assigned role, maintaining security and data integrity within the program.



Figure 17: print welcome statment for user(admin/shopper)

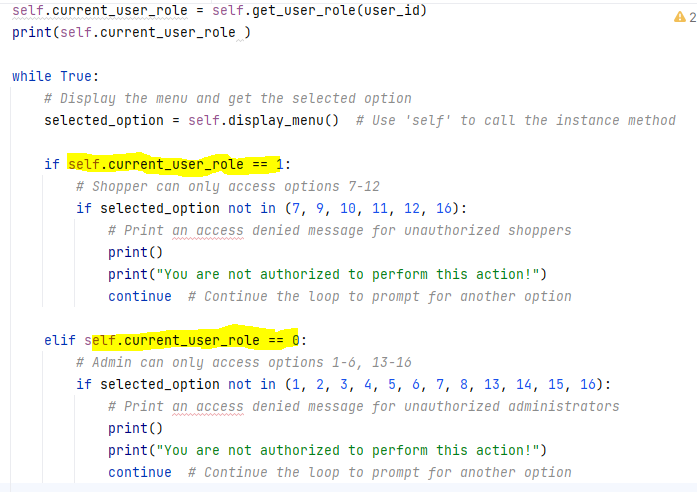


Figure 18:Access control of users

* The outputs of Validate User ID and Print Access Denied Messages:

1. Login as an admin:

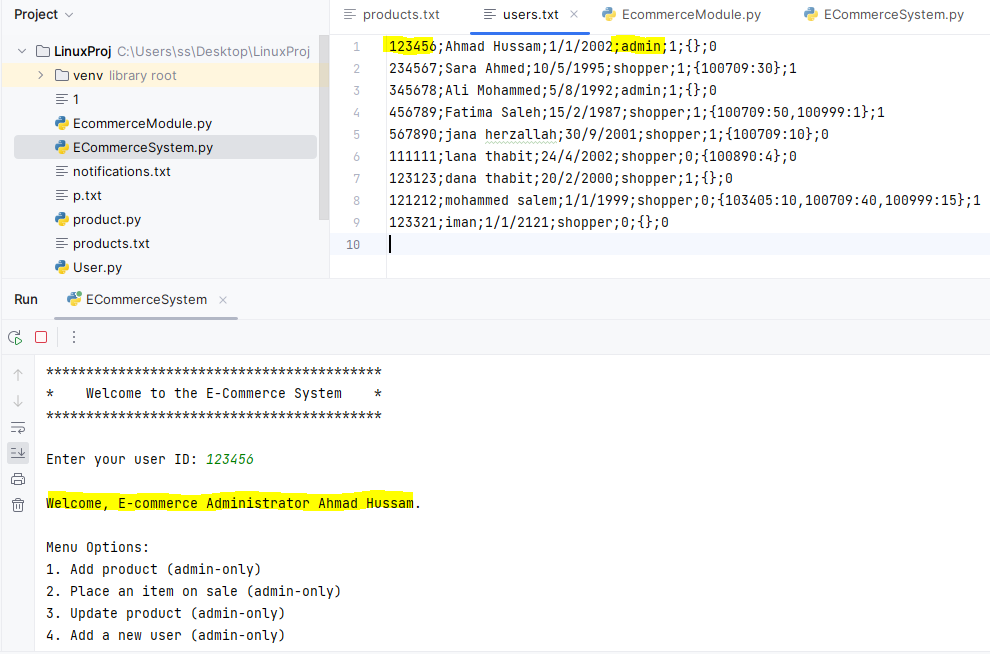


Figure 19:print welcome message for user (admin)

when the admin choose from his choices we let him to do it normally

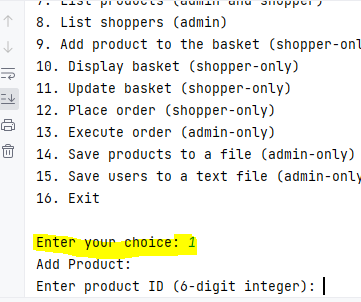


Figure 20:available admin choice

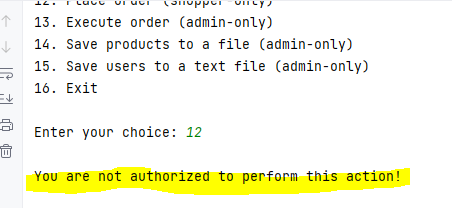
 But if admin choose from shopper choices then access denied message will display.

Figure 21:unavailable admin choice

2.Login as shopper:

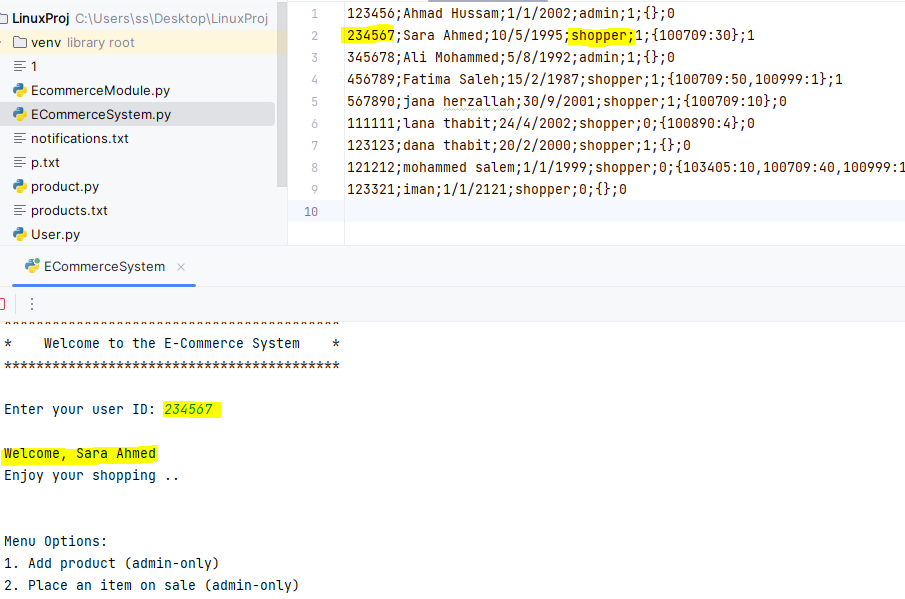


Figure 22:print welcome message for user (shopper)

when the shopper choose from his choices we let him/her to do it normally

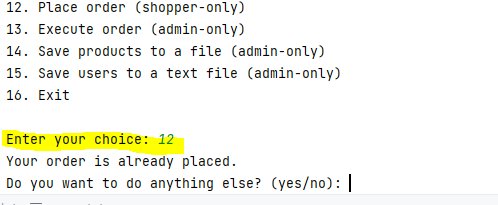


Figure 23:available shopper choice

But if shopper choose from shopper choices then access denied message will display.

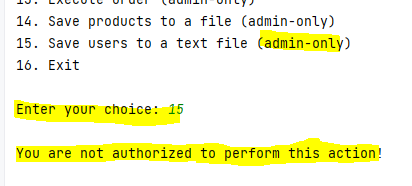


Figure 24:unavailable shopper choice

## Create system main menu

The system's main menu serves as the central control panel for our E-commerce System project, orchestrating a diverse range of functionalities designed to enhance the online shopping experience for both administrators and shoppers. The main menu is a user-friendly interface that embodies principles of structured programming, ensuring efficiency and input validation adherence.

Menu options:

Here we set up an important process to show the main menu and collect user input. We used a loop to keep showing the menu options to the user, making it easy for them to interact with our E-commerce System.

While doing this, we made sure that the system could handle mistakes from the user. We built a system to catch any problems that might come up, especially when the user needed to type a number. This way, we could nicely handle issues like when the user didn't type a number or picked an option that wasn't on the menu.

We also made sure that the user could only pick options from 1 to 16. This helped prevent mistakes and made sure that the user had a smooth experience while using our system.

### 

### 

### 

Figure 25:display main menu

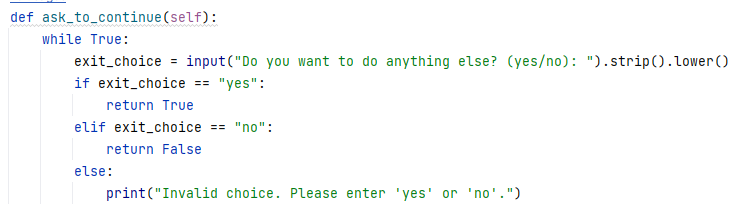
We also wrote a function for menu choices that asks users after each choice if they want to complete (display menu again) or not :

Figure 26: function to display menu for multiple times

### 1.Add Product (Admin-Only):

In this section, we've implemented a user-friendly product addition feature. Users are prompted to enter essential product details, and rigorous validation ensures data accuracy. If a product's ID is not a unique 6-digit integer, or other input errors occur, clear error messages guide the user. Once validated, a Product object is created, capturing all relevant attributes. The product is then added to the system's catalog and saved in an external text file, 'products.txt,' (the adding occured on products list and also on the products.txt file ) ensuring data persistence and efficient product management.

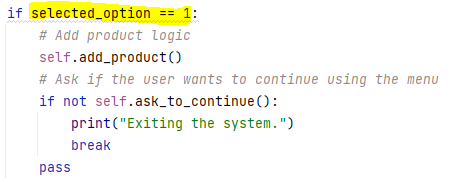
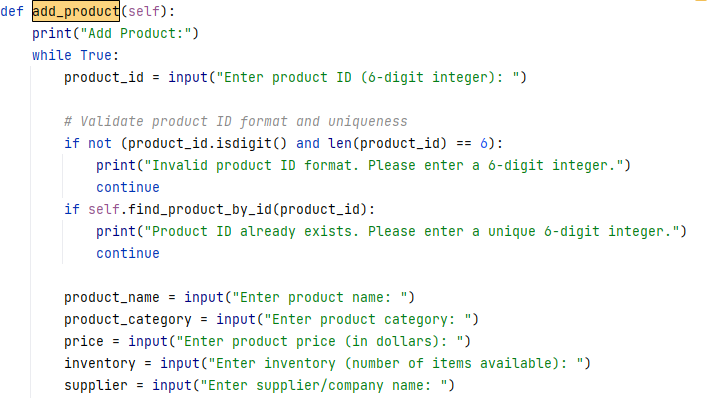
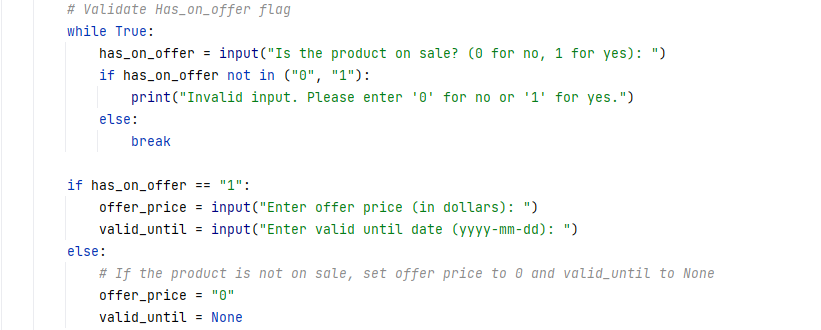


Figure 27:choice 1 handling





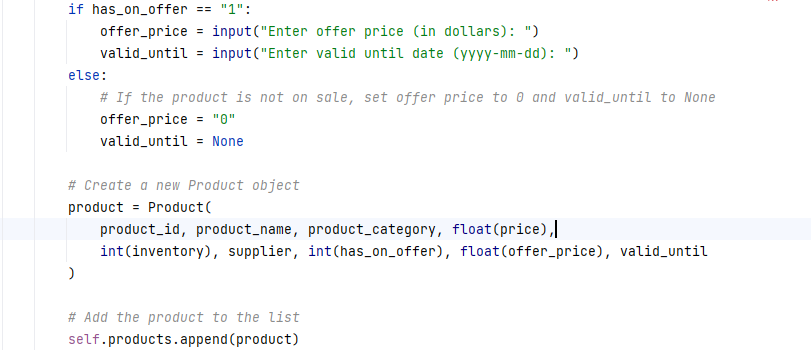
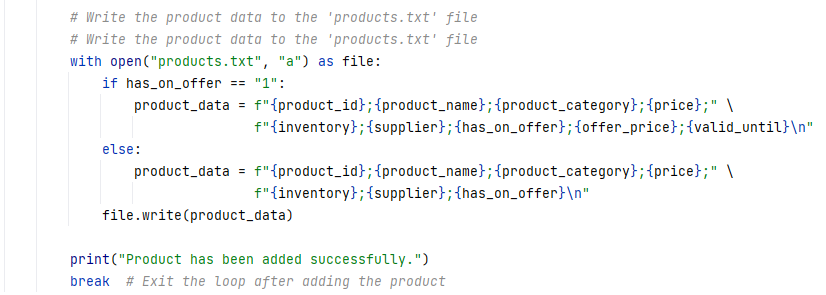
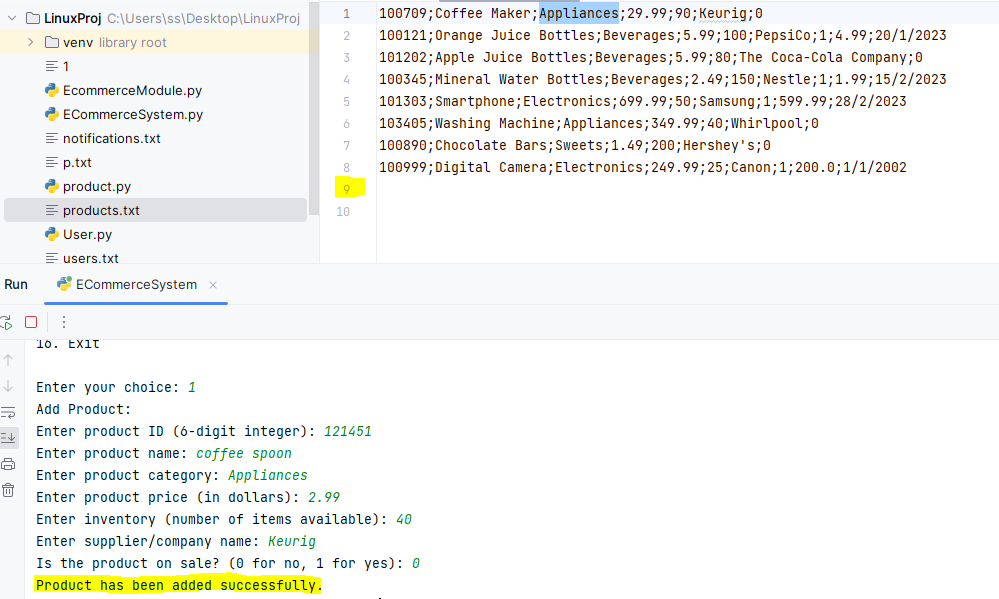
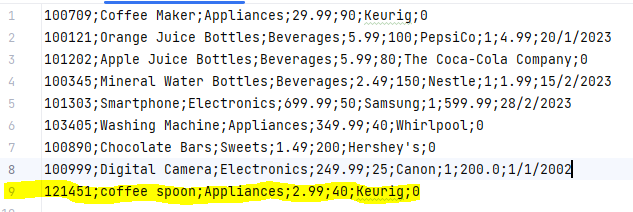


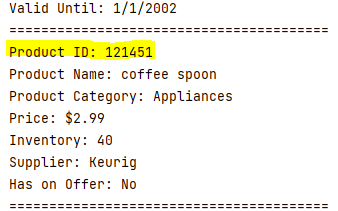
Figure 28:add product function

Output:



The file updated succssesfuly:

The list also updated succssesfuly:



2.Place an Item on Sale (Admin-Only):

In this section, we have implemented a feature that allows the user to place an item on sale within the system. This feature ensures user-friendliness by guiding users through the process and providing rigorous validation to ensure data accuracy.

Upon selecting the option to place an item on sale, the system prompts the user to enter the product's ID, which must be a unique 6-digit integer. Rigorous validation checks are applied to ensure the entered ID format meets this requirement. If any input errors occur, clear error messages guide the user until a valid ID is provided.

Next, the system checks if the entered product ID exists in the catalog (product list). If the product does not exist, the system offers the option to cancel or try again with a different ID, ensuring flexibility for the user's needs.

If the product exists and is not already on sale, the system proceeds to set the "Has\_on\_offer" flag for the product to indicate that it's now on sale. The user is prompted to enter the offer price, with further validation to ensure a valid numeric value is provided.

After successfully setting the offer price, the system asks for the valid until date in the format 'yyyy-mm-dd.' While further validation for date format is recommended, it is not shown here in the code snippet.

Once all necessary information is collected, the product's data is updated in the system's catalog. This includes setting the "Has\_on\_offer" flag, updating the offer price, and setting the valid until date.

Finally, the updated product data is written back to an external text file named 'products.txt,' ensuring data persistence and efficient product management. This file contains all relevant attributes of each product, including whether it's on sale, the offer price, and the valid until date.

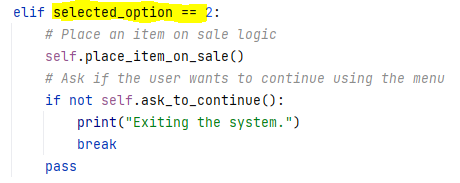
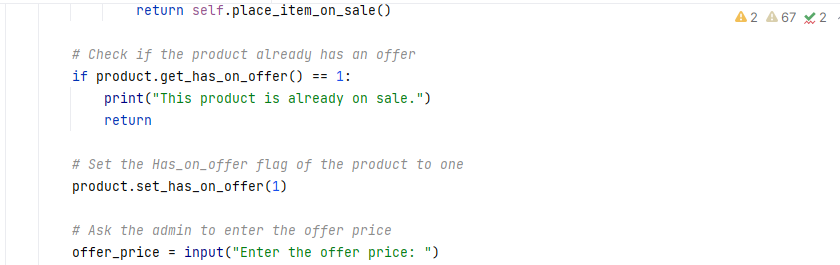
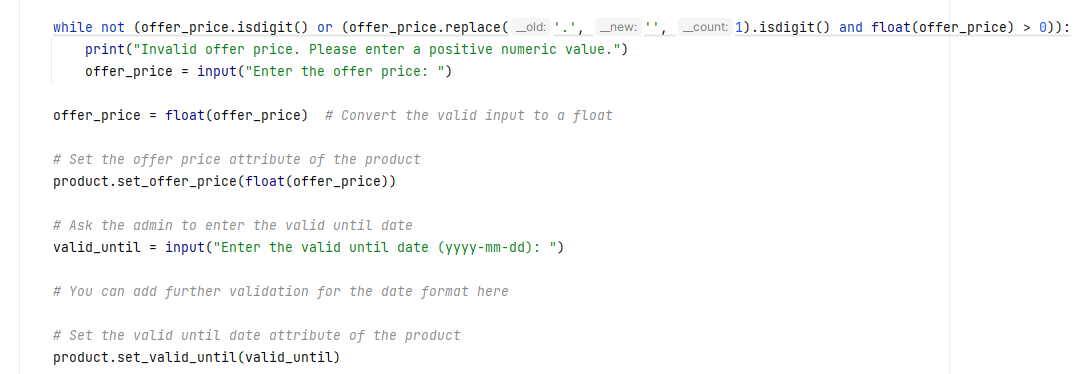


Figure 29:choice 2 handling





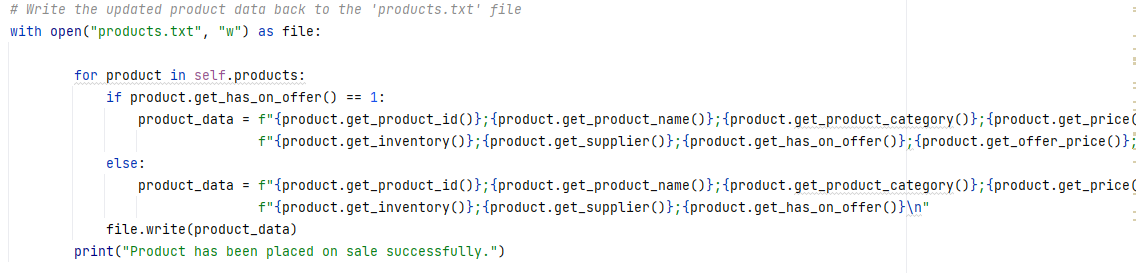


Figure 30:place product on sale function

Output :

1. If the product already on sale :

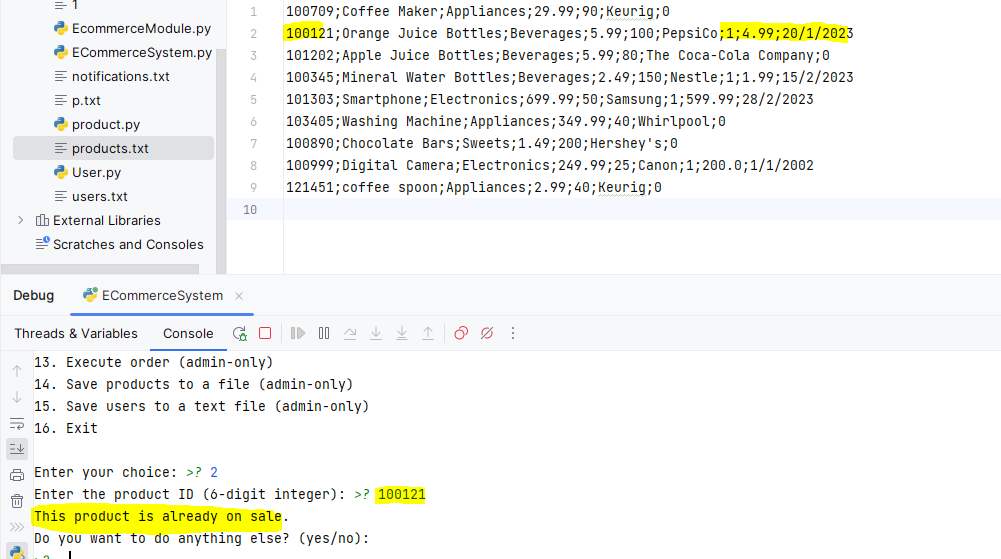


Figure 31:when product is on sale

1. If the product not on sale :

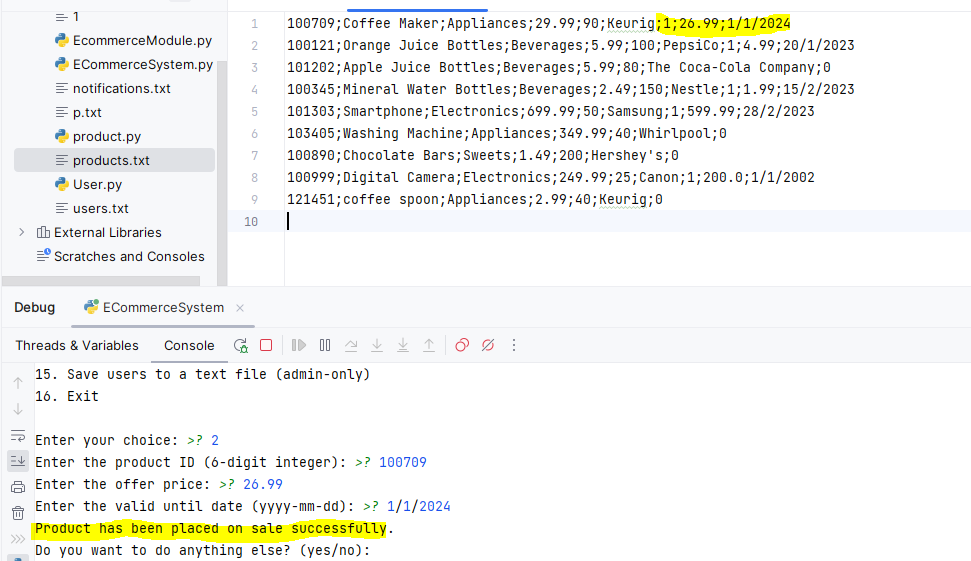


Figure 32:when product is not on sale

### 3.Update Product (Admin-Only):

In this section, we have implemented a feature that allows the user to update the attributes of an existing product within the system. This feature provides a user-friendly interface for updating product information, ensuring data accuracy and flexibility.

The process begins by prompting the user to enter the product ID of the item they want to update. The system checks whether the product with the provided ID exists in the catalog. If the product does not exist, an appropriate message is displayed, and the update process is terminated.

Once a valid product is selected, the user is presented with a menu of attribute choices to update. The available choices include updating the product name, product category, price, inventory, supplier, "Has on Offer" flag, offer price (if the product is on sale), valid until date (if the product is on sale), and finishing the update process.

Each attribute update choice is followed by a series of input validations to ensure the integrity of the data. Here's a breakdown of each attribute update:

1.Product Name: The user is prompted to enter a new product name, and the product's name is updated accordingly.

2.Product Category: The user can input a new product category, and the product's category is updated.

3.Price: The user is prompted to input a new price (in dollars), with validation to ensure a valid numeric non-negative value.

4.Inventory: The user can input a new inventory value (number of items available), with validation to ensure it's a valid non-negative integer.

5.Supplier: The user can input a new supplier or company name, and the product's supplier is updated.

6.Has on Offer: The user can indicate whether the product is on sale (0 for no, 1 for yes). If set to 0, the offer price and valid until date are set to default values (0 and None, respectively). If set to 1, the user is prompted to enter the offer price and valid until date, with appropriate validations.

7.Offer Price (if on offer): If the product is on sale, the user can input a new offer price.

8.Valid Until Date (if on offer): If the product is on sale, the user can input a new valid until date in the 'yyyy-mm-dd' format.

9.Finish Updating: This option allows the user to exit the update process.

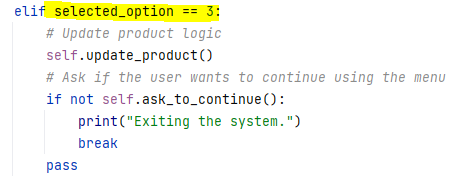
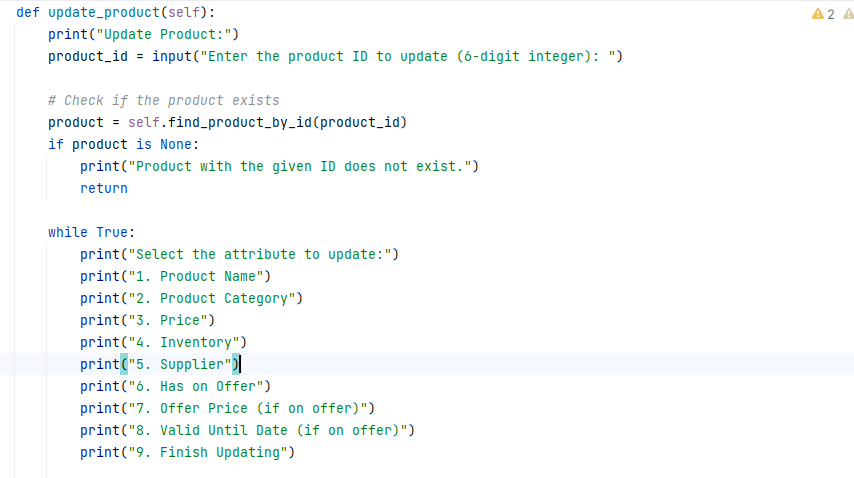
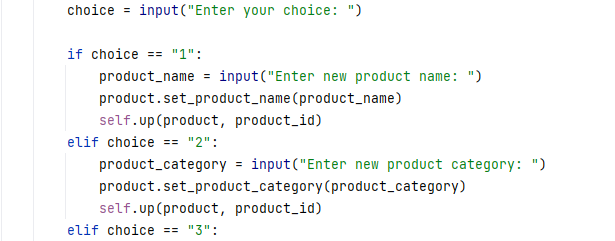
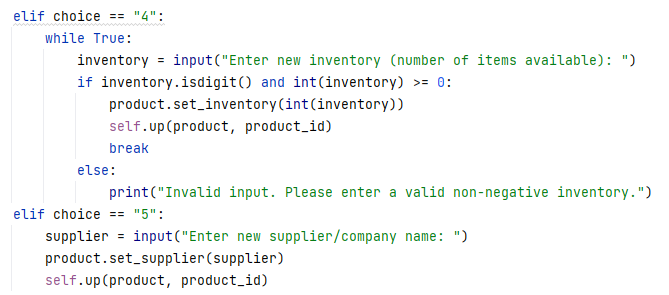
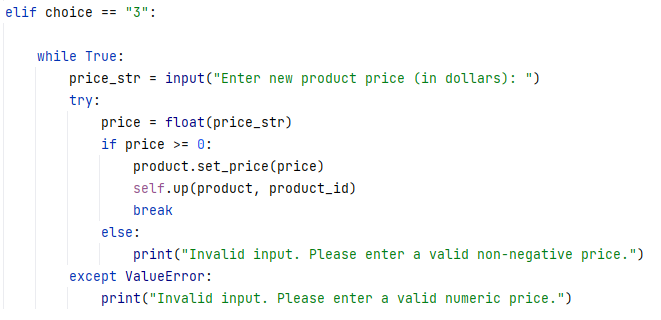
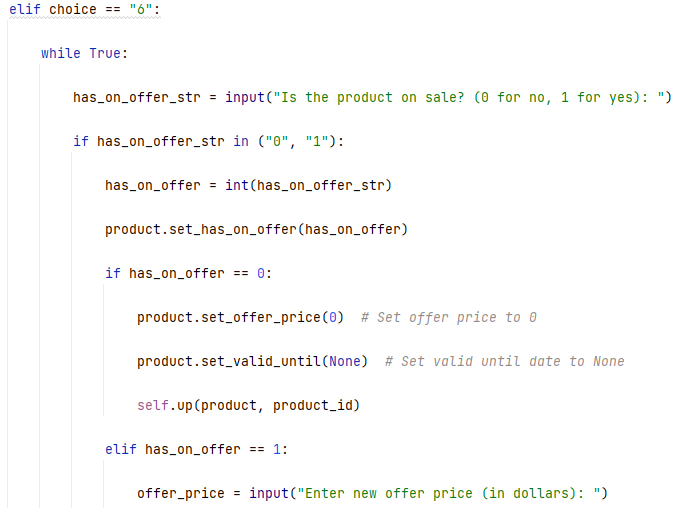


Figure 33:choice 3 handling





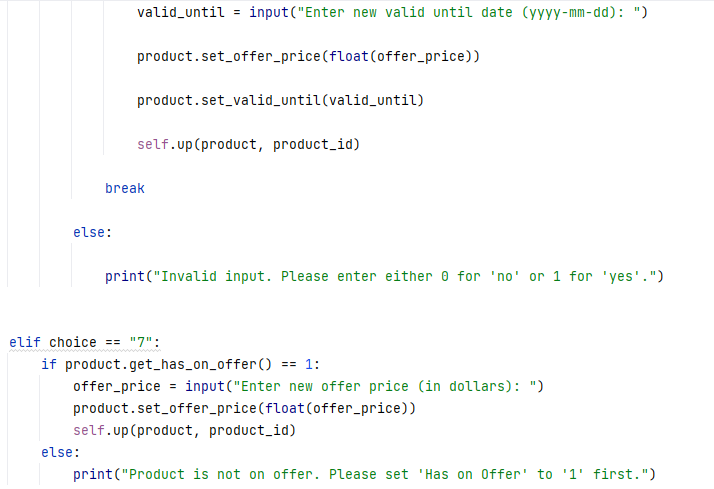
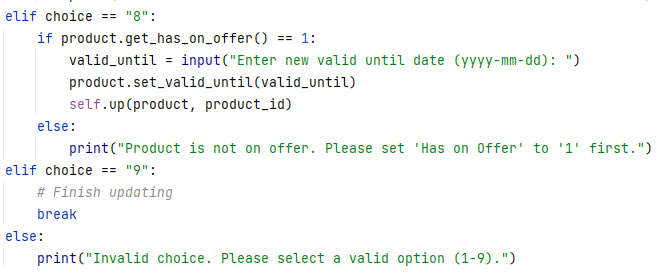


Figure 34:Update\_product function

Output:

Example of one option of update product menu:

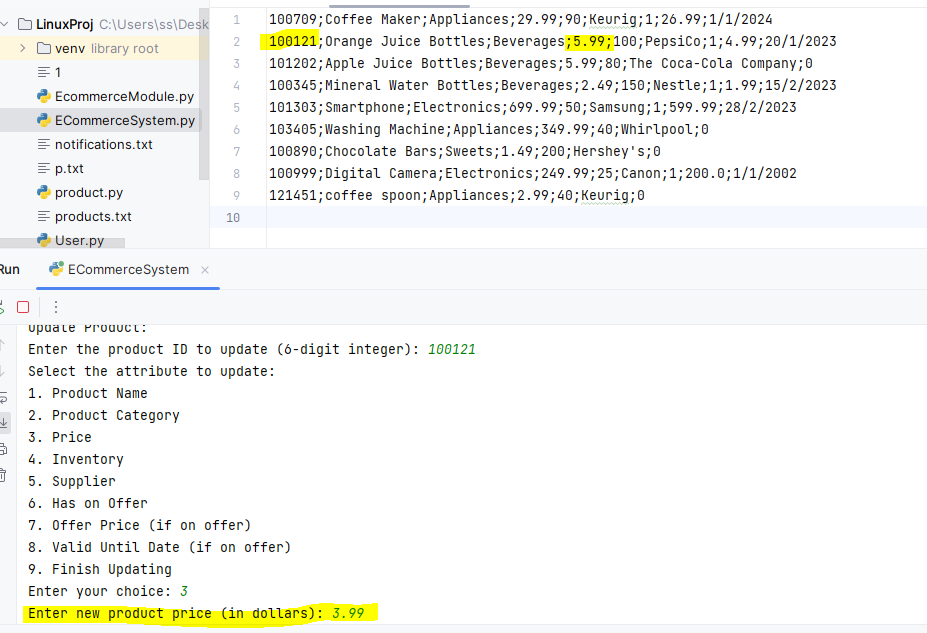


Figure 35:update product example

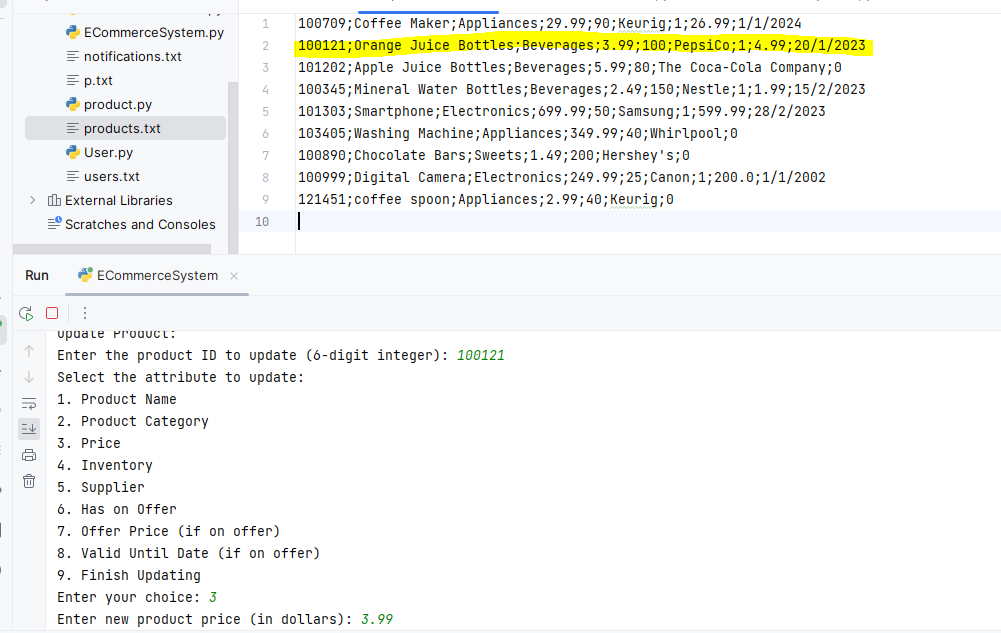


Figure 36: update product output

### 4.Add a New User (Admin-Only):

In this code segment, we have implemented a feature to add a new user to the system with user-friendly input validation to ensure data accuracy and consistency.

Here's a breakdown of how the code works:

Adding a New User:

The user is prompted to enter a 6-digit integer as the user ID.

A while loop validates the user ID format using the validate\_user\_id method. If the input is not a 6-digit integer or is not unique, the user is prompted to enter the user ID again.

The user is then prompted to enter their name and date of birth (in 'dd/mm/yyyy' format).

The role input is validated by converting it to an integer. The user is prompted to enter either 0 for admin or 1 for shopper until valid input is provided.

Similarly, the active status input is validated to be either 0 for inactive or 1 for active.

Creating a User Object:

Once all the user information is validated, a new User object is created with the provided details.

Appending User to List and Writing to File:

The new user is added to the list of users, ensuring efficient user management within the system.

The user's information is also written to an external text file named 'users.txt' to ensure data persistence.

Input Validation Methods:

The code includes three validation methods:

validate\_user\_id: Validates the user ID format and checks for uniqueness.

validate\_role: Validates the user's role (0 for admin, 1 for shopper).

validate\_active: Validates the user's active status (0 for inactive, 1 for active).

These methods help ensure that the user inputs are correct and meet the system's requirements.

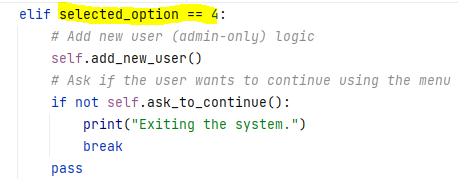


Figure 37:choice 4 handling

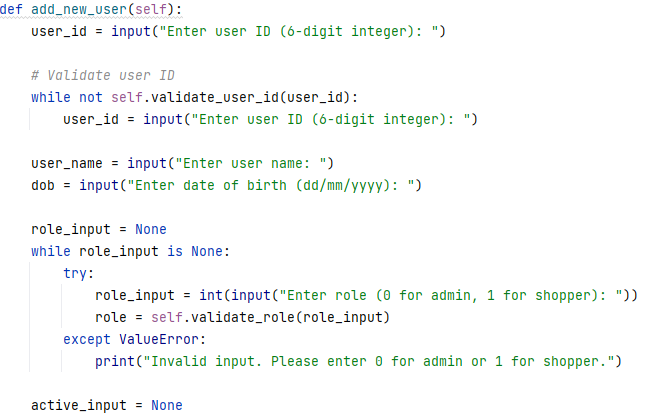
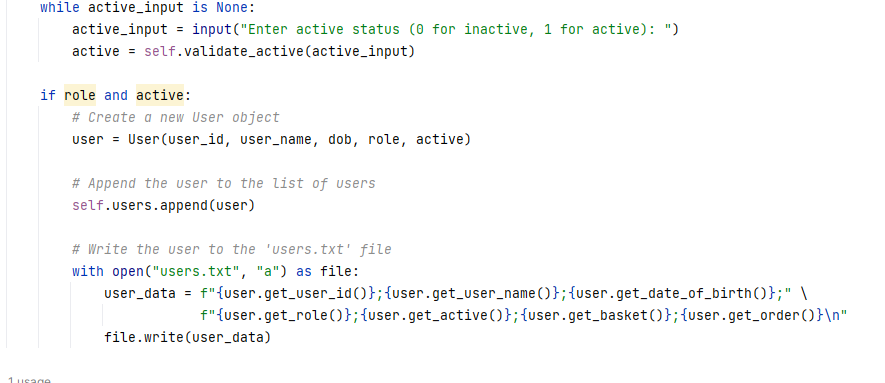


Figure 38:add user function

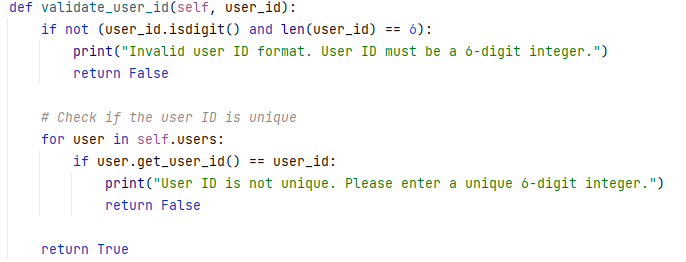


Figure 39: validate user id function

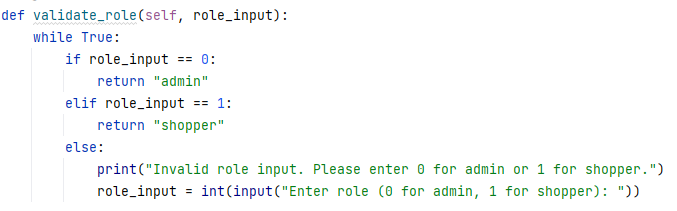


Figure 40:validate role function

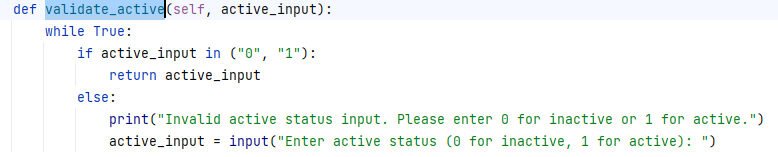


Figure 41:validate active function

Output:

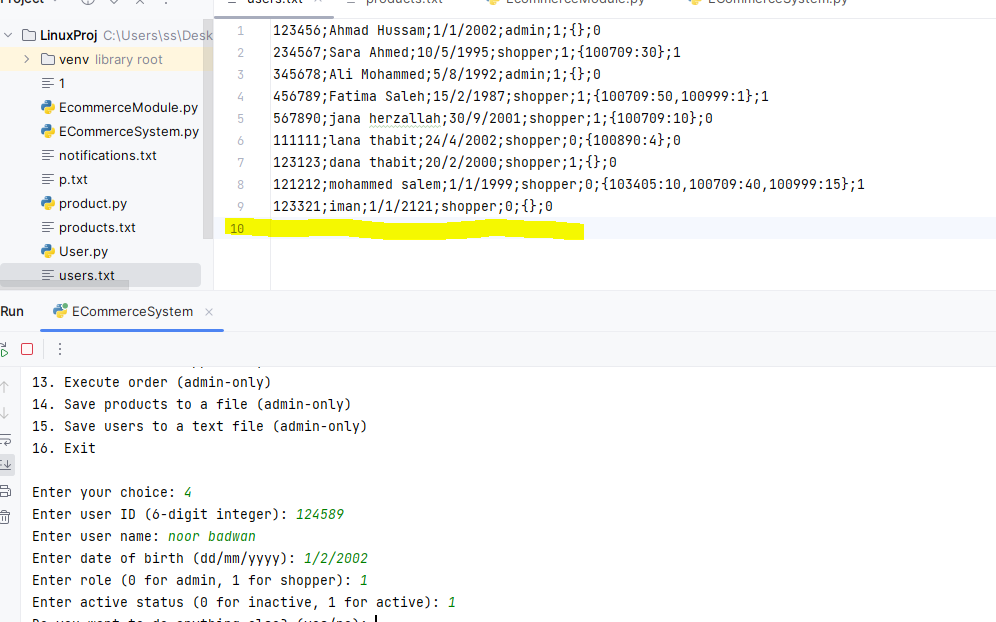


Figure 42:before add a user

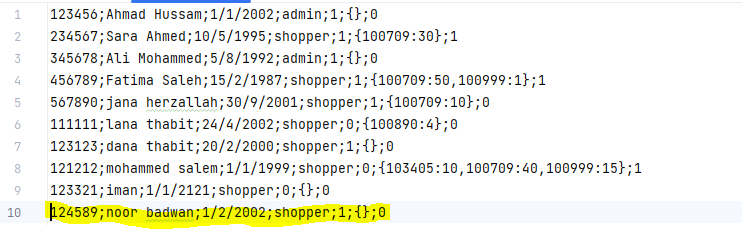


Figure 43: after user added

### 5.Update User (Admin-Only):

In this section, we have implemented a feature to update the attributes of an existing user within the system. This feature provides a user-friendly interface for updating user information, ensuring data accuracy and flexibility.

Here's how the code works:

Updating User Information:

The process starts by prompting the user to enter the user ID of the user they want to update. The system checks whether the user with the provided ID exists in the user catalog. If the user does not exist, an appropriate message is displayed, and the update process is terminated.

Selecting Attribute to Update:

Once a valid user is selected, the user is presented with a menu of attribute choices to update. The available choices include updating the user's name, date of birth, role, active status, and finishing the update process.

Updating User Attributes:

Each attribute update choice corresponds to an action:

User Name: The user is prompted to enter a new user name, and the user's name is updated.

Date of Birth: The user can input a new date of birth (in 'dd/mm/yyyy' format), and the user's date of birth is updated.

Role: The user is prompted to input a new role (0 for admin, 1 for shopper). The role is validated and converted to an integer for consistency, and the user's role is updated.

Active Status: The user can indicate a new active status (0 for inactive, 1 for active). The status is validated and updated accordingly.

Updating User File:

After each successful attribute update, the user's information is updated in the user catalog. Additionally, the user's information is written back to an external text file named 'users.txt' to ensure data persistence.

Finish Updating:

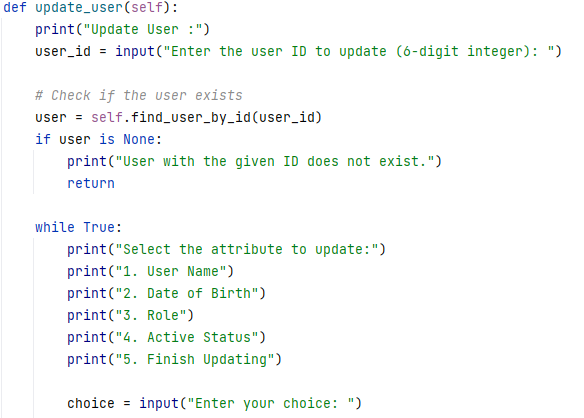
The user has the option to finish the updating process by selecting "5" from the menu.

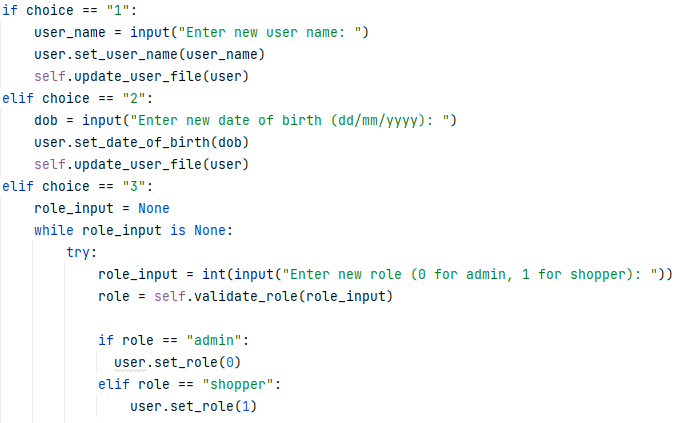
Input Validation Methods:

The code leverages the validate\_role and validate\_active methods to ensure that the role and active status inputs are correct and meet the system's requirements.



Figure 44:choice 5 handling





### 

Figure 45: update user function

### 

Figure 46update user file function

Output:

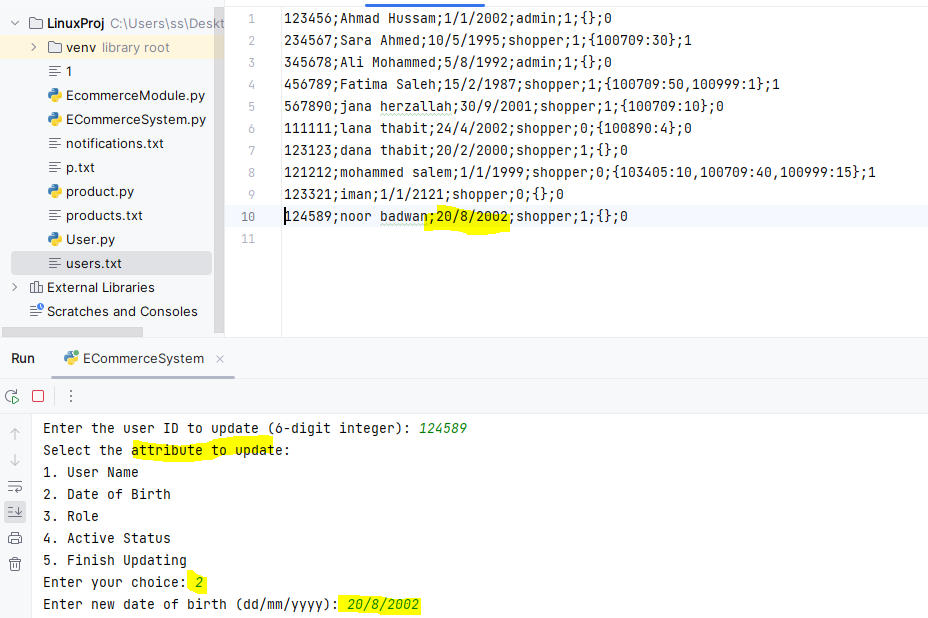


Figure 47:output of update user function

### 6.Display All Users (Admin-Only):

In this section, a method named display\_users is implemented to display a list of users along with their attributes. This method provides a clear and organized way to present user information to the system's users or administrators.

Here's how the code works:

Displaying User Information:

The method starts by printing a header indicating that it's displaying a list of users.

It then iterates through the list of users stored in the system.

For each user, it prints their user ID, user name, date of birth, role, active status, basket, and order information.

To enhance readability, a line of equal signs (=) is printed after each user's information to separate them clearly.

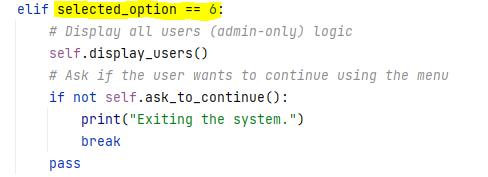


Figure 48:choice 6 handling

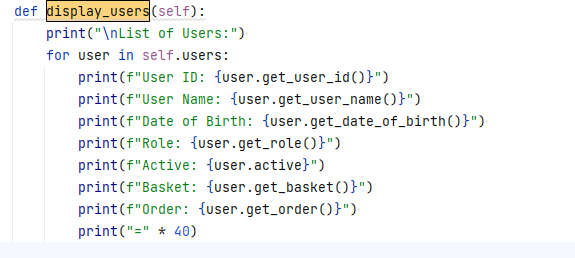


Figure 49:display users function

Output:



Figure 50:display users output

### 7.List Products (Admin and Shopper):

In this code segment, a series of methods are implemented to list and display products based on different criteria, offering users a flexible and informative way to access product information within the system. Let's break down how these methods work:

list\_products(self):

This method provides a menu-driven interface for users to choose criteria for listing products.

Users can choose to display all products, products with offers, products by category, products by name, or exit the listing process.

Input validation ensures that the user's choice is a valid option (1/2/3/4/5).

Depending on the user's choice, it calls the relevant display method to show the products.

1.display\_all\_products(self):

This method iterates through all products in the system and prints each product's information by calling product.to\_dict().

2.display\_products\_with\_offers(self):

It filters products with offers (where get\_has\_on\_offer() returns 1) and displays their information.

display\_categories(self):

This method identifies unique product categories within the system and displays them to help users choose a category for listing products by category.

3.display\_products\_by\_category(self, category\_name):

Users can enter a category name, and this method filters products in the specified category and displays their information.

4.display\_products\_by\_name(self, product\_name):

Users can enter a product name, and this method filters products with the specified name and displays their information.

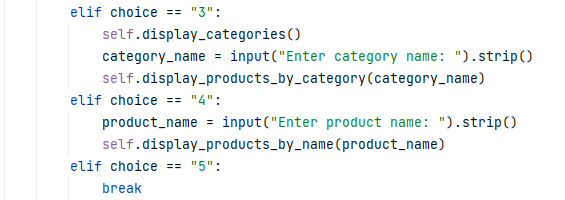
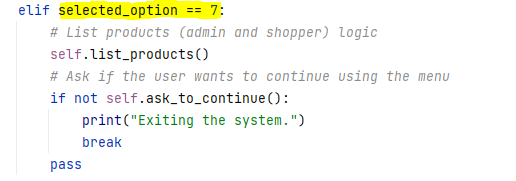
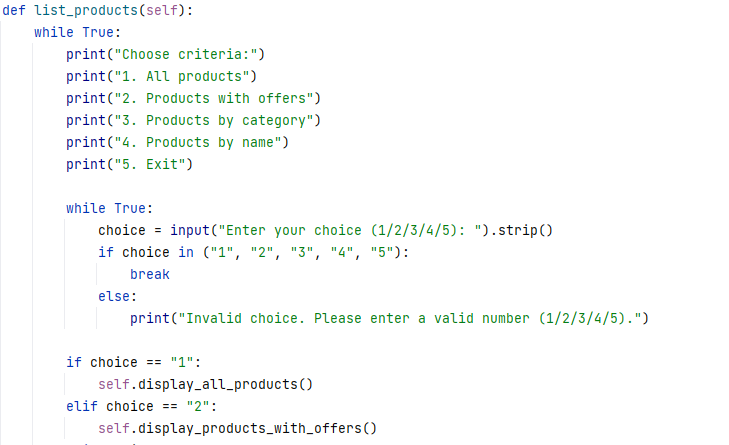
 

Figure 51: list products function

Figure 52:choice 7 handling

### 

Figure 53:other functions used to display products

### 

Output:

### 

Figure 54:list products output

### 8.List Shoppers (Admin-Only):

In this section, a series of methods are implemented to list and display shoppers based on different criteria, providing a user-friendly way to access shopper information within the system. Here's how these methods work:

1.list\_all\_shoppers(self):

This method provides a menu-driven interface for users to choose criteria for listing shoppers.

Users can choose to display all shoppers, shoppers with items in the basket, shoppers with unprocessed orders, or exit the listing process.

Input validation ensures that the user's choice is a valid option (1/2/3/4).

Depending on the user's choice, it calls the relevant display method to show the shoppers.

2.display\_all\_shoppers(self):

This method filters all users in the system with the role of "shopper" (role 1) and displays their information by calling shopper.to\_dict().

3.display\_shoppers\_with\_items\_in\_basket(self):

It filters shoppers (users with role 1) who have items in their basket (non-empty basket) and displays their information.

4.display\_shoppers\_with\_unprocessed\_orders(self):

This method filters shoppers (users with role 1) who have unprocessed orders (orders with order status 1) and displays their information.

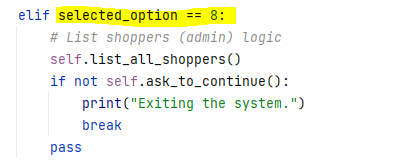


Figure 55:choice 8 handling

### 

Figure 56: List all shoppers function

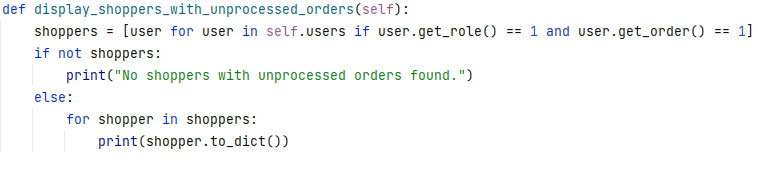
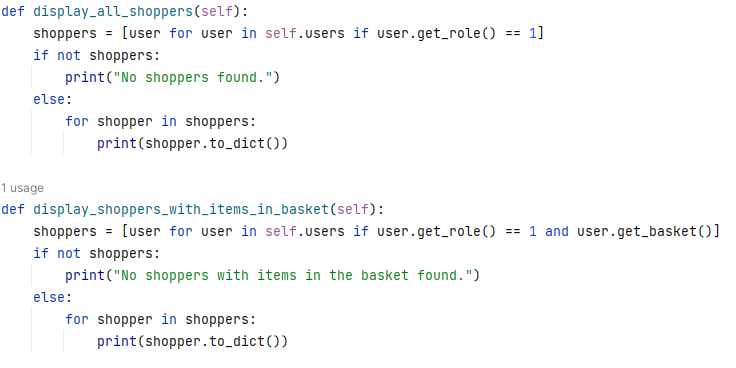


Figure 57:other functions used in display shoppers function

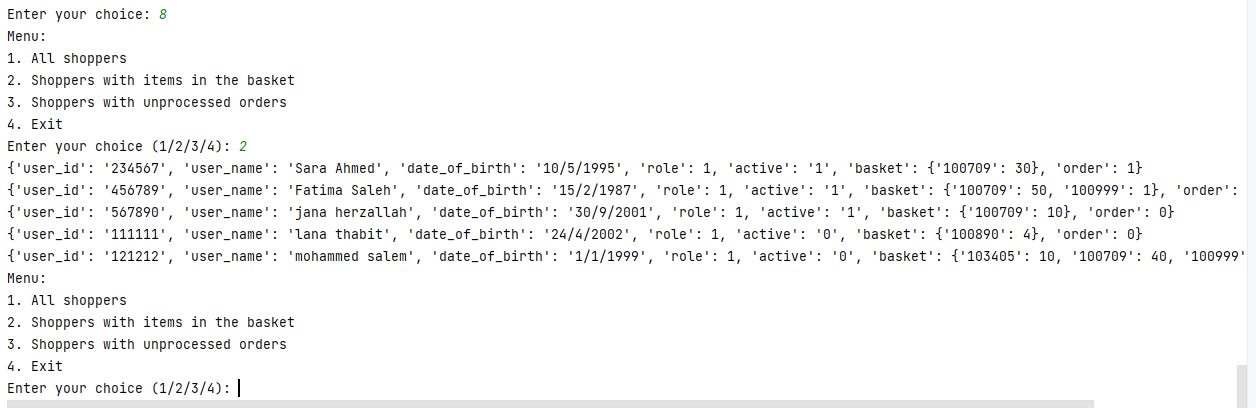
Output:

Figure 58:List shoppers output

### 9.Add Product to the Basket (Shopper-Only):

In this code segment, a method named add\_product\_to\_basket is implemented to allow a shopper to add products to their basket within the system. This method ensures a user-friendly and accurate process for adding products. Let's break down how it works:

Find Shopper by User ID:

The method starts by finding the user (shopper) by their user ID.

Check Shopper Existence:

It checks if the shopper (user) exists in the system. If the shopper is not found, the method prints an appropriate error message and exits.

Product Addition Loop:

Once the shopper is identified, the method enters a loop that allows the shopper to add products to their basket.

Within the loop, a list of available products is displayed using self.display\_all\_products().

Product Selection:

The user is prompted to enter the Product ID they want to add to their basket. They can also enter 'q' to quit and exit the loop.

Quantity Entry:

The user is then prompted to enter the quantity of the product they want to add. Input validation is applied to ensure a valid integer quantity.

Product Validation:

The method checks if the entered Product ID is valid (exists in the system) and if there is enough inventory to fulfill the request.

Basket Update:

If the product is valid and there is enough inventory, it is added to the shopper's basket, and a confirmation message is displayed.

The method also updates the user's information in an external text file ('users.txt') to reflect the changes in the basket. This involves reading the file, updating the basket data for the shopper, and writing the updated information back to the file.

Error Handling:

If the product ID is invalid or the quantity is invalid or exceeds available inventory, appropriate error messages are displayed.

User Existence Handling:

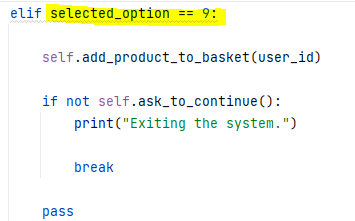
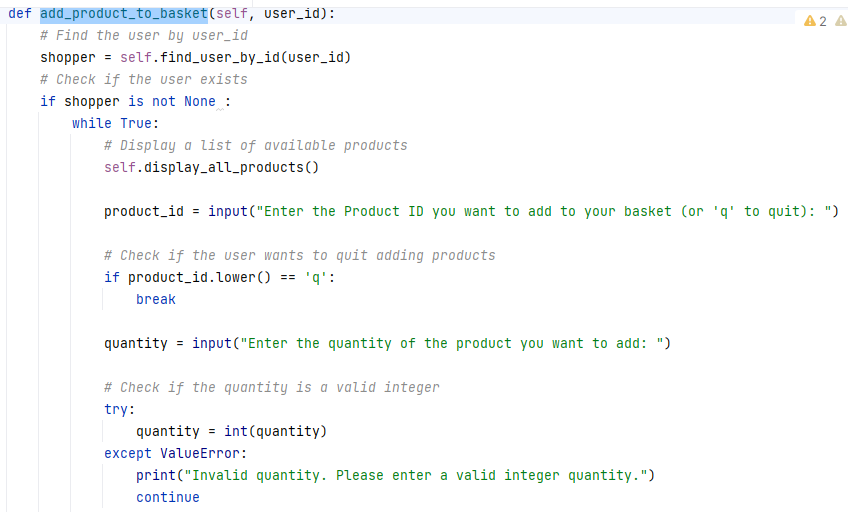
If the user ID provided is invalid or the user is not a shopper, an error message is displayed to indicate that the user is not found or is not eligible for this action.

Figure 59:choice 9 handling

### 

### 

Figure 60:add to basket function

Output:

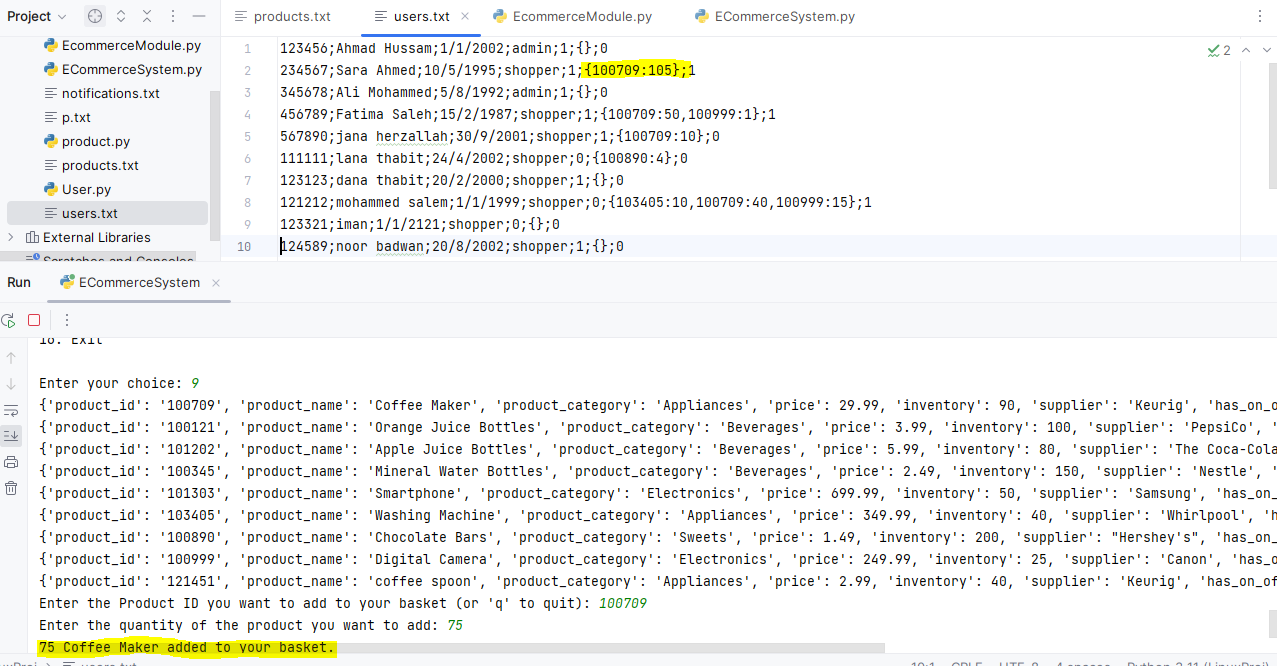


Figure 61:add to basket output

### 10.Display Basket (Shopper-Only):

In this section, a method named display\_basket is implemented to display the contents of a shopper's basket along with detailed product information and total cost calculation. This method provides a clear and informative view of the products in the basket. Let's break down how it works:

Find User by User ID:

The method starts by finding the user (shopper) by their user ID.

User Existence Check:

It checks if the user exists in the system. If the user is not found, an appropriate error message is displayed, and the method exits.

Shopper Role Verification:

The method verifies that the user has a shopper role (role 1). If the user is not a shopper, an error message is displayed, indicating that this function is available for shoppers only.

Basket Display:

If the user is a shopper with a valid basket, the method retrieves the basket data and displays it.

Basket Content Display:

It iterates through the items in the basket and displays detailed information for each product.For each product, it shows the product ID, product name, product category, price, inventory, supplier, offer price (if applicable), valid until date (if applicable), quantity in the basket, and cost of purchase for that product.

Cost Calculation:

The method calculates the total cost of items in the basket by summing up the cost of each product in the basket. It considers the offer price if the product has an offer, or the regular price if there is no offer.

Display Total Cost:

The total cost of items in the basket is displayed at the end with proper formatting.

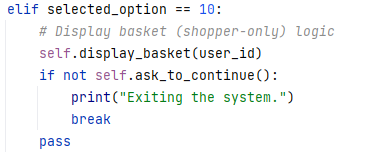


Figure 62:choice 10 handling

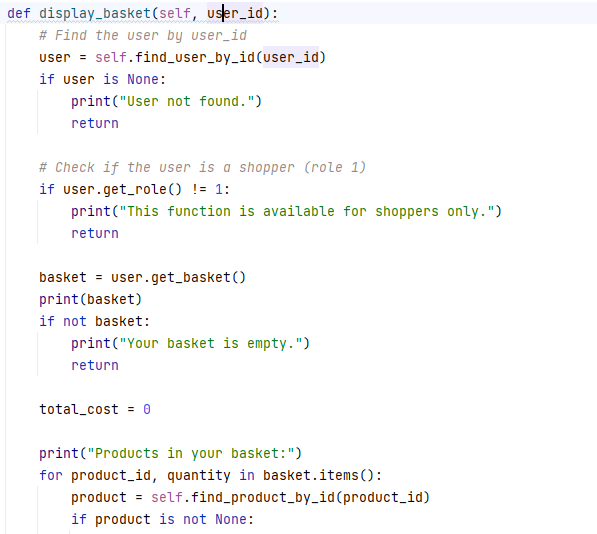
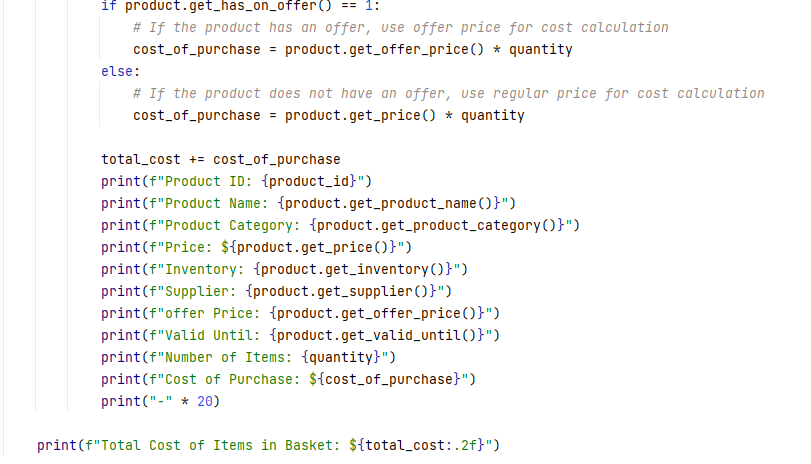


Figure 63:display basket function

Output:

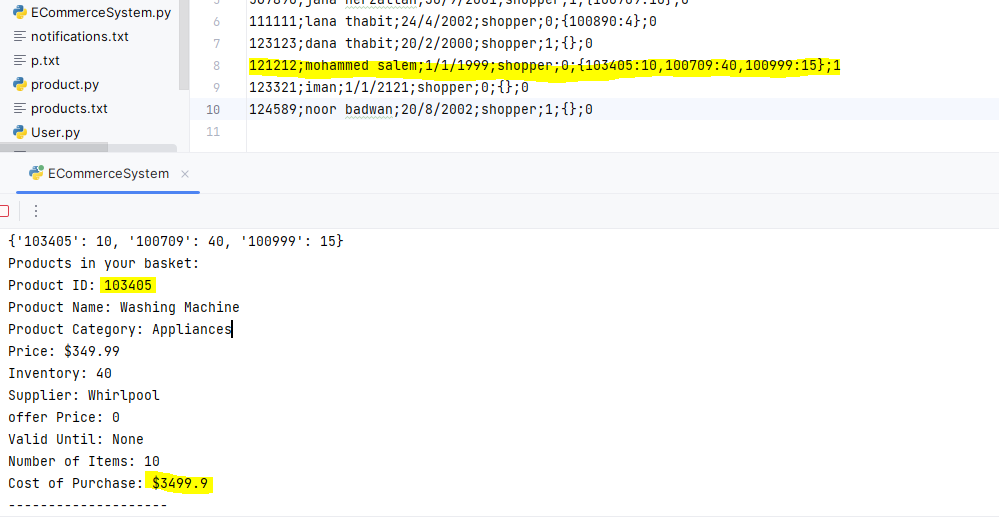
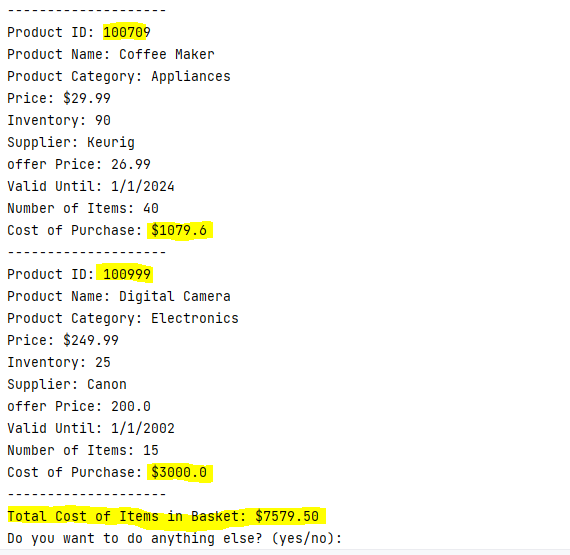


Figure 64:display basket output.

11.Update Basket (Shopper-Only):

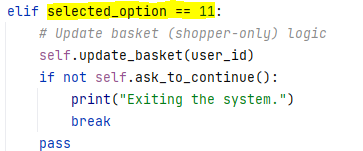


Figure 65:choice 11 handling

### 

### 

Figure 66:update basket function

### 

Output:

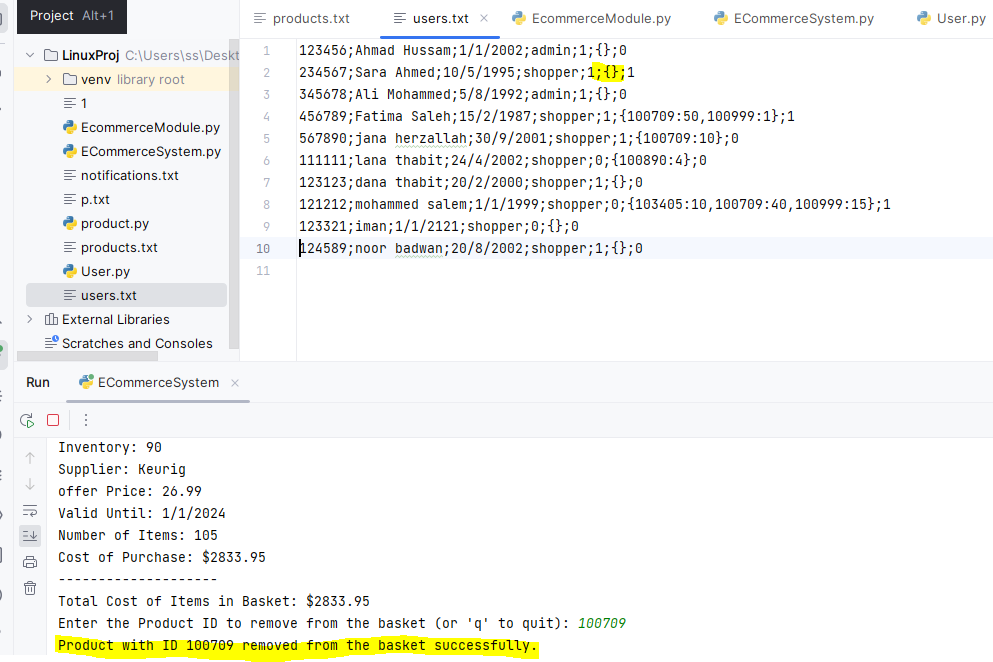


Figure 67:update basket output

### 12.Place Order (Shopper-Only):

In this section, a method named place\_order is implemented to allow shoppers to place an order within the system. Here's a breakdown of how it works:

Find User by User ID:

The method starts by finding the user (shopper) by their user ID.

User Existence and Role Check:

It checks if the user exists in the system and if the user is a shopper. If the user is not found or is not a shopper, an appropriate error message is displayed.

Order Placement Check:

It checks if the shopper already has an order placed (the order field is set to 1). If an order is already placed, a message is displayed to inform the shopper.

Placing an Order:

If the shopper does not have an order placed, the method proceeds to place an order. It sets the order field to 1 for the shopper, both in the in-memory list of users and in the external text file ('users.txt').

File Update:

The method updates the user's information in the external text file ('users.txt') to reflect the updated order status. It does this by reading the file, locating the user's data, modifying the order field to 1, and writing the updated information back to the file.

Success Message:

A success message is displayed to confirm that the order has been placed successfully.

Error Handling:

If the user ID is invalid or the user is not a shopper, an error message is displayed to indicate that the user is not found or is not eligible to place an order.

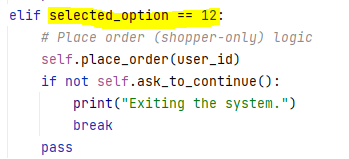


Figure 68:choice 12 handling

### 

### 

Figure 69:place order function

Output:

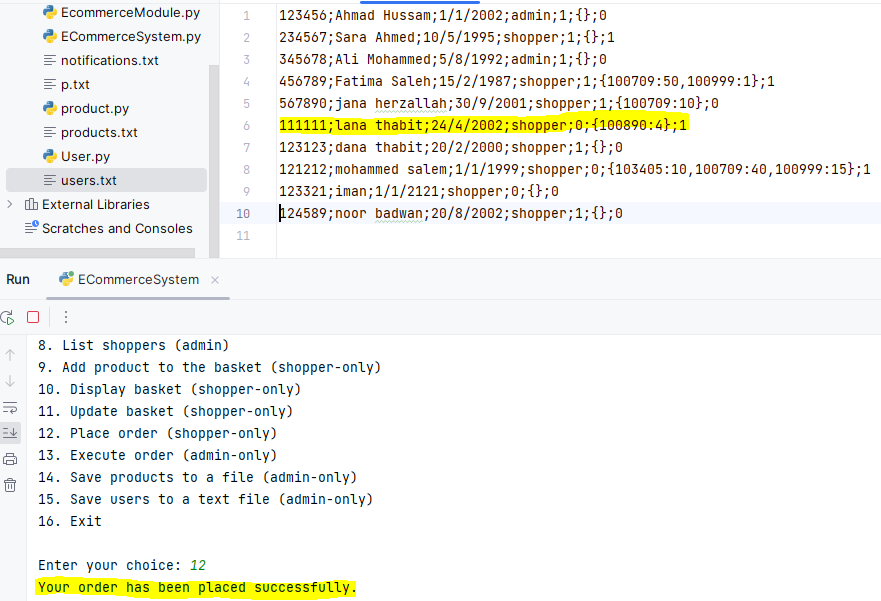


Figure 70:place order output

### 13.Execute Order (Admin-Only):

In this section, a method named execute\_order is implemented to allow admins to execute orders and save the notificaon if the order have a problems to inform the user when login with its id:

Input Validation:

The function begins by checking if the user (admin) exists and is an admin. It ensures that the admin is authorized to execute orders.

Menu:

The function presents a menu with three options:

List Shoppers with Unprocessed Orders: Lists all shoppers who have placed orders that are pending execution.

Execute Order for a Shopper: Allows the admin to execute an order for a specific shopper.

Exit: Allows the admin to exit the execute order menu and return to the main menu.

Option 1: List Shoppers with Unprocessed Orders:

If the admin selects this option, the function retrieves a list of shoppers with unprocessed orders.

Shoppers with unprocessed orders are those who have the shopper role (role 1) and have an order status of 1 (indicating an unprocessed order).

If no such shoppers are found, it displays a message indicating that no shoppers with unprocessed orders were found.

If shoppers are found, it lists their User IDs and User Names.

Option 2: Execute Order for a Shopper:

If the admin selects this option, the function prompts the admin to enter the User ID of the shopper whose order they want to execute.

The admin can also enter 'q' to quit and return to the main menu.

The function then checks if the entered User ID corresponds to a valid shopper.

Order Execution:

If the shopper is valid and has an unprocessed order, the function proceeds to execute the order.

It first checks if there is sufficient inventory for each product in the shopper's basket.

If there is sufficient inventory for all products, it deducts the items from the product inventory, clears the shopper's basket, and updates the order status for that shopper to indicate that the order has been processed.

If there is insufficient inventory for any product, the order is not accepted, and the reason for rejection is provided.

Order Outcome:

If the order is accepted and successfully executed, the function updates the product and user information.

It also displays a success message indicating that the order has been executed successfully.

If the order is not accepted due to insufficient inventory, the function provides a message indicating the reason for rejection and logs a notification in the 'notifications.txt' file.

Option 3: Exit:

If the admin selects this option, the function exits the execute order menu, and the loop ends.

Invalid Choices:

If the admin enters an invalid choice (other than 1, 2, or 3), the function displays an error message indicating that the choice is invalid.

That's the overall logic of the execute\_order function, which allows an admin to manage and execute shopper orders within the system.

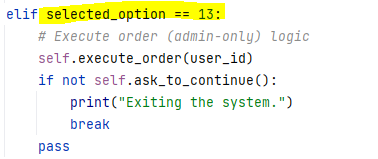


Figure 71:choice 13 handling

### 

### 

### 

### 

Figure 72:execute order function

We build the check notification function to inform shoppers with orders that have a problem so can not be execute by the admin.

This method allows a user (identified by User ID) to check their notifications related to their last order.

It reads the 'notifications.txt' file, searches for notifications related to the provided User ID, and displays any found notifications.

If notifications are found, they are displayed, and the user is informed.

The method also updates the 'notifications.txt' file by removing notifications related to the user, ensuring that notifications are not displayed multiple times.

If the 'notifications.txt' file does not exist, it informs the user.

These methods enhance the functionality of the system by allowing admins to execute orders for shoppers and allowing users to check notifications related to their orders. They ensure efficient order processing and communication with users regarding order status.

### 

Figure 73:check notification function

Outputs of execute order, check notification functions:

1. Accepted order:

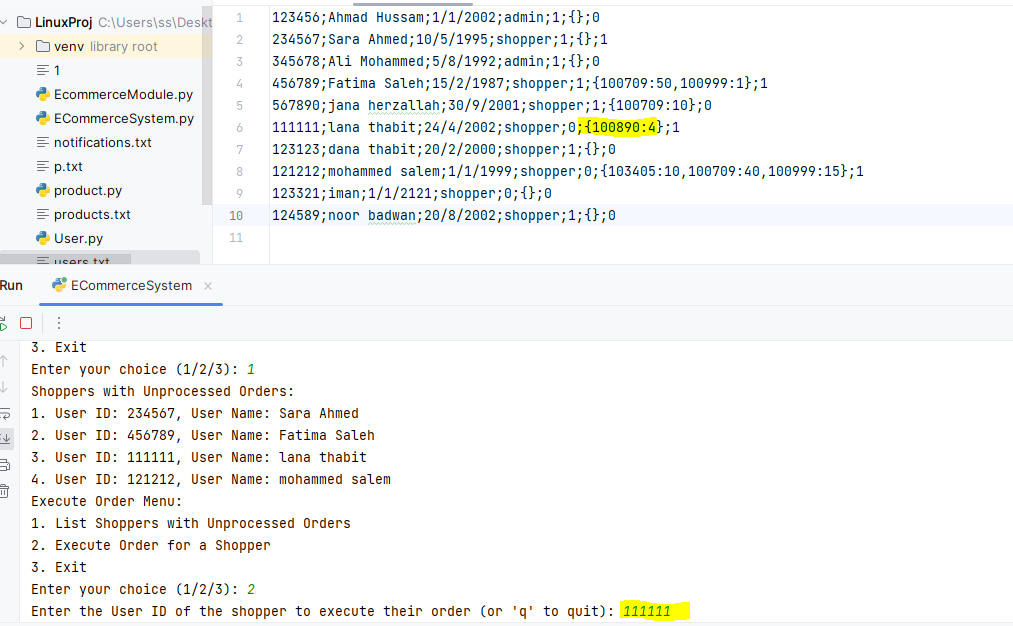


Figure 74:the file before execute the order

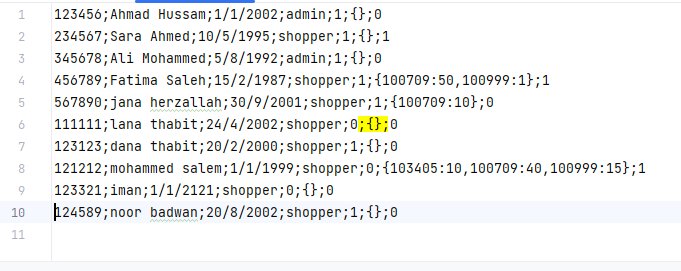
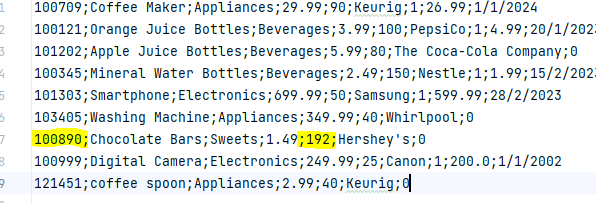


Figure 75:the file after execute the order

Note that the basket was cleared .  
  


The inventory of all products is also updated by decrement it with items numbers.

1. Unaccepted order (with user notification):

Assume shopper with id = 234567 has a basket order of product that has number of items large than its inventory.

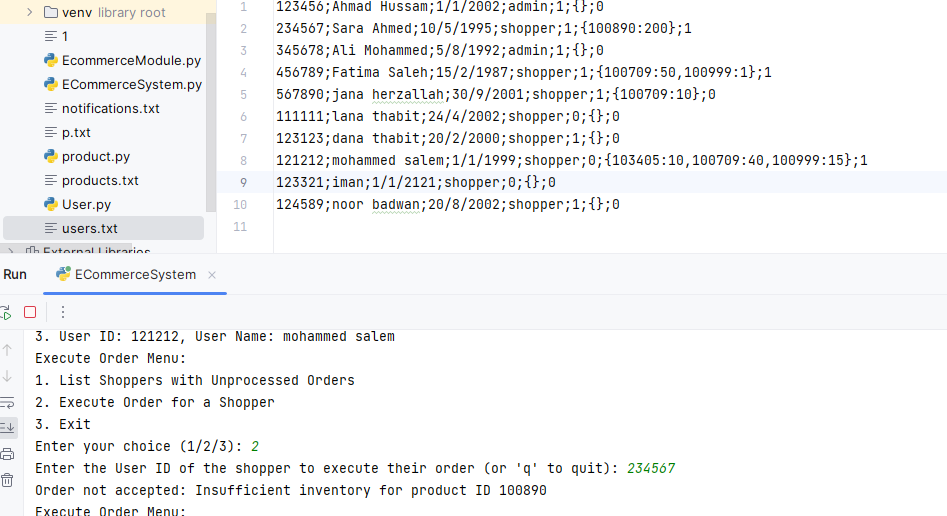
When the admin executes the order an alert notification added to notifications.txt file with shopper id such that when shopper log in with (his/her) id the notification will display.

Figure 76:unaccepted order

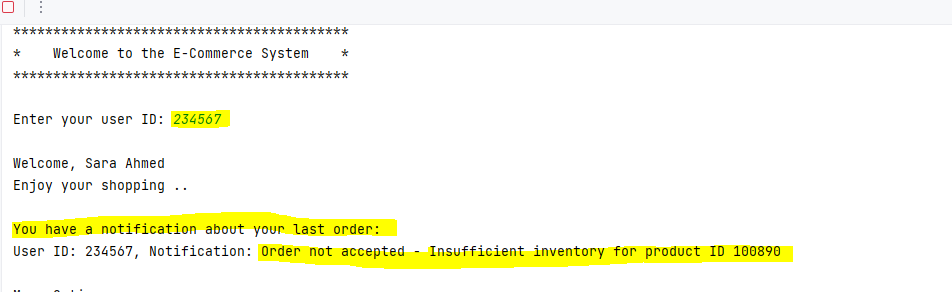


Figure 77:displaying shopper notification

### 14.Save Products to a File (Admin-Only):

In this section we define a module to manage writing data to files :

The function begins by opening the specified file (filename) in write mode ('w') using a context manager (with statement). This ensures that the file is properly closed after use.

It then iterates over each product object in the product\_list.

For each product, it checks whether the product has an offer (product.get\_has\_on\_offer() == 1). If the product has an offer, it includes additional information (offer price and valid until date) in the saved data. If not, it saves the basic product information without offer details.

It constructs a string (product\_data) containing all the relevant product information separated by semicolons (;).

It writes the product\_data string to the file, followed by a newline character (\n), to separate each product's data.

Once all products have been processed and their data written to the file, the function closes the file.

If there are any issues during the file write operation (e.g., if the file cannot be opened or written to), it handles the exception and prints an error message.

Finally, if the data is successfully written to the file, it prints a success message indicating that the products have been saved to the specified file.

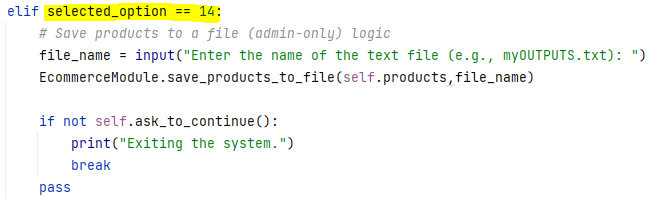


Figure 78:choice 14 handling

### 

### 

Figure 79:save products to a file function

Output:



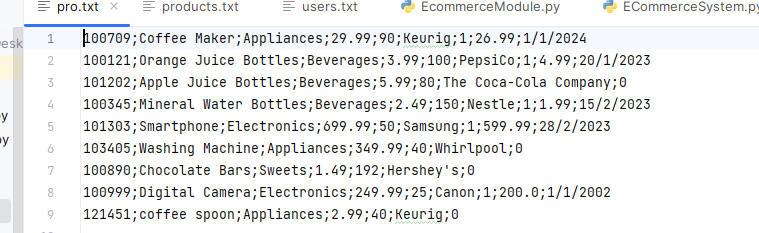


Figure 80: save products to file output

### 15.Save Users to a Text File (Admin-Only):

The goal of this section is to save a list of user objects to a text file in a specific format.

The function starts with a try...except block to handle potential exceptions, particularly IOError. This is a good practice to ensure the code can handle file-related errors gracefully.

Within the try block, the function opens the specified file (filename) in "append" mode ("a") using a context manager (with statement). Opening the file in append mode means that if the file already exists, new data will be appended to it, and if the file does not exist, it will be created.

The function then enters a for loop to iterate through each user object in the users\_list. For each user:

It extracts various user attributes (e.g., user ID, user name, date of birth, role, active status, basket content, and order status) using getter methods like user.get\_user\_id(), user.get\_user\_name(), etc.

It concatenates these user attributes into a string called user\_data in a specific format. Each attribute is separated by a semicolon (;), and the entire user record ends with a newline character (\n) to indicate the end of one user's data and the beginning of the next.

It then writes the user\_data string to the opened file using the file.write() method. This effectively appends the user's data to the end of the file.

After iterating through all users and writing their data to the file, the function prints a success message indicating that the users have been successfully saved to the specified file.

In the event of an IOError, the except block is executed, and an error message is printed, indicating that there was an issue writing to the file.

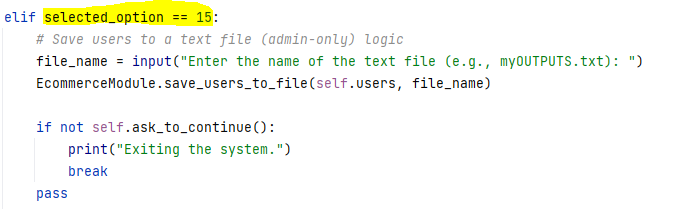


Figure 81:choice 15 handling

### 

Figure 82:choice 15 handling

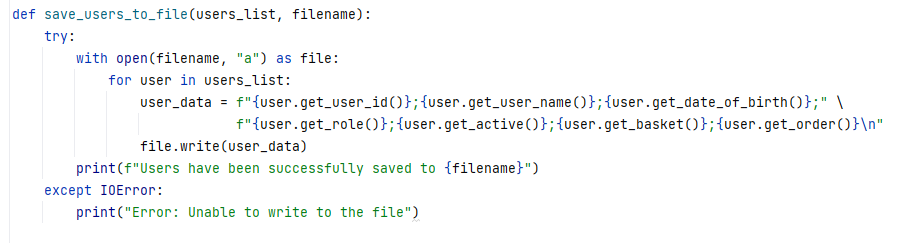


Figure 83:save users to file function

Output:

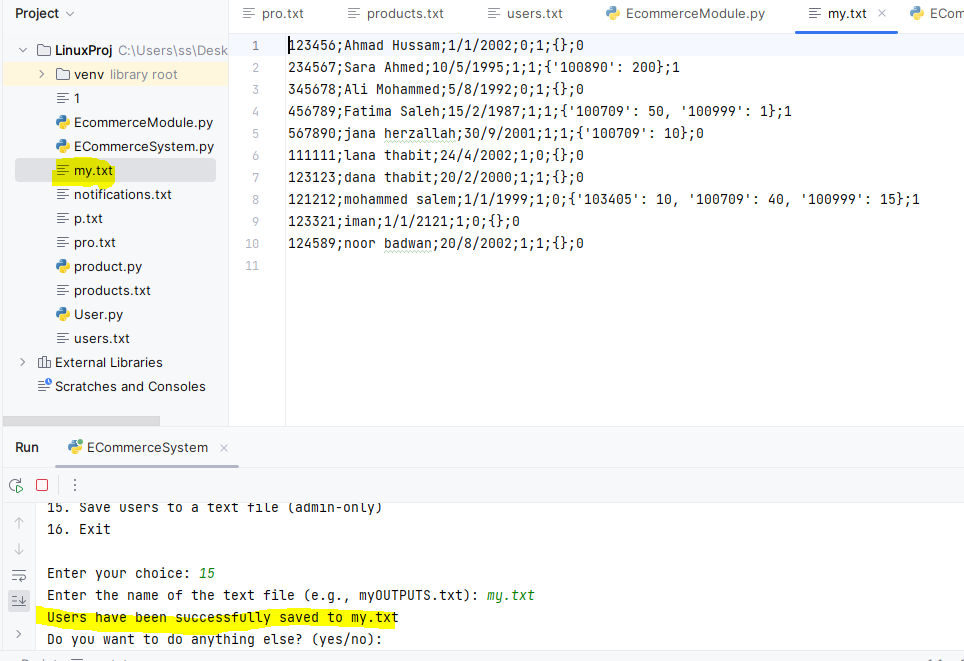


Figure 84:save users to file output

### 16.Exit:

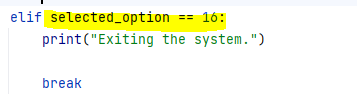
This section is just to exit the program.

Figure 85::choice 16 handling

Output:

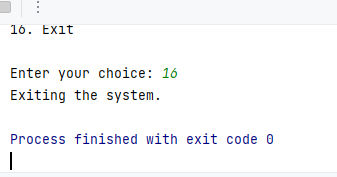


Figure 86:Exiting program output

# Conclusion

In conclusion, this E-commerce System project at Birzeit University's Department of Electrical & Computer Engineering for the Summer Semester 2022/2023 provided a comprehensive opportunity to apply key programming concepts, including OOP principles and modular organization. The project aimed to create a functional online shopping system with features for both administrators and shoppers, ensuring data integrity and user role validation. This hands-on experience allowed students to develop practical skills in Python and gain insights into building complex software systems. Overall, it was a valuable learning experience that challenged students to design and implement a real-world application with a focus on code quality and functionality.