Final Project Report

Name: Farrell Sevillen Arya

Student ID: 2702323540\

Class: L1AC

Course Name: Algorithm and Programming

Course Code: COMP6047001

Lecturer: Jude Joseph Lamug Martinez, MCS

A. Background

For the culmination of our algorithm and programming course, the final project presents an opportunity to showcase the knowledge acquired throughout the semester. This project isn't just an assessment; it's a challenge to push our boundaries and apply our skills in a real-world scenario. The task at hand involves creating an innovative application that not only leverages the core concepts we've learned but also ventures beyond the classroom teachings.

In the current landscape of retail, numerous stores struggle to adapt to the digital era. The primary hurdles include complex user interfaces and the necessity for exclusive devices, making it difficult for many businesses to transition into the online realm. Recognizing this, our project aims to tackle these challenges head-on by developing an application that prioritizes simplicity, accessibility, and ease of use.

B. Project Specification

1. Introduction

There were a lot of different ideas that I wanted to do, but the one I decided to make was an inventory system as it was what interested me the most. I had other ideas like making a game or something else, but I couldn’t decide on what type of game it should be.The objective of the inventory system is to provide an easy to use and accessible inventory system for store owners to store their data and modify it easily.

The user will use a device of their choice to use the app. They will open the app and be greeted with the user interface where they will have the option to add a

new item if they want to, if there is already an existing item, they can click on it to update any detail about the item or they can also delete the item. The user will also be able to see a chart at the bottom of the application page so that they have a visualization of what the stock of their items are like, making it easier to analyze whether the items that they are selling are any good or not as they can make their next move according to the represented data.

2. Libraries

- Sqlite3 - SQLite3 is a C library that provides a lightweight, disk-based database that doesn't require a separate server process and allows access to the database using a nonstandard variant of the SQL query language. It is a self-contained, serverless, and zero-configuration relational database engine. SQLite is particularly well-suited for embedded systems, mobile applications, and small to medium-sized websites.

- Tkinter - Tkinter is the standard GUI (Graphical User Interface) toolkit that comes with Python. It provides a set of tools and modules for creating graphical user interfaces in desktop applications. Tkinter is based on the Tk GUI toolkit and is the most commonly used GUI toolkit for Python.

- Matplotlib - Matplotlib is a 2D plotting library for Python that produces high-quality, publication-ready visualizations. It is a widely used library for creating static, animated, and interactive plots in Python. Matplotlib provides a versatile set of tools for creating a variety of plots and charts, making it suitable for data analysis, scientific research, and visualization tasks.

3. Important Files

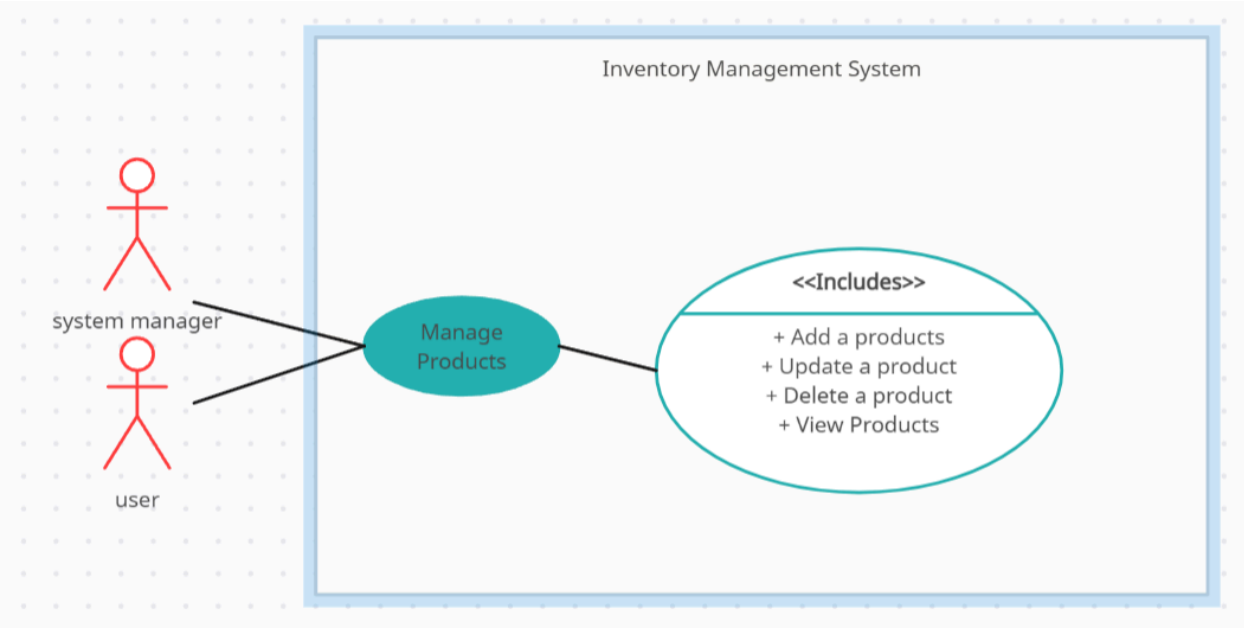
- ‘Database.py’ - This file includes the class which contains the code to create and modify the data inside the database that was created for the inventory system.
- ‘Execute.py’ - This file includes the code to create the GUI, and the buttons that will execute the functions in the ‘Database.py’ file, as well as containing the code that makes it so that the user can input the information of the products. Other than that, it also contains the code to display the list of entries in the database as well as the code to create the barchart to visualize the data of the stock levels of the entries.

C. Solution Design

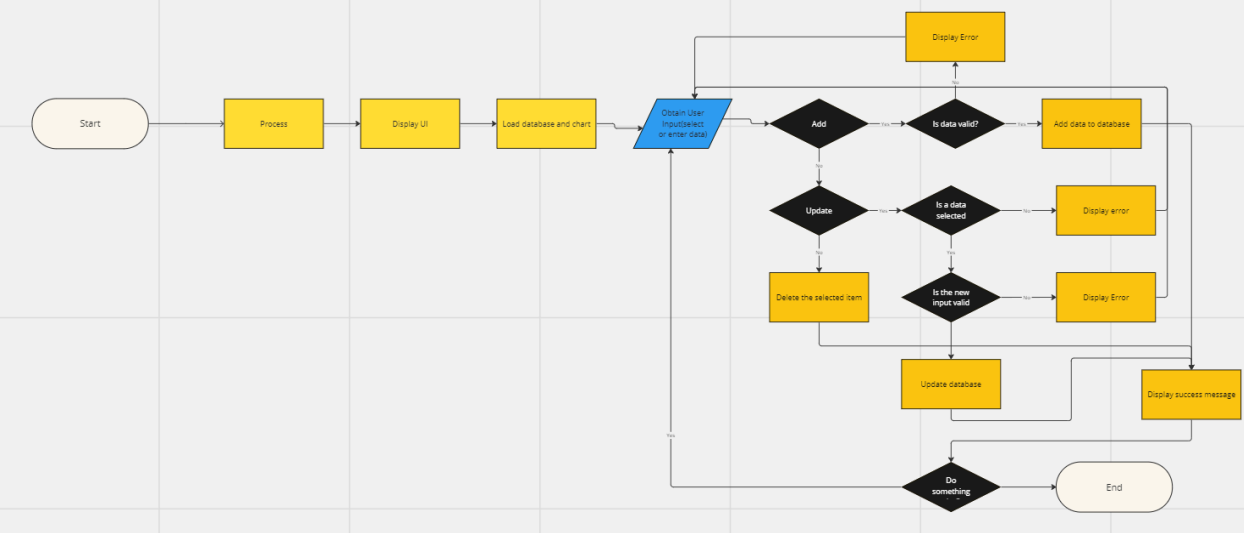
1. GUI MockUp

This is the mock up of the graphical user interface that I created using figma before creating the application. I wanted the user to see that it is an easy system to use and has a visualization of their current stock of products. The graphical user interface has 3 main components which are the place for the user to input and control the data, the list of entries, and the bar chart of the stock of the product.

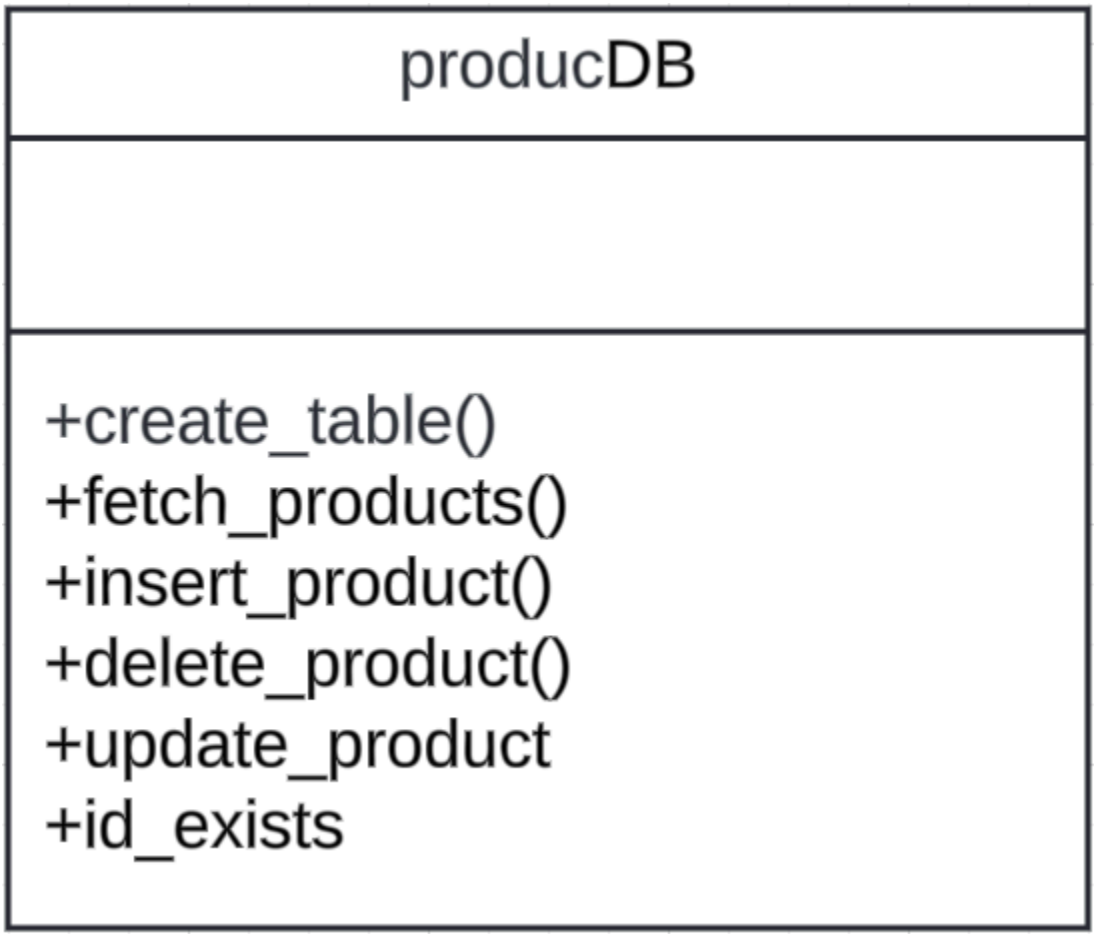
2. Use Case Diagram



3. Task Flow Diagram



4. Class Diagram



D. Code Explanation

1) Database.py

- ProductDB class

```
def __init__(self):
        self.create_table()


    def create_table():
        # Connect to the SQLite database
        connection = sqlite3.connect("Products.db")
```

```
        cursor = connection.cursor()

        # Create table with specified columns
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS Products (
                id TEXT PRIMARY KEY,
                name TEXT,
                in_stock INTEGER
            )
        """)

        # Commit changes and close the connection
        connection.commit()
        connection.close()
```

This function creates a new SQLite table called 'Products'. It includes three columns: 'id', 'name', and 'in_stock'. Each column stores the corresponding attributes of a product. This function is executed only once when the 'Products.db' file is first created.

```
def fetch_products():
        # Connect to the SQLite database
        connection = sqlite3.connect("Products.db")
        cursor = connection.cursor()

        # Retrieve all rows from the Products table
        cursor.execute("SELECT * FROM Products")

        # Fetch all retrieved rows as a list of tuples
        Products = cursor.fetchall()
```

```
        # Close the connection
        connection.close()
        return Products
```

This function retrieves all products from the 'Products' table. It takes no parameters. The function first establishes a connection to the SQLite database file. Then, it creates a new cursor object and executes an SQL query that selects all rows from the 'Products' table. The function then fetches all rows from the cursor and stores them in a list. Finally, the function returns the list of products and closes the connection.

```
def insert_product(id, name, in_stock):
        # Connect to the SQLite database
        connection = sqlite3.connect('Products.db')

        # Prepare SQL query to insert a new product
        cursor = connection.cursor()
        cursor.execute("INSERT INTO Products VALUES (?, ?, ?)", (id,
name, in_stock))

        # Commit changes and close the connection
        connection.commit()
        connection.close()
```

This function is used to insert a new product into the 'Products' table. It takes three parameters: 'id', 'name', and 'in_stock'. These parameters represent the unique identifier, name, and availability status of a product. The function first establishes a connection to the SQLite database file. Then, it creates a new

cursor object, which allows the function to execute SQL queries on the database. The cursor is used to execute an SQL query that inserts a new row into the 'Products' table, containing the provided product attributes. Finally, the function commits the changes to the database and closes the connection.

```
def delete_product(id):
        connection = sqlite3.connect('Products.db')
        cursor = connection.cursor()
        cursor.execute('DELETE FROM Products WHERE id = ?', (id,))

        # Commit changes and close the connection
        connection.commit()
        connection.close()
```

This function deletes a product from the 'Products' table based on its 'id'. It takes one parameter: 'id'. The function first establishes a connection to the SQLite database file. Then, it creates a new cursor object and executes an SQL query that deletes the row in the 'Products' table with the matching 'id'. Finally, the function commits the changes to the database and closes the connection.

```
def update_product(new_name, new_stock, id):
        connection = sqlite3.connect('Products.db')
        cursor = connection.cursor()
        cursor.execute('UPDATE Products SET name = ?, in_stock = ?
WHERE id = ?', (new_name, new_stock, id))

        # Commit changes and close the connection
        connection.commit()
        connection.close()
```

This function updates the information of an existing product in the 'Products' table. It takes four parameters: 'id', 'name', 'in_stock', and 'update_in_stock'. The first three parameters are the same as those of the 'insert_product' function, representing the product attributes. The last parameter, 'update_in_stock', is a boolean value that determines whether the 'in_stock' status of the product should be updated. The function first establishes a connection to the SQLite database file. Then, it creates a new cursor object and executes an SQL query that updates the row in the 'Products' table with the matching 'id'. If 'update_in_stock' is set to True, the function updates the 'in_stock' status of the product as well. Finally, the function commits the changes to the database and closes the connection.

```
def id_exists(id):
        connection = sqlite3.connect('Products.db')
        cursor = connection.cursor()
        cursor.execute('SELECT COUNT(*) FROM Products WHERE id = ?',
(id,))
        result = cursor.fetchone()
        connection.close()
        return result[0] > 0
```

This function checks if a product with a given 'id' exists in the 'Products' table. It takes one parameter: 'id'. The function first establishes a connection to the SQLite database file. Then, it creates a new cursor object and executes an SQL query that counts the number of rows in the 'Products' table with the matching 'id'. The function then fetches the result of the query, which is the count of rows, from the cursor. Finally, the function returns True if the count is greater than 0, indicating that a product with the given 'id' exists, and False otherwise. The function also closes the connection to the SQLite database file.

2) Execute.py

```
def create_chart():
    # Fetch product details from DB
    product_details = productDB.fetch_products()

    # Extract product names and stock values from details
    product_names = [product[1] for product in product_details]
    stock_values = [product[2] for product in product_details]

    # Create figure and subplot for bar chart
    figure = Figure(figsize=(10, 3.8), dpi=80,facecolor='#0A0B0C')
    ax = figure.add_subplot(111)

    # Create bar chart with product names and stock values
    ax.bar(product_names, stock_values, width=0.4, color='#1E90FF')

    # Customize chart labels and title
    ax.set_xlabel('Product Name', color='#fff',fontsize= 10)
    ax.set_ylabel('Stock Value', color='#fff', fontsize=10)
    ax.set_title('Product Stock Levels',color='#fff',fontsize=12)

    # Customize tick parameters
    ax.tick_params(axis='y',labelcolor='#fff', labelsize=12)
    ax.tick_params(axis='x',labelcolor='#fff', labelsize=12)

    # Set chart face color
    ax.set_facecolor('#1B181B')

    # Create Tkinter canvas for chart display
    canvas = FigureCanvasTkAgg(figure)
    canvas.draw()

    # Display chart in canvas widget
    canvas.get_tk_widget().grid(row=0, column=0, padx=0, pady=700)
```

This code fetches the data in the database and uses it to create a barchart using the fetch_products method from the productDB module. Then, it extracts the product names and stock values from the product details. Then, it creates a figure and a subplot for the bar chart using the Figure and add_subplot methods from the matplotlib.figure and matplotlib.axes modules, respectively. It creates the bar chart using the bar method of the ax object. This method takes the product names and stock values as arguments and creates a bar chart with the product names on the x-axis and the stock values on the y-axis. The rest are mostly to do with the customization of the bar chart and creating a space in Tkinter and displaying the chart.

```
def display_data(event):
    #Select row, populate inputs.
    selected_item = tree.focus()
    if selected_item:
        #Get selected row values.
        row = tree.item(selected_item)['values']
        clear()
        id_entry.insert(0,row[0])
        name_entry.insert(0,row[1])
        stock_entry.insert(0,row[2])
    else:
        pass
```

The display_data function is triggered when a row in the Treeview widget is clicked. The event object contains information about the click event. The function uses the tree.focus() method to get the currently selected item in the Treeview widget. If an item is selected, the function proceeds to extract the data from the selected row. This is done using the tree.item(selected_item)['values'] expression, which retrieves the values of the selected row. The clear function is called to clear the input fields. The input fields are populated with the data from the selected row. This is done using the insert method of the Entry widget, which inserts the specified text at the specified position in the Entry widget. In short, this

function is to display the selected item into the input spaces so that the user can edit it.

```
def add_to_treeview():
    #Add products to treeview.
    products = productDB.fetch_products()
    tree.delete(*tree.get_children())

    for product in products:
        #Insert product into treeview.
        tree.insert('', END, values=product)
```

The code starts by fetching all the products from the product database using the productDB.fetch_products() function. Next, the code deletes all the existing items in the treeview using the tree.delete(*tree.get_children()) function. Then, the code iterates over each product in the list of products fetched from the database. For each product, the code inserts the product into the treeview using the tree.insert() function.

```
def delete():
    # Selecting item in treeview
    selected_item = tree.focus()

    # Checking if an item is selected
    if not selected_item:
        messagebox.showerror('Error', 'Choose a product to delete.')
    else:
        # Getting the id of the selected product
        id = id_entry.get()

        # Deleting the product from the database
        productDB.delete_product(id)
```

```
        # Refreshing treeview and clearing the input fields
        add_to_treeview()
        clear()

        # Recreating the chart with updated data
        create_chart()

        # Displaying a success message
        messagebox.showinfo('Success','Data has been deleted')
```

First, the function retrieves the selected item in the treeview widget. If no item is selected, it shows an error message and does nothing else. If an item is selected, the function retrieves the ID of the selected product from the id_entry widget. Next, the function deletes the product with the given ID from the productDB database. After the deletion, the function refreshes the treeview by calling the add_to_treeview() function, and clears the input fields using the clear() function. It then updates the chart by calling the create_chart() function. Finally, the function displays a success message to indicate that the data has been deleted successfully.

```
def update():
    # Fetch the selected item in the treeview
    selected_item = tree.focus()

    # Check if any item is selected
    if not selected_item:
        # Display an error message if no item is selected
        messagebox.showerror('Error','Choose a product to update.')
    else:
        # Get the input values
        id = id_entry.get()
        name = name_entry.get()
```

```
        stock = stock_entry.get()

        # Update the product in the database
        productDB.update_product(name, stock, id)

        # Add the updated product to the treeview
        add_to_treeview()

        # Clear the input fields
        clear()

        # Update the chart with the new data
        create_chart()

        # Display a success message
        messagebox.showinfo('Success','Data has been updated.')
```

The code begins by fetching the selected item in the treeview, which is a widget that displays a hierarchical list of items. The focus() method is used to get the currently selected item. Next, the code checks if any item is selected. If no item is selected, an error message is displayed using the messagebox.showerror() function. If an item is selected, the code proceeds to get the input values for the updated product information from the user. The get() method is used to retrieve the input values from the respective Entry widgets. After retrieving the updated information, the code updates the product in the database using the update_product() method of the productDB object. This method takes the new name, stock, and product ID as arguments. After updating the product in the database, the code adds the updated product to the treeview using the add_to_treeview() function. The input fields are then cleared using the clear() function. The chart is updated with the new data using the create_chart() function. Finally, a success message is displayed to the user using the messagebox.showinfo() function.

```
def clear(*clicked):
    #Remove selected items, clear entries.
    if clicked:
        tree.selection_remove(tree.focus())
        tree.focus('')
    id_entry.delete(0,END)
    name_entry.delete(0,END)
    stock_entry.delete(0,END)
```

The clear() function in this code is designed to handle any number of arguments (*clicked) to accommodate potential items clicked in the program. It initiates by checking if there are any clicked items. If found, it deselects the item currently in focus and sets the focus to an empty string, ensuring that no items remain in a selected state. After addressing clicked items, the function proceeds to clear the id_entry, name_entry, and stock_entry widgets. This is achieved by invoking the delete method on each widget, specifying '0,END' as parameters to delete all characters from each entry field. The intended outcome of this code is to deselect any selected items and clear the contents of the relevant entry fields. This functionality proves particularly useful when a user has selected an item for editing but wishes to discard any changes. In summary, the clear() function ensures a clean slate by deselecting items and wiping entry fields, offering users a convenient way to reset or cancel ongoing operations.

```
def insert():
    # Retrieve input data from Entry fields
    id = id_entry.get()
    name = name_entry.get()
    stock = stock_entry.get()

    # Check if all fields are filled
    if not (id and name and stock):
```

```
        messagebox.showerror("Error", "Please fill all fields")

    # Check if ID already exists in database
    elif productDB.id_exists(id):
         messagebox.showerror("Error","ID already exists! Please use
a different ID.")

    else:
        try:
            # Convert stock value to integer
            stock_value = int(stock)

            # Insert product into database
            productDB.insert_product(id,name,stock_value)

            # Update Treeview with new data
            add_to_treeview()

            # Clear Entry fields
            clear()

            # Create/update chart with new data
            create_chart()

            # Show success message
            messagebox.showinfo('Success','Data has been inserted')

        except ValueError:
            # Show error message if stock value is not an integer
              messagebox.showerror("Error","Stock must be an integer
value")
```

The function first retrieves the input data from Entry fields. These fields are typically text input boxes where the user can type in data. It then checks if all the fields (ID, Name, and Stock) are filled. If not, an error message is displayed using

messagebox.showerror(). If all fields are filled, the function proceeds to check if the product ID already exists in the database. If the ID exists, an error message will be displayed. If the product ID does not exist, the function attempts to convert the stock value to an integer. If this conversion fails (because the stock value is not an integer), an error message will be displayed. If the stock value can be successfully converted to an integer, the function proceeds to insert the product into the database. After successfully inserting the product, the function updates the Treeview with the new data by calling the add_to_treeview function. The function then clears the Entry fields to prepare for the next product entry. To update the chart with the new data, the function calls the create_chart function. Finally, the function displays a success message to inform the user that the data has been inserted.

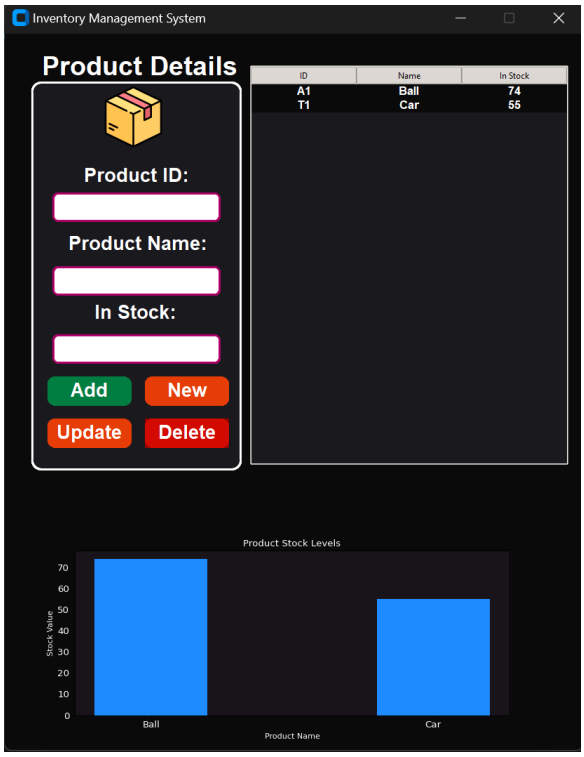
```
# Making the label and buttons in the UI
title_label = customtkinter.CTkLabel(app,font=font1,text='Product
Details',text_color='#fff',bg_color='#0A0B0C')
title_label.place(x=35,y=15)

frame =
customtkinter.CTkFrame(app,bg_color='#0A0B0C',fg_color='#1B1B21',cor
ner_radius=10,border_width=2,border_color='#fff',width=200,height=37
0)
frame.place(x=25,y=45)

image1 = PhotoImage(file="box.png")
image1_label = Label(frame, image=image1,bg='#1B1B21')
image1_label.place(x=100,y=5)

id_label = customtkinter.CTkLabel(frame,font=font2,text='Product
ID:',text_color='#fff',bg_color='#1B1B21')
id_label.place(x=50,y=75)
```

```
id_entry =
customtkinter.CTkEntry(frame,font=font2,text_color='#000',fg_color='
#fff',border_color='#B2016C',border_width=2,width=160)
id_entry.place(x=20,y=105)

name_label = customtkinter.CTkLabel(frame,font=font2,text='Product
Name:',text_color='#fff',bg_color='#1B1B21')
name_label.place(x=35,y=140)

name_entry =
customtkinter.CTkEntry(frame,font=font2,text_color='#000',fg_color='
#fff',border_color='#B2016C',border_width=2,width=160)
name_entry.place(x=20,y=175)

stock_label = customtkinter.CTkLabel(frame,font=font2,text='In
Stock:',text_color='#fff',bg_color='#1B1B21')
stock_label.place(x=60,y=205)

stock_entry =
customtkinter.CTkEntry(frame,font=font2,text_color='#000',fg_color='
#fff',border_color='#B2016C',border_width=2,width=160)
stock_entry.place(x=20,y=240)

add_button =
customtkinter.CTkButton(frame,command=insert,font=font2,text_color='
#fff',text='Add',fg_color='#047E43',hover_color='#025B30',bg_color=
'#1B1B21',cursor='hand2' ,corner_radius=8,width=80)
add_button.place(x=15,y=280)

clear_button =
customtkinter.CTkButton(frame,command=lambda:clear(True),font=font2,
text_color='#fff',text='New',fg_color='#E93E05',hover_color='#A82A00
',bg_color='#1B1B21',cursor='hand2',corner_radius=8,width=80)
clear_button.place(x=108,y=280)
```

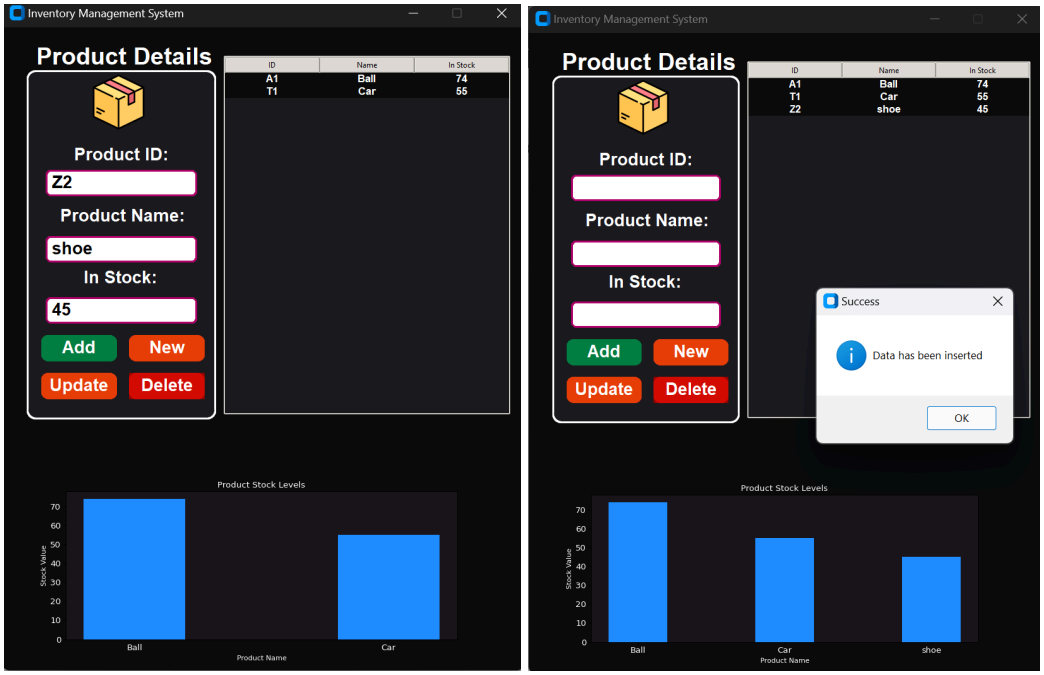
```
update_button = customtkinter.CTkButton(frame,
command=update,font=font2,text_color='#fff',text='Update',fg_color='
#E93E05',hover_color='#A82A00',bg_color='#1B1B21',cursor='hand2',cor
ner_radius=8,width=80)
update_button.place(x=15,y=320)

delete_button = customtkinter.CTkButton(frame,
command=delete,font=font2,text_color='#fff',text='Delete',fg_color='
#D20B02',hover_color='#A82A00',bg_color='#8F0600',cursor='hand2',cor
ner_radius=8,width=80)
delete_button.place(x=108,y=320)
```

This code is to create all the things such as the text above the place where the user inputs the product information, the place to input the information, the buttons that execute the functions, by linking them to it.

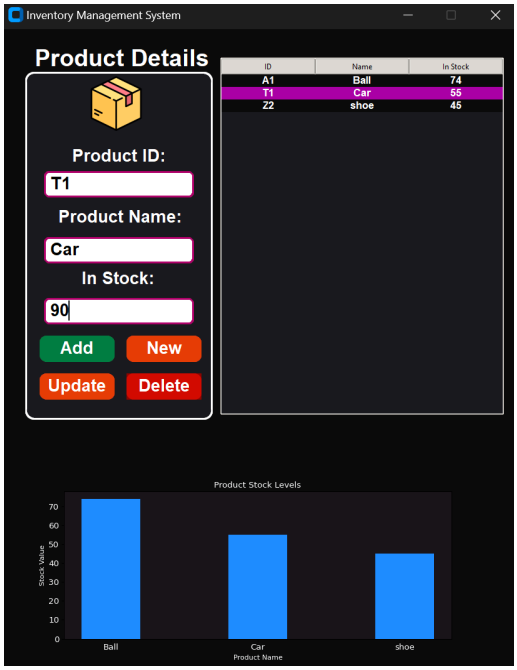
E. Application Evidence

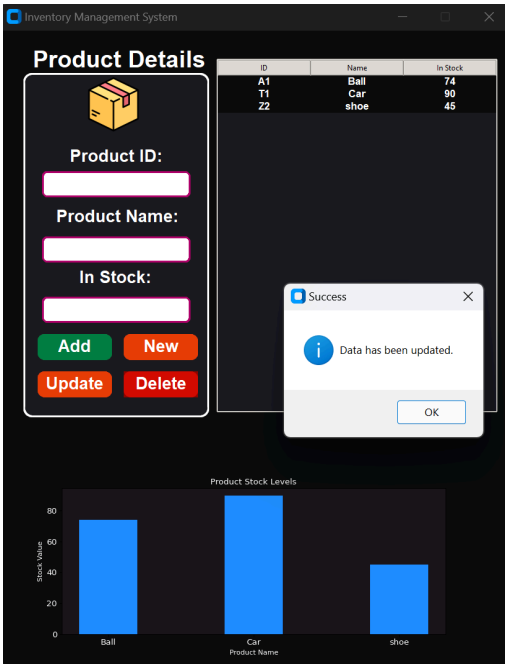


This is the app when it is booted up

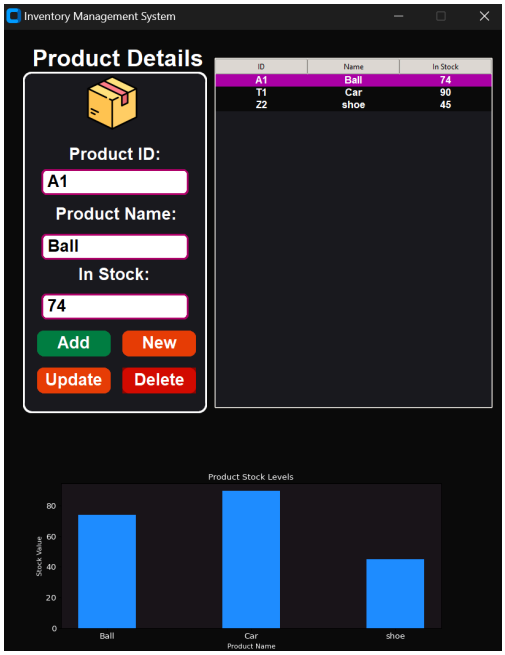


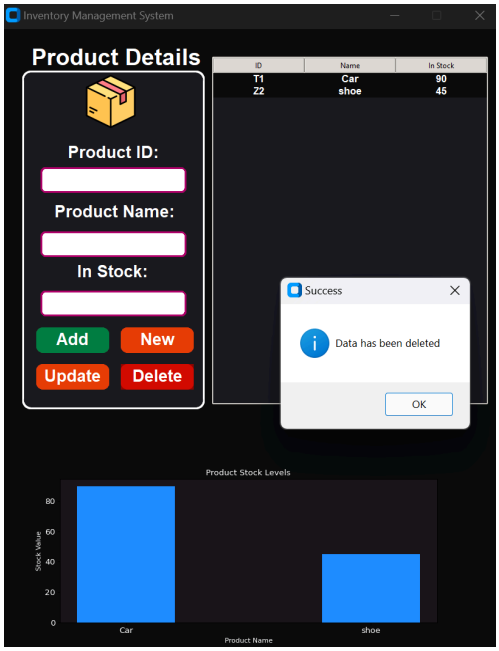
Successfully adding an item





Updating an item





Deleting an item

F. Evaluation and Reflection

1. Evaluation

During the creation of the application, there were quite a few challenges I had to face and overcome. The first was that there were multiple times when I was trying to install the libraries such as Tkinter, sqlite3, and others, it would not install and say that there was an error in installing it. To overcome that challenge I tried to do multiple things, but the one that worked was to just restart my computer as well as my visual studios code application.

When creating the program, I took a more step by step process to how it was built. I first started with the functions to create a table for the data in the application. Then I went on with creating the graphical user interface, there were quite a few problems for me in this part. I had to map out the GUI's X and Y axis so that it would fit all the elements I wanted to place in my application. Another problem was that sometimes, the elements would not be in the right place when I ran the program, so I had to do some trial and error on the location of the elements so that it would be in the right place.

Overall, the problems were able to be solved and I finished the program in the right time to submit.

2. Reflection

Throughout this project I was faced with multiple challenges that made me adapt and learn on the go. Before thinking up of the Inventory management application app, I thought of multiple different things, such as a game or a random color generator, but ultimately I chose to create the inventory management app.

The first problem I ran into was that when I came around to starting to code the program, there was a bit of a time crunch as I was busy previously with different final projects and tasks from different classes. When starting the process of creating the program, I also found another problem which is that there were too many different tutorials for me to learn from. At first I decided to follow a few

different ones, but all of them were a bit too complicated for me or they had libraries that my coding app could not install.

Another problem was that I felt like my program was too simple, but in the end I decided to leave it as there was not enough time for me to thinker and change the code as I was afraid that it would cause an error that would be too hard to fix, or that the function would have taken too much time for me to properly implement to my code. Some of the ideas I wanted to implement was to be able to read bar codes and make the adding and updating functions more intuitive with automatic reorder when the stock reached zero.

Overall, I am quite satisfied with my work, but there is still a part of me that still feels like I could have done way more and made the application better. I am very thankful that I was given the opportunity to embark on this journey as I was able to learn a lot from this experience and develop my skills.

References:

- INVENTORY MANAGEMENT SYSTEM PYTHON CUSTOMTKINTER MODERN TKINTER PROJECT WITH SQLITE3 DATABASE - https://www.youtube.com/watch?v=zO8W_0FdllA&t=285s