OptiChain

1. Abstract

In the dynamic area of supply chain management within the fast-moving consumer goods (FMCG) sector, the OptiChain project emerges as a significant breakthrough, leveraging the power of Python programming to change established processes. This project, which has its roots in the dynamic world of business operations, goes beyond simple software development and is a prime example of how technology and industrial knowledge can be combined to solve difficult problems.

Fundamentally, OptiChain is a full-featured supply chain management tool created to give companies the information and resources they need to successfully negotiate the complexities of contemporary trade. By using the concepts of Python programming with great care, the project provides a comprehensive strategy for increasing productivity, improving decision-making, and promoting corporate success.

The project is implemented with a feature set that has been painstakingly designed to satisfy the various needs. With features like revenue tracking, sales analysis, inventory management, logistics data, and data export capabilities, OptiChain offers a comprehensive solution made to meet the particular needs of the sector. With the help of user-friendly interfaces and the smooth integration of data analytics tools, users can successfully identify growth possibilities, manage risks, and make well-informed decisions.

OptiChain is an example of how technology may revolutionize traditional business procedures by representing the meeting point of academic knowledge and real-world application. Beyond its practical use, the project represents a team effort that leverages the combined knowledge of business analysts, software engineers, and industry experts to produce a solution that cuts over conventional boundaries.

OptiChain is a shining example of innovation that helps companies traverse the complexity of the current supply chain environment by providing a window into the activities of FMCG in the future. Its influence goes beyond the field of software development, igniting a revolution in the way companies see supply chain management and highlighting the revolutionary potential of Python programming to bring about change across the board.

2. Project Aim and Objectives

Aim:

Design and implement OptiChain, a comprehensive supply chain management application, to streamline business operations and drive efficiency.

Objectives:

- Develop a user-friendly interface for easy navigation and interaction.
- Integrate modules for recording revenue, managing inventory, analyzing sales growth, and optimizing logistics.
- Implement dynamic data handling to ensure real-time updates and accuracy.
- Incorporate visualization tools to provide insightful analytics and reports.
- Enable seamless data export functionality for further analysis and sharing.
- Ensure scalability and flexibility to accommodate diverse business needs.
- Document user guidelines and instructions for efficient usage.
- Foster collaboration by facilitating communication and coordination among stakeholders.
- Collect feedback from users to continuously improve and enhance the application.

- Conduct regular evaluations to assess performance and identify areas for optimization.

3. Background

OptiChain emerges as a response to the growing demand for efficient supply chain management solutions in today's dynamic business landscape. Rooted in the recognition of the critical role supply chain optimization plays in driving business success, OptiChain aims to provide organizations with a powerful tool to streamline their operations and maximize efficiency.

Educational Foundation:
The project is founded on the principle of experiential learning, offering users a practical platform to deepen their understanding of supply chain management concepts. By engaging users in hands-on exploration and decision-making scenarios, OptiChain reinforces key principles and methodologies in a dynamic and interactive manner.

Innovative Approach:
OptiChain pioneers a novel approach to supply chain management education by combining traditional learning methods with cutting-edge technology. Through the integration of intuitive interfaces, data visualization tools, and real-time analytics, the project empowers users to grasp complex concepts and make informed decisions with confidence.

Practical Application:
Driven by the belief that learning is most effective when applied in real-world scenarios, OptiChain provides users with a simulated environment to test their skills and strategies in managing various aspects of the supply chain. By simulating common challenges and scenarios encountered in real-world operations, the project equips users with practical experience and prepares them for success in their professional endeavors.

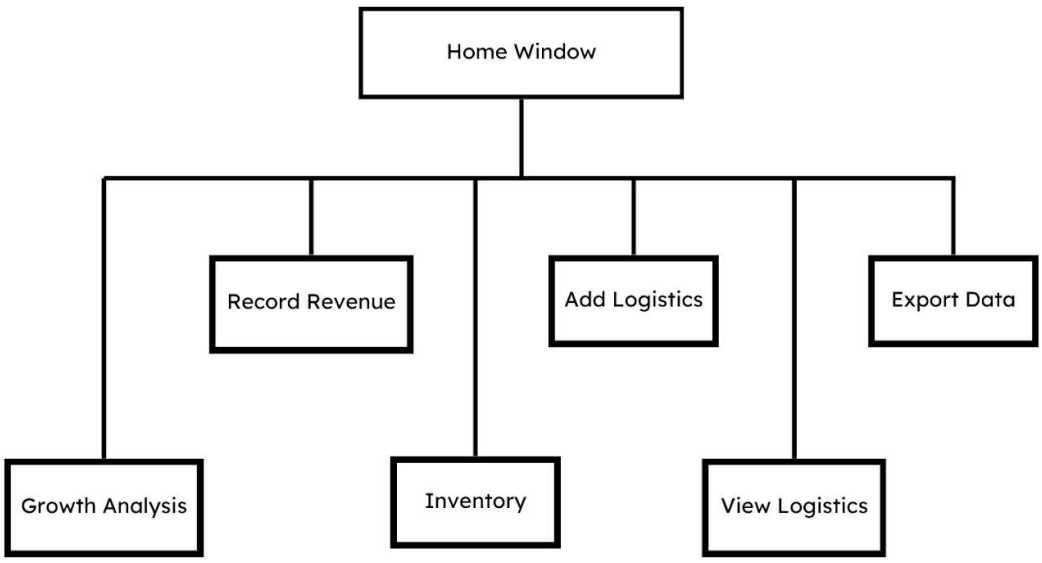
Collaborative Development:
OptiChain fosters collaboration and knowledge-sharing within the supply chain management community, inviting industry professionals, educators, and enthusiasts to contribute their expertise and insights. Through collaborative development efforts, the project aims to continuously evolve and adapt to meet the evolving needs of users and the industry as a whole.

Enhancing Efficiency:
At its core, OptiChain is driven by a commitment to enhancing efficiency and driving operational excellence across the supply chain. By providing users with tools and insights to optimize inventory management, streamline logistics, and maximize revenue, the project seeks to empower organizations to achieve their business goals and thrive in competitive markets.

4. Project Scope

Here, we talk about the various important fragments of the program code.

a. Home Window



i. The home window of the OptiChain app is the main interface that users encounter upon launching the application. It serves as the gateway to accessing various features and functionalities offered by OptiChain. Here's an explanation of the components and elements present in the home window.
ii. Clickable buttons arranged in a user-friendly layout, each representing a distinct feature or application functionality.
iii. Features include options such as recording revenue, managing inventory, analyzing sales growth, viewing logistics information, adding new suppliers, and exporting data.
iv. Users can intuitively navigate to the desired feature by clicking on the corresponding button.

b. Record Revenue Window

i. The window title is set to "Record Revenue" to indicate its purpose clearly.
ii. A description provides guidance to users on how to use the window, including instructions for recording revenue, displaying the current date and time, and any other relevant details.
iii. A label "Revenue (₹)" is displayed alongside a Spinbox widget, allowing users to input the revenue amount. The Spinbox provides a convenient interface for selecting numeric values within a specified range.
iv. Upon entering the revenue amount and clicking the "Enter data" button, the entered data is processed and stored in the SQLite database.
v. The data insertion process involves executing an SQL query to insert the revenue entry into the "Revenue_Sheet" table of the database. The date and revenue amount are included in the INSERT query as values.

c. Growth Analysis Window

i. The "View Sales Pitch" window in the OptiChain app provides users with a graphical representation of sales data over time.
ii. The graph represents data for the last 20 revenue entries, allowing them to understand the scope of the data displayed.
iii. Users can gain insights into sales trends by analyzing the graph.
iv. They may observe patterns, fluctuations, or trends in revenue over the specified time period.
v. This graphical representation helps users make informed decisions about sales strategies, inventory management, and overall business performance.

d. Logistics Window

i. This window provides a convenient way to access and view details about logistics partners associated with their firm.
ii. Users notice that the window displays a list of logistics partners along with their respective manager's name and contact details.
iii. Each logistics partner is presented within a labeled frame, making it easy to distinguish between different partners.
iv. Users observe that the window dynamically adjusts its content based on the data available in the database.
v. If there are no logistics partners stored in the database, users see a message indicating so and suggesting a course of action to add new contacts.
vi. If there are logistics partners stored in the database, users can view their details directly within the window.
vii. The window retrieves data from the database, ensuring that the information displayed is up-to-date.
viii. They can rely on this window to access important contact information for logistics partners, facilitating smooth communication and coordination in supply chain management tasks.

e. Add Logistics Window

i. This window provides a convenient way to add new logistics contacts to the database.
ii. Users observe a form with fields for entering the retailer's name, manager's name, and contact number.
iii. Each field is labeled clearly, making it easy for users to understand what information is expected in each field.
iv. Users can input the necessary details directly into the entry fields.
v. Once the user fills in the required information and clicks the "Add Contact" button, the data will be added to the database.
vi. They can rely on this functionality to keep the database updated with the latest logistics contacts, ensuring smooth communication and coordination in supply chain management tasks.

f. Export Data

i. Users can access the export data feature from the main window of the OptiChain application, where it is likely represented by a button or menu option labeled "Export Data."
ii. The application connects to the SQLite database where the relevant data is stored.
iii. It retrieves the necessary data, such as revenue records, inventory information, or logistics contacts, depending on what data is meant to be exported.
iv. The application formats the retrieved data in a suitable manner for export to Excel.
v. This formatting may involve organizing the data into rows and columns, applying appropriate headers, and ensuring the data types are compatible with Excel.
vi. Once the data is properly formatted, the application generates a new Excel file (usually with a predefined name or user-specified name).
vii. The Excel file contains the exported data in a structured format that users can easily view and manipulate using Excel or any compatible spreadsheet software.

5. Development Environment

a. Programming Language:

i. OptiChain is primarily developed using Python, a versatile and widely-used programming language known for its simplicity, readability, and extensive library support.

ii. Python's robust ecosystem provides developers with access to libraries such as Tkinter for building graphical user interfaces (GUIs), SQLite for database management, Pandas for data manipulation, and Matplotlib for data visualization.

b. Integrated Development Environment (IDE):

i. Developers may utilize popular IDEs like PyCharm, Visual Studio Code, or Jupyter Notebook to write, debug, and test Python code efficiently.

ii. These IDEs offer features such as syntax highlighting, code completion, debugging tools, version control integration, and package management, enhancing the development workflow.

c. Version Control System (VCS):

i. Git is employed as the version control system for OptiChain's codebase, enabling collaborative development, code review, and tracking of changes across the project.

ii. Platforms like GitHub or GitLab serve as centralized repositories for storing the project's source code, facilitating collaboration among developers and ensuring code integrity.

d. Database Management System (DBMS):

i. OptiChain utilizes SQLite, a lightweight and self-contained relational database management system (RDBMS), for storing and managing data related to revenue, inventory, logistics, and other aspects of the supply chain.

ii. SQLite's simplicity, portability, and seamless integration with Python make it suitable for embedded and small to medium-scale applications.

e. User Interface (UI) Development:

i. Tkinter, a standard GUI toolkit for Python, is leveraged to create the user interface components of OptiChain's desktop application.
ii. Tkinter provides widgets such as buttons, labels, entry fields, and frames, enabling developers to design intuitive and interactive interfaces for managing inventory, recording revenue, analyzing sales, and accessing logistics information.

f. Testing Frameworks:

i. OptiChain incorporates testing frameworks like unittest or pytest to automate the testing process and ensure the reliability, functionality, and performance of its components.
ii. Test-driven development (TDD) practices may be employed to write tests before implementing new features, promoting code quality and facilitating code maintenance.

g. Continuous Integration and Deployment (CI/CD):

i. CI/CD pipelines are established to automate the build, test, and deployment processes of OptiChain, streamlining the delivery of updates and enhancements to end-users.
ii. Tools like Jenkins, Travis CI, or GitHub Actions may be configured to orchestrate CI/CD workflows, perform code analysis, execute tests, and deploy releases to production environments.

6. Source Code

Full code repository and other files are available at :
https://github.com/theasych/OptiChain

```
import tkinter
from tkinter import PhotoImage
import sqlite3
from datetime import datetime
import tkinter.messagebox
import matplotlib.pyplot as plt
import pandas as pd
import subprocess

def record_revenue():
    """
    Function to record revenue in a database table.
    """
    conn = sqlite3.connect("data.db")
    table_create_query = """CREATE TABLE IF NOT EXISTS Revenue_Sheet
                    (date STR, revenue INT)
            """
    conn.execute(table_create_query)

    cursor = conn.cursor()

    # Check if there is an entry for today's date
    cursor.execute("SELECT * FROM Revenue_Sheet WHERE date = ?",
(datetime.today().date(),))
    existing_row = cursor.fetchone()

    revenue_window = tkinter.Toplevel(window)
    revenue_window.title("Record Revenue")
    revenue_frame = tkinter.Frame(revenue_window)
    revenue_frame.pack()
    desc_frame = tkinter.LabelFrame(revenue_frame, text="Description")
    desc_frame.grid(row=0, column=0, padx=20, pady=10)

    description = f"""
    Record total revenue income for today.
```

```
    Date : {datetime.today().date()}
    Time : {datetime.now().time()}
    """
    label_description = tkinter.Label(
        desc_frame, text=description, padx=10, pady=20, justify="left"
    )
    label_description.pack()


    app_frame = tkinter.LabelFrame(revenue_frame, text="Add data")
    app_frame.grid(row=1, column=0, padx=20, pady=10)

    income_label = tkinter.Label(app_frame, text="Revenue (₹)")
    income_spinbox = tkinter.Spinbox(app_frame, from_=1, to="infinity")
    income_label.grid(row=2, column=0, padx=10, pady=5)
    income_spinbox.grid(row=2, column=1, padx=10, pady=5)
    data_insert_query = (
        """INSERT INTO Revenue_Sheet (date, revenue) VALUES (?, ?)"""
    )
    data_insert_tuple = (datetime.today().date(), income_spinbox.get())

    def enter_revenue():
        """
        Function to enter revenue data into the database.
        """
        cursor.execute(data_insert_query, data_insert_tuple)
        conn.commit()
        conn.close()
        # Close the revenue window after data is entered
        revenue_window.destroy()

    button = tkinter.Button(revenue_frame, text="Enter data",
command=enter_revenue)
    button.grid(row=3, column=0, sticky="news", padx=20, pady=10)

def inventory_window():
    """
    Opens a new window to manage inventory and update SKU quantities.
    """
    inventory_window = tkinter.Toplevel(window)
    inventory_window.title("Manage Inventory")
```

```
    inventory_frame = tkinter.Frame(inventory_window)
    inventory_frame.pack()

    desc_frame = tkinter.LabelFrame(inventory_frame, text="Description")
    desc_frame.grid(row=0, column=0, padx=20, pady=10)

    description = "Manage inventory for the below SKU units.\nUse the
spinboxes to edit the quantity numbers\nof each SKUs"
    label_description = tkinter.Label(
        desc_frame, text=description, padx=27, pady=20, justify="left"
    )
    label_description.pack()

    app_frame = tkinter.LabelFrame(inventory_frame, text="Add data")
    app_frame.grid(row=1, column=0, padx=20, pady=10)

    conn = sqlite3.connect("data.db")
    table_create_query = """CREATE TABLE IF NOT EXISTS SKU_Sheet
                    (SKU87423 INT, SKU53982 INT, SKU21675 INT, SKU70148
INT, SKU38291 INT, SKU96574 INT)
            """
    conn.execute(table_create_query)

    cursor = conn.cursor()

    cursor.execute("SELECT * FROM SKU_Sheet")
    row = cursor.fetchone()

    SKU87423, SKU53982, SKU21675, SKU70148, SKU38291, SKU96574 = row

    inv_label = tkinter.Label(app_frame, text="SKU87423")
    inv_label.grid(row=0, column=0, padx=10, pady=5)
    income_spinbox0 = tkinter.Spinbox(app_frame, from_=f"{SKU87423}",
to="infinity")
    income_spinbox0.grid(row=1, column=0, padx=10, pady=5)

    inv_label = tkinter.Label(app_frame, text="SKU53982")
    inv_label.grid(row=2, column=0, padx=10, pady=5)
    income_spinbox1 = tkinter.Spinbox(app_frame, from_=f"{SKU53982}",
to="infinity")
    income_spinbox1.grid(row=3, column=0, padx=10, pady=5)
```

```
    inv_label = tkinter.Label(app_frame, text="SKU21675")
    inv_label.grid(row=4, column=0, padx=10, pady=5)
    income_spinbox2 = tkinter.Spinbox(app_frame, from_=f"{SKU21675}",
to="infinity")
    income_spinbox2.grid(row=5, column=0, padx=10, pady=5)

    inv_label = tkinter.Label(app_frame, text="SKU70148")
    inv_label.grid(row=0, column=1, padx=10, pady=5)
    income_spinbox3 = tkinter.Spinbox(app_frame, from_=f"{SKU70148}",
to="infinity")
    income_spinbox3.grid(row=1, column=1, padx=10, pady=5)

    inv_label = tkinter.Label(app_frame, text="SKU38291")
    inv_label.grid(row=2, column=1, padx=10, pady=5)
    income_spinbox4 = tkinter.Spinbox(app_frame, from_=f"{SKU38291}",
to="infinity")
    income_spinbox4.grid(row=3, column=1, padx=10, pady=5)

    inv_label = tkinter.Label(app_frame, text="SKU96574")
    inv_label.grid(row=4, column=1, padx=10, pady=5)
    income_spinbox5 = tkinter.Spinbox(app_frame, from_=f"{SKU96574}",
to="infinity")
    income_spinbox5.grid(row=5, column=1, padx=10, pady=5)

    def enter_inventory():
        """
        Updates the SKU quantities in the database based on the values
entered in the spinboxes.
        """
        revenue0 = income_spinbox0.get()
        revenue1 = income_spinbox1.get()
        revenue2 = income_spinbox2.get()
        revenue3 = income_spinbox3.get()
        revenue4 = income_spinbox4.get()
        revenue5 = income_spinbox5.get()

        # Insert Data
        data_insert_query = """UPDATE SKU_Sheet SET SKU87423 = ?,
SKU53982 = ?, SKU21675 = ?, SKU70148 = ?, SKU38291 = ?, SKU96574 = ?"""
        data_insert_tuple = (revenue0, revenue1, revenue2, revenue3,
```

```
revenue4, revenue5)
        cursor = conn.cursor()
        cursor.execute(data_insert_query, data_insert_tuple)
        conn.commit()
        conn.close()

    button = tkinter.Button(inventory_frame, text="Enter data",
command=enter_inventory)
    button.grid(row=3, column=0, sticky="news", padx=20, pady=10)


def sales_pitch_window():
    """
    Opens a new window to display the sales pitch graph.

    Retrieves the last 20 revenue entries from the database and plots
them on a graph.

    Args:
        None

    Returns:
        None
    """
    # Create a new window
    sales_window = tkinter.Toplevel(window)
    sales_window.title("View Sales Pitch")

    # Create a frame for the sales window
    sales_frame = tkinter.Frame(sales_window)
    sales_frame.pack()

    # Create a label frame for the description
    desc_frame = tkinter.LabelFrame(sales_frame, text="Description")
    desc_frame.grid(row=0, column=0, padx=20, pady=10)

    # Set the description text
    description = "View the sales graph over time.\nNOTE: The graph shows
data only for the last 20 revenue entries"
    label_description = tkinter.Label(
        desc_frame, text=description, padx=20, pady=20, justify="left"
```

```
    )
    label_description.pack()

    # Connect to the database
    conn = sqlite3.connect("data.db")
    cursor = conn.cursor()

    # Retrieve the last 20 revenue entries from the database
    cursor.execute(
        "SELECT date, revenue FROM Revenue_Sheet ORDER BY date DESC LIMIT
20"
    )
    rows = cursor.fetchall()

    # Separate the dates and revenues into separate lists
    dates, revenues = zip(*rows)

    # Close the database connection
    cursor.close()
    conn.close()

    # Convert the dates to datetime objects
    dates = [datetime.strptime(date, "%Y-%m-%d") for date in dates]

    # Plot the dates and revenues on a graph
    plt.plot(dates, revenues, marker="o", linestyle="-")

    # Customize the plot
    plt.title("Date vs Revenue")
    plt.xlabel("Date")
    plt.ylabel("Revenue")
    plt.xticks(rotation=45)
    plt.grid(True)

    # Show the plot
    plt.tight_layout()
    plt.show()


def logistics_info():
    """
```

```
    Retrieves and displays logistics information from the database.
    """
    conn = sqlite3.connect("data.db")  # Connect to the database

    # Create the Logistics table if it doesn't exist
    table_create_query = """CREATE TABLE IF NOT EXISTS Logistics
                    (firm_name STR, manager STR, contact int)
            """
    conn.execute(table_create_query)

    log_window = tkinter.Toplevel(window)  # Create a new window for
displaying logistics information
    log_window.title("Logistics Information")

    log_frame = tkinter.Frame(log_window)  # Create a frame within the
window
    log_frame.pack()

    log_frame2 = tkinter.LabelFrame(log_frame, text="Dictionary of all
suppliers")  # Create a labeled frame within the main frame
    log_frame2.grid(row=0, column=0, padx=20, pady=10)

    desc = "View all the current logistics partners for your firm."
    log_frame2desc = tkinter.Label(
        log_frame2, text=desc, padx=20, pady=20, justify="left"
    )
    log_frame2desc.pack()  # Display the description within the labeled
frame

    conn = sqlite3.connect("data.db")  # Connect to the database again
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM Logistics")  # Retrieve all rows from
the Logistics table
    rows = cursor.fetchall()

    if len(rows) == 0:  # If there are no rows in the table
        app_frame = tkinter.LabelFrame(log_frame)
        app_frame.grid(row=1, column=0, padx=20, pady=10)

        manager_label = tkinter.Label(app_frame, text="Hmmm, looks like
there are no stored contacts.\nGo back to the home page and use the 'Add
```

```
Contact'\napp to add a new contact.", padx=20, pady=20, justify="left")
        manager_label.pack()  # Display a message indicating no stored
contacts

    else:  # If there are rows in the table
        for i, row in enumerate(rows, start=1):  # Iterate over each row
            retailer, manager, contact = row

            app_frame = tkinter.LabelFrame(log_frame, text=f"{retailer}")
# Create a labeled frame for each retailer
            app_frame.grid(row=i, column=0, padx=20, pady=10)

            manager_label = tkinter.Label(app_frame, text=f"Manager Name:
{manager}")
            contact_label = tkinter.Label(
                app_frame, text=f"Contact: {str(contact)}"
            )
            manager_label.grid(row=0, column=0, padx=13, pady=5)
            contact_label.grid(row=0, column=1, padx=13, pady=5)  #
Display the manager name and contact within the labeled frame



def add_logistics():
    """
    Function to add logistics information to the database.
    """
    add_window = tkinter.Toplevel(window)
    add_window.title("Add New Supplier")
    add_frame = tkinter.Frame(add_window)
    add_frame.pack()

    add_frame2 = tkinter.LabelFrame(add_frame, text="Description")
    add_frame2.grid(row=0, column=0, padx=20, pady=10)
    add_frame2desc = tkinter.Label(
        add_frame2,
        text="Add a new contact entry in your database.\nFill in all the
details and hit the Button!",
        padx=20,
        pady=20,
        justify="left",
```

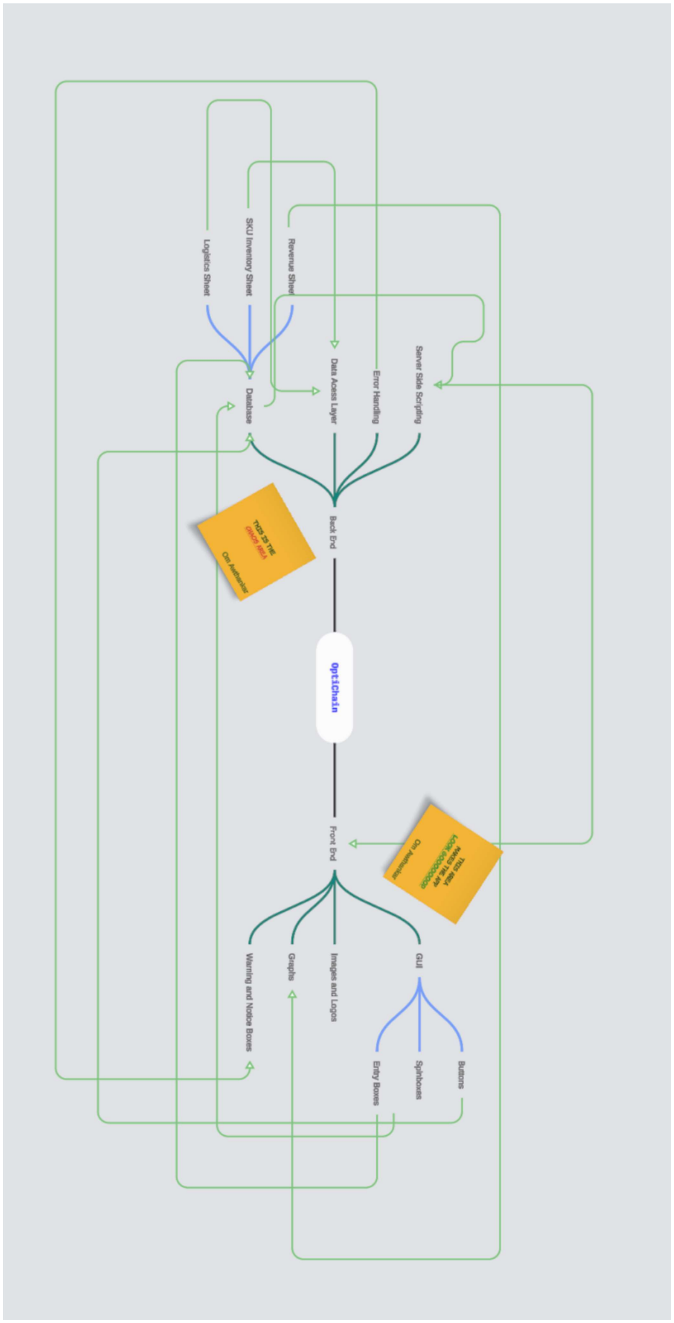
```
    )
    add_frame2desc.pack()

    add_frame3 = tkinter.LabelFrame(add_frame, text="New Contact Form")
    add_frame3.grid(row=1, column=0, padx=20, pady=20)

    retail_label = tkinter.Label(add_frame3, text="Retailer Name")
    retail_entry = tkinter.Entry(add_frame3)
    retail_label.grid(row=0, column=0, padx=10, pady=5)
    retail_entry.grid(row=0, column=1, padx=10, pady=5)

    manager_label = tkinter.Label(add_frame3, text="Manager Name")
    manager_entry = tkinter.Entry(add_frame3)
    manager_label.grid(row=1, column=0, padx=10, pady=5)
    manager_entry.grid(row=1, column=1, padx=10, pady=5)

    contact_label = tkinter.Label(add_frame3, text="Contact Number")
    contact_entry = tkinter.Entry(add_frame3)
    contact_label.grid(row=2, column=0, padx=10, pady=5)
    contact_entry.grid(row=2, column=1, padx=10, pady=5)

    def addtodb():
        """
        Function to add the entered logistics information to the
database.
        """
        try:
            retailer = retail_entry.get()
            manager = manager_entry.get()
            contact = int(contact_entry.get())

            conn = sqlite3.connect("data.db")
            table_create_query = """CREATE TABLE IF NOT EXISTS Logistics
                        (firm_name STR, manager STR, contact STR)
                """
            conn.execute(table_create_query)

            # Insert Data
            data_insert_query = """INSERT INTO Logistics (firm_name,
manager, contact) VALUES (?, ?, ?)"""
            data_insert_tuple = (retailer, manager, str(contact))
```

```
            cursor = conn.cursor()
            cursor.execute(data_insert_query, data_insert_tuple)
            conn.commit()
            conn.close()
        except ValueError:
            tkinter.messagebox.showwarning(
                "Notice", "Contact number must be a 10 digit sequence."
            )

    button = tkinter.Button(add_frame, text="Add Contact",
command=addtodb)
    button.grid(row=2, column=0, sticky="news", padx=20, pady=10)

def export():
    """
    Export data from SQLite database to an Excel file.
    """

    # Connect to the SQLite database
    conn = sqlite3.connect("data.db")

    # Create an Excel writer object
    excel_writer = pd.ExcelWriter("data.xlsx", engine="openpyxl")

    # Create a dictionary to store the dataframes for each table
    dataframes = {}

    # List of tables to export
    tables = ["Revenue_Sheet", "SKU_Sheet", "Logistics"]

    # Iterate over each table and retrieve the data as a dataframe
    for table in tables:
        dataframes[table] = pd.read_sql_query(f"SELECT * FROM {table}",
conn)

    # Write each dataframe to a separate sheet in the Excel file
    for table, df in dataframes.items():
        df.to_excel(excel_writer, sheet_name=table, index=False)

    # Close the Excel writer and the database connection
    excel_writer.close()
```

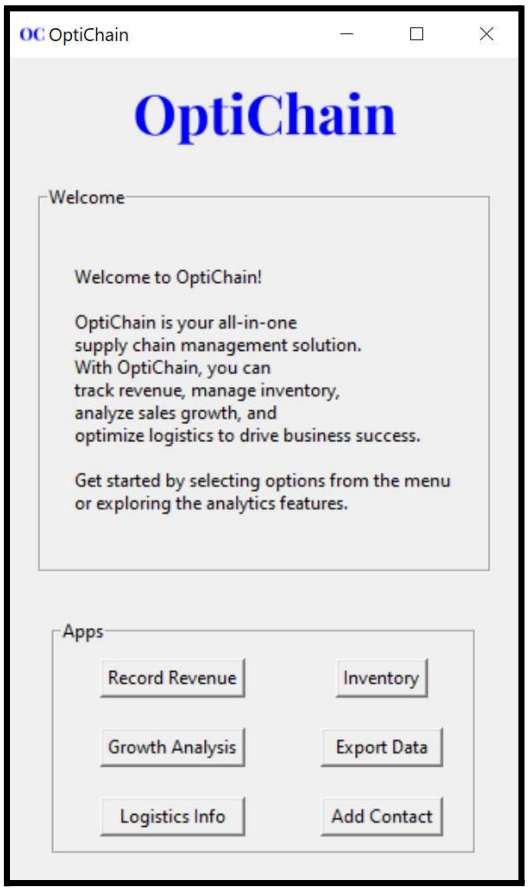
```
    conn.close()

    # Open the exported Excel file
    subprocess.Popen("data.xlsx", shell=True)

    # Show a message box indicating the export is completed
    tkinter.messagebox.showinfo("Action Completed", "Your data has been
exported successfully as 'data.xlsx'")

window = tkinter.Tk()
window.title("OptiChain")

frame = tkinter.Frame(window)
frame.pack()


welcome_frame = tkinter.LabelFrame(frame, text="Welcome")
welcome_frame.grid(row=1, column=0, padx=20, pady=10)

welcome_message = """
Welcome to OptiChain!

OptiChain is your all-in-one
supply chain management solution.
With OptiChain, you can
track revenue, manage inventory,
analyze sales growth, and
optimize logistics to drive business success.

Get started by selecting options from the menu
or exploring the analytics features.
"""

label_welcome = tkinter.Label(
    welcome_frame, text=welcome_message, padx=20, pady=20, justify="left"
)
label_welcome.pack()

features_frame = tkinter.LabelFrame(
    frame,
    text="Apps",
```

```
)
features_frame.grid(row=2, column=0, padx=5, pady=20)
button_revenue = tkinter.Button(
    features_frame, text="Record Revenue", command=record_revenue
)
button_revenue.grid(row=0, column=0, sticky="news", padx=30, pady=10)
button_inventory = tkinter.Button(
    features_frame, text="Inventory", command=inventory_window
)
button_inventory.grid(row=0, column=1, sticky="news", padx=30, pady=10)
button_sales = tkinter.Button(
    features_frame, text="Growth Analysis", command=sales_pitch_window
)
button_sales.grid(row=1, column=0, sticky="news", padx=30, pady=10)
button_logistics = tkinter.Button(
    features_frame, text="Logistics Info", command=logistics_info
)
button_logistics.grid(row=2, column=0, sticky="news", padx=30, pady=10)
button = tkinter.Button(features_frame, text="Add Contact",
command=add_logistics)
button.grid(row=2, column=1, sticky="news", padx=20, pady=10)


button = tkinter.Button(features_frame, text="Export Data",
command=export)
button.grid(row=1, column=1, sticky="news", padx=20, pady=10)


image_path = "OptiChain Res.png"
image = PhotoImage(file=image_path)


image_label = tkinter.Label(frame, image=image, padx=70, pady=10)
image_label.grid(row=0, column=0)


icon = "Icon.ico"
window.iconbitmap(False, icon)
window.mainloop()
```

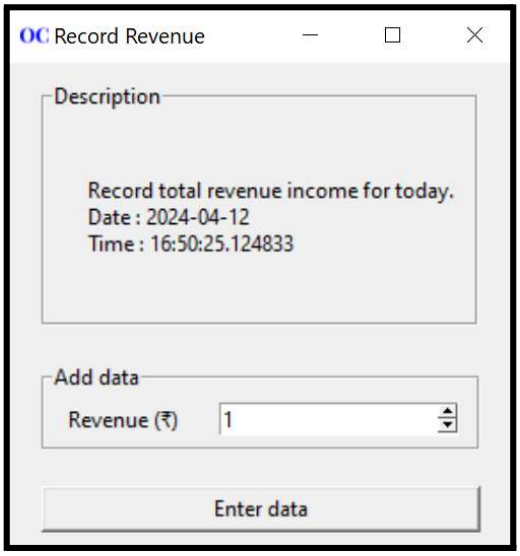
7. System Design



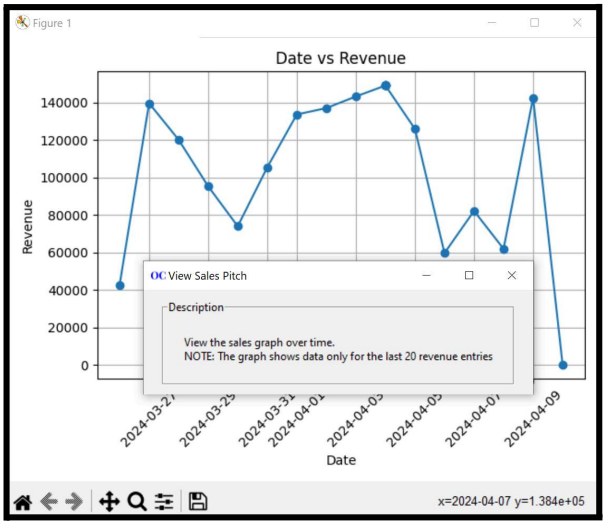
8. System Implementation



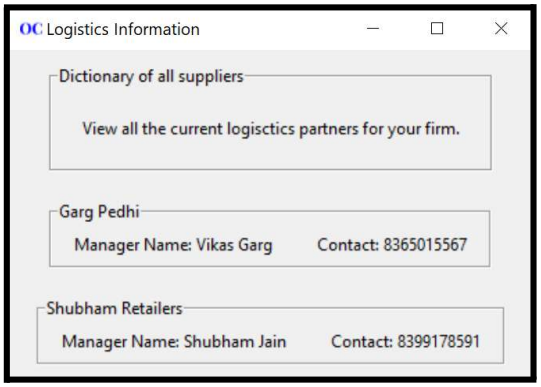
Optichain — Home Window



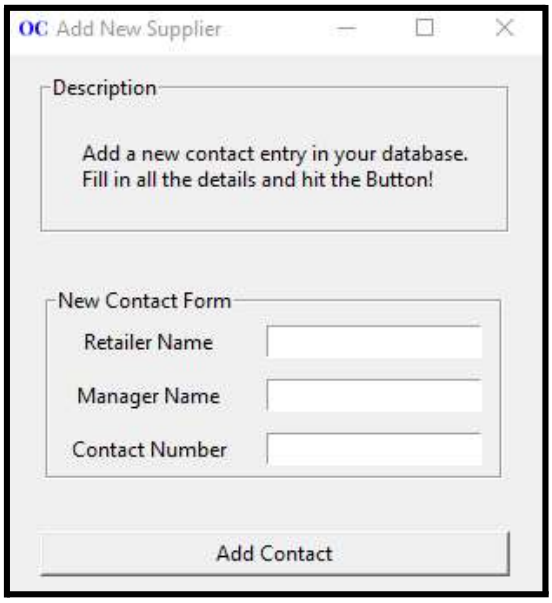
Record Revenue Window



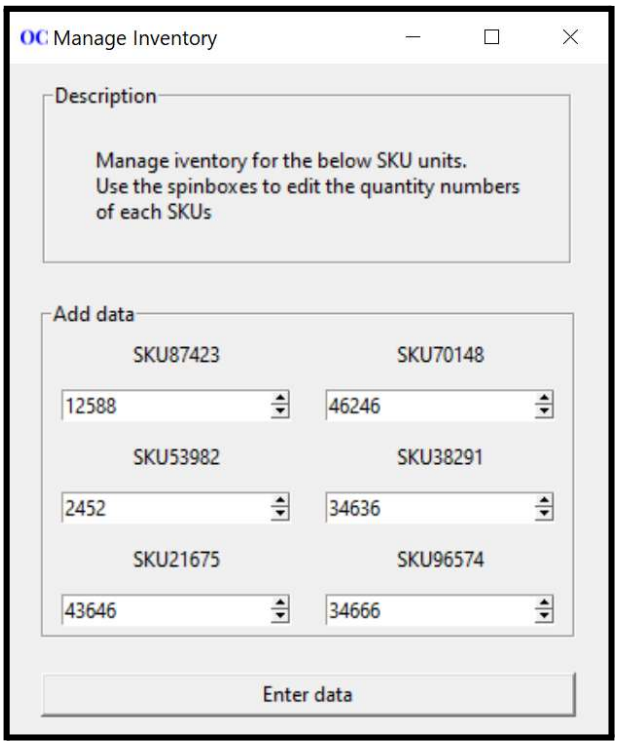
Sales Analysis Graph



Logistics Info Window



Add Logistics Partner Window



Manage Inventory Window

date	revenue
2024-04-07	82184
2024-03-04	132646
2024-03-14	104582
2024-03-03	80711
2024-04-09	142464
2024-03-24	81618
2024-03-06	45204
2024-04-01	133640
2024-03-17	73541
2024-04-03	143236
2024-03-12	72042
2024-03-29	95298
2024-04-08	61965
2024-03-31	105428
2024-03-08	55358
2024-03-05	115495
2024-03-25	104804
2024-03-19	49982
2024-03-23	127910
2024-03-09	52370

Revenue Database Sheet

SKU87423	SKU53982	SKU21675	SKU70148	SKU38291	SKU96574
12588	2452	43646	46246	34636	34666

Inventory Database Sheet

firm_name	manager	contact
Garg Pedhi	Vikas Garg	8365015567
Shubham Retailers	Shubham Jain	8399178591

Logistics Database Sheet

9. Table Designs

Table design refers to the process of defining the structure and organization of tables within a database. This involves specifying the columns (fields) each table will have, the data types for each column, and the relationships between tables.

Below are the table designs for the databases in Optichain.

a. Revenue_Sheet Table Design

Column Name	Data Type	Description
date	TEXT	Date of the revenue entry
revenue	INTEGER	Total revenue of the date

b. SKU_Sheet Table Design

Column Name	Data Type	Description
SKU87423	INTEGER	Quantity of SKU item 87423
SKU53982	INTEGER	Quantity of SKU item 53982
SKU21675	INTEGER	Quantity of SKU item 21675
SKU70148	INTEGER	Quantity of SKU item 70148
SKU38291	INTEGER	Quantity of SKU item 38291
SKU96574	INTEGER	Quantity of SKU item 96574

c. Logistics Table Design

Column Name	Data Type	Description
firm_name	TEXT	Name of the firm
manager	TEXT	Name of the manager
contact	TEXT	Contact no. of the manager

10. Algorithm and Flowcharts

a. Initialize the Application

i. Import necessary libraries.
ii. Create the main window for the application.

b. Define Functions for Various Features

i. record_revenue: Function to record daily revenue.
    1. Create a new window for revenue entry.
    2. Check if today's revenue is already recorded.
    3. Create UI elements for revenue entry.
    4. Define enter_revenue to insert data into the database.
ii. inventory_window: Function to manage inventory.
    1. Create a new window for inventory management.
    2. Create UI elements for displaying and updating SKU quantities.
    3. Define enter_inventory to update SKU data in the database.
iii. sales_pitch_window: Function to display sales pitch graph.
    1. Create a new window for the sales graph.
    2. Retrieve and plot the last 20 revenue entries.
iv. logistics_info: Function to display logistics information.
    1. Create a new window for logistics info.

2. Retrieve and display logistics data from the database.

v. add_logistics: Function to add new logistics contacts.
   1. Create a new window for adding contacts.
   2. Create UI elements for entering contact details.
   3. Define addtodb to insert new contact into the database.

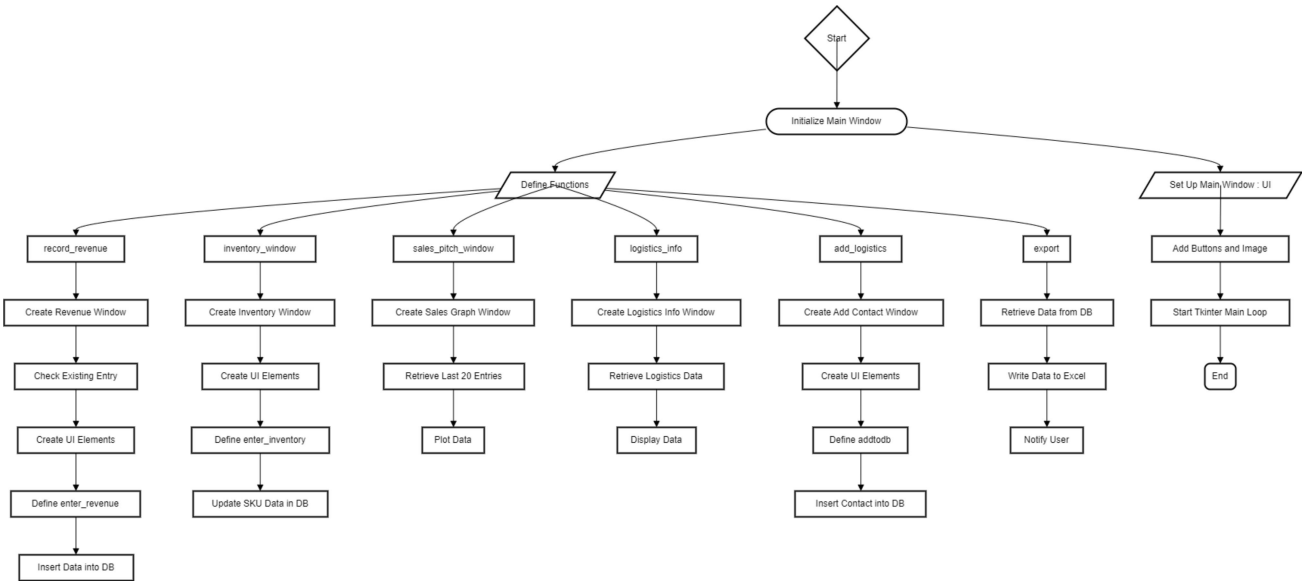
vi. export: Function to export data to an Excel file.
   1. Retrieve data from all tables.
   2. Write data to an Excel file.
   3. Notify users of successful exports.

c. Set Up the Main Window UI

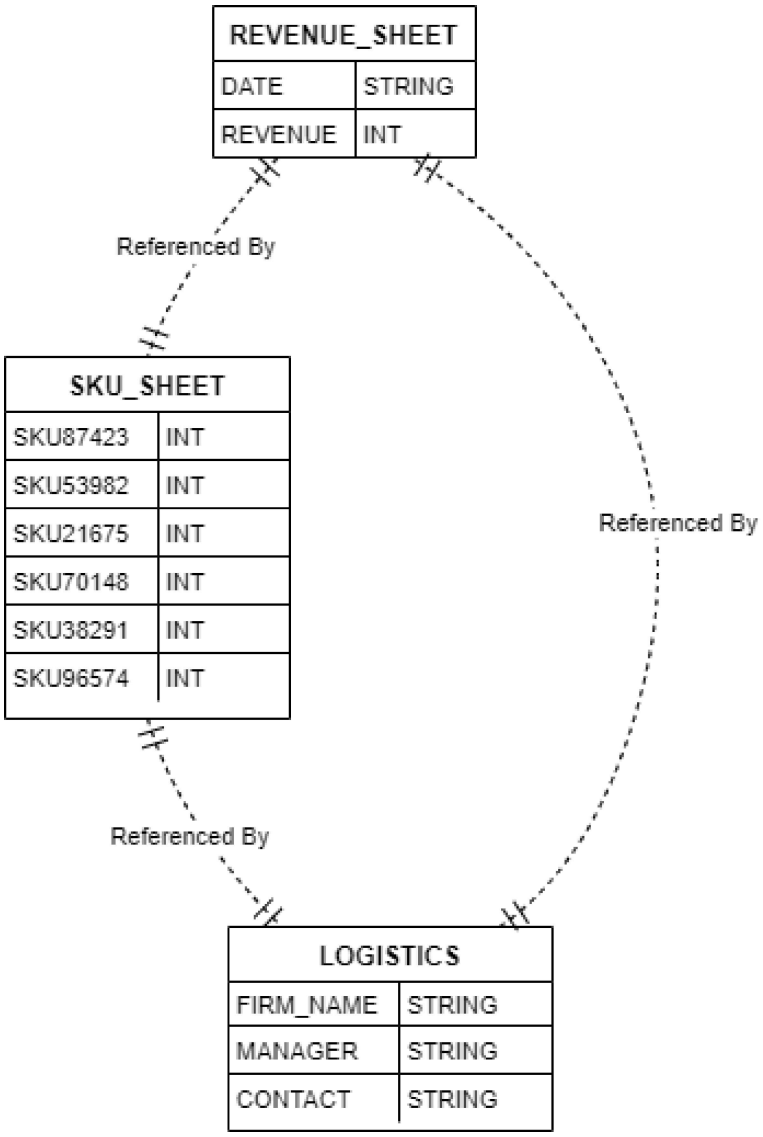
i. Create welcome and feature frames.
ii. Add buttons for each feature (Record Revenue, Inventory, Growth Analysis, Logistics Info, Add Contact, Export Data).
iii. Add an image to the main window.
iv. Set the application icon.

d. Run the Application

i. Start the Tkinter main event loop.



11. Entity Relationship Diagram



The ER (Entity-Relationship) diagram provided depicts the relationships and entities involved in a system that manages various aspects of a supply chain, including revenue recording, inventory management, sales pitch analysis, logistics information, and data export. Here is a detailed explanation of the ER diagram.
All components of the ER Diagram have been covered above under the title **"Project Scope"** please kindly refer to that