KENDRIYA VIDYALAYA BAMANGACHI

Session 2024-2025

A project report on

BILLING SYSTEM

NAME : HEM CHANDRA PAL
CLASS : XII COMMERCE
ROII.NO : 17

CERTIFICATE

This is to certify that **Hem Chandra Pal**, a student of **Class XII-Commerce**, Roll No: **17**, has prepared this report on the project entitled **"billing-system"**.

This report is the result of his sincere efforts and endeavors. It is found worthy of acceptance as the final report for the subject Informatics Practices of Class XII.

He has prepared this report under my guidance.

Mr. Sunil Kumar
PGT (Informatics Practices)

Mr. Samsher Kumar Singh
Principal

External Examiner

ACKNOWLEDGEMENT

I sincerely thank my Information Practices teacher, Mr. Sunil Kumar, for his invaluable guidance and support throughout this project.
I am deeply grateful to my parents and sister for their constant encouragement and motivation during the course of this work. I also extend my heartfelt thanks to my friends for their assistance with various aspects of the project.
Lastly, I would like to express my gratitude to CBSE for providing me with this wonderful opportunity to undertake this project and enhance my learning experience.

INDEX

PROBLEM ANALYSIS

1. **Automated Invoice Generation:** The billing system eliminates the manual process of creating invoices, ensuring accuracy and saving time.

2. **Tax Calculation and Compliance:** It automatically calculates taxes such as GST or VAT, ensuring adherence to regulatory requirements and reducing the chances of manual errors.

3. **Customer Management:** The system stores customer information, including purchase history and contact details, enabling personalized services and efficient record-keeping.

4. **Discount and Offer Application:** Discounts and promotional offers are applied accurately, enhancing customer satisfaction and boosting sales.

5. **Real-Time Inventory Updates:** Automatically tracks stock levels after each transaction, helping businesses avoid overstocking or stock shortages.

6. **Error-Free Transactions:** Automates calculations for totals, taxes, and discounts, minimizing human errors and enhancing customer trust.

7. **User-Friendly Interface:** Features a simplified and intuitive interface that ensures ease of use for staff, reducing training time and boosting operational efficiency.

8. **Data Security and Backup:** Protects sensitive business and customer data through secure storage and periodic backups, ensuring data integrity.

9. **Multi-Platform Accessibility:** Operates seamlessly across desktops, tablets, and mobile devices, enabling flexible and remote access to billing operations.

10. **Scalability and Adaptability:** Handles increased transaction volumes and integrates with additional modules, making it suitable for business growth.

REQUIREMENTS ANALYSIS

1) HARDWARE REQUIREMENTS

> COMPUTER FOR CODING AND TYPING THE REQUIRED DOCUMENTS OF THE PROJECT

> PRINTER TO PRINT THE REQUIRED DOCUMENTS OF THE PROJECT

> COMPACT DRIVE

> PROCESSOR : PENTIUM QUAD CORE

> RAM : 2 GB

> HARD DISK : 20 GB

2) **SOFTWARE ANALYSIS**

> OPERATING SYSTEM: WINDOWS 10 OR LATER , LINUX ANY DESTRO

> PYTHON: VERSION 3.X (FOR PROGRAM EXECUTION)

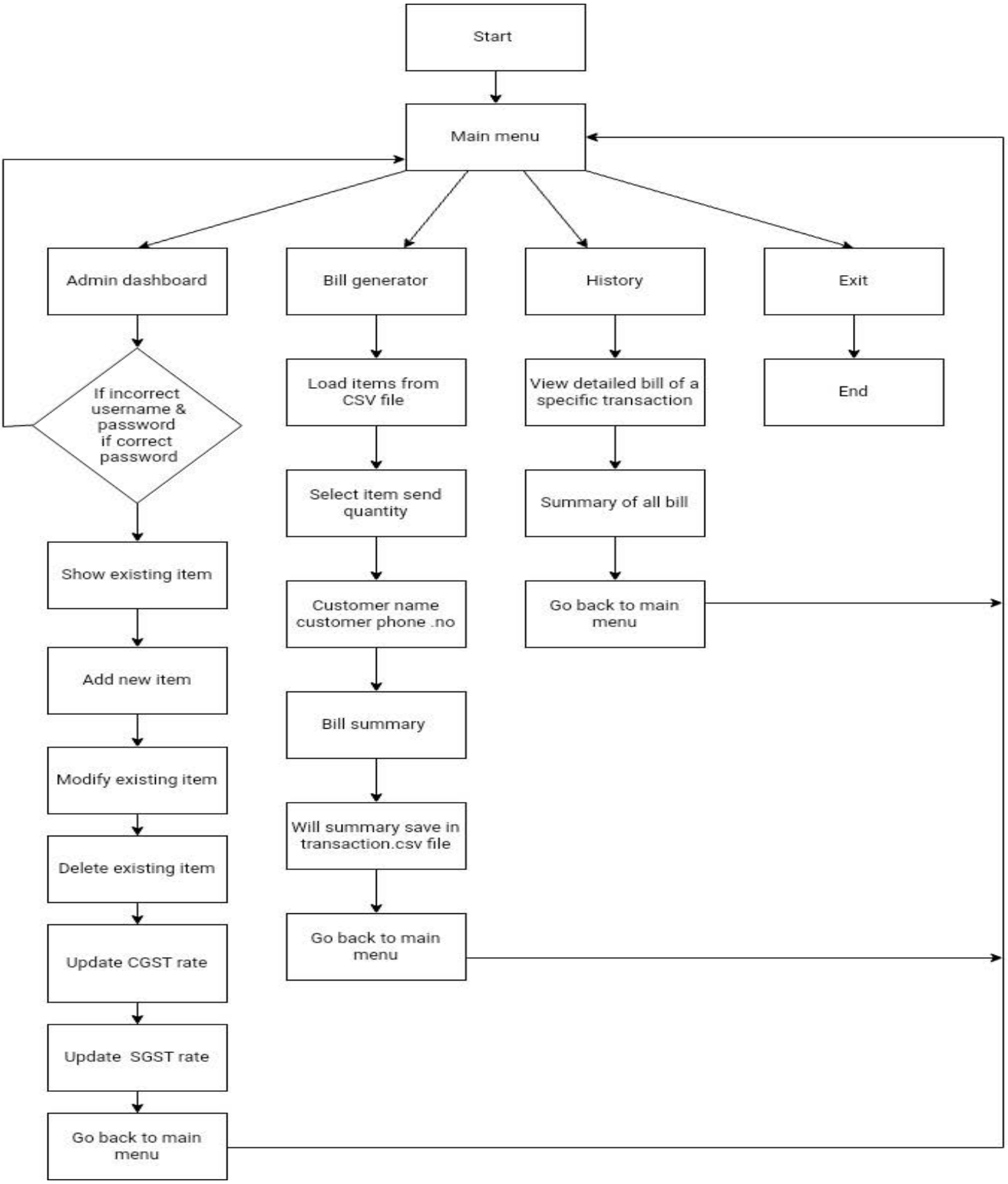
> CSV FILE SUPPORT: REQUIRED FOR DATA STORAGE AND MANAGEMENT

> TEXT EDITOR/IDE: VS CODE, PYCHARM, OR ANY PYTHON-COMPATIBLE Editor For Writing And Editing The Code

> PYTHON LIBRARIES: BUILT-IN LIBRARIES AND ANY OTHERS USED IN THE PROJECT

> MICROSOFT WORD (OPTIONAL): FOR CREATING AND PRESENTING DOCUMENTATION OR REPORTS

WORKFLOW


Start
Main menu
Admin dashboard
Bill generator
History
Exit
If incorrect username & password if correct password
Load items from CSV file
View detailed bill of a specific transaction
End
Select item send quantity
Summary of all bill
Show existing item
Customer name customer phone .no
Go back to main menu
Add new item
Bill summary
Modify existing item
Will summary save in transaction.csv file
Delete existing item
Go back to main menu
Update CGST rate
Update SGST rate
Go back to main menu

SOURCE CODE

```
import os
import csv
from datetime import datetime

# File paths
ITEMS_FILE = "items.csv"
TRANSACTIONS_FILE = "transactions.csv"

# Initialize CSV files
def initialize_files():
    try:
        with open(ITEMS_FILE, mode="x", newline="") as file:
            writer = csv.writer(file)
            writer.writerow(["ID", "Name", "Price"])
    except FileExistsError:
        pass

    try:
        with open(TRANSACTIONS_FILE, mode="x", newline="") as file:
            writer = csv.writer(file)
            writer.writerow(["Bill No", "Date", "Time", "Transaction Type", "Total
Amount", "Details"])
    except FileExistsError:
        pass

# Clear screen
def clear_screen():
    os.system('cls' if os.name == 'nt' else 'clear')
```

```
# Read CSV file
def read_csv(file_path):
    try:
        with open(file_path, mode="r") as file:
            reader = csv.reader(file)
            return list(reader)
    except FileNotFoundError:
        print(f"Error: {file_path} not found. Please initialize the system.")
        return []

# Write CSV file
def write_csv(file_path, rows, mode="w"):
    try:
        with open(file_path, mode=mode, newline="") as file:
            writer = csv.writer(file)
            writer.writerows(rows)
    except Exception as e:
        print(f"Error writing to {file_path}: {e}")

# Main Menu
def main_menu():
    while True:
        clear_screen()
        print("Main Menu")
        print("1. Admin Dashboard")
        print("2. Bill Generator")
        print("3. History")
        print("4. Exit")
        choice = input("Select an option: ").strip()
```

```
if choice == "1":
            admin_dashboard()
        elif choice == "2":
            bill_generator()
        elif choice == "3":
            view_history()
        elif choice == "4":
            print("Exiting the system...")
            break
        else:
            print("Invalid option. Please try again.")

# Admin Dashboard
def admin_dashboard():
    clear_screen()
    username = input("Username: ")
    password = input("Password: ")

    if username == "admin" and password == "admin":
        print("Login Successful!")
        input("Press Enter to continue to the Admin Panel...")
        admin_panel()
    else:
        print("Incorrect username or password. Try again.")
        input("Press Enter to return to the main menu...")

# Admin Panel
def admin_panel():
    global cgst_rate, sgst_rate
    while True:
```

```
clear_screen()
        print("Admin Dashboard")
        print(f"CGST Rate: {cgst_rate}% | SGST Rate: {sgst_rate}%")
        print("1. Show Existing Items")
        print("2. Add New Item")
        print("3. Modify Existing Item")
        print("4. Delete Existing Item")
        print("5. Update CGST Rate")
        print("6. Update SGST Rate")
        print("7. Go Back to Main Menu")
        choice = input("Select an option: ").strip()

        if choice == "1":
            show_existing_items()
        elif choice == "2":
            add_new_item()
        elif choice == "3":
            modify_existing_item()
        elif choice == "4":
            delete_existing_item()
        elif choice == "5":
            update_cgst_rate()
        elif choice == "6":
            update_sgst_rate()
        elif choice == "7":
            break
        else:
            print("Invalid option. Please try again.")

# Show Existing Items
def show_existing_items():
    clear_screen()
```

```
items = read_csv(ITEMS_FILE)
    if len(items) <= 1:
        print("No items found.")
    else:
        print("Existing Items:")
        for item in items[1:]:  # Skip the header row
            print(f"ID: {item[0]}, Name: {item[1]}, Price: {item[2]}")
    input("Press Enter to return to the admin dashboard...")

# Modify Existing Item
def modify_existing_item():
    clear_screen()
    items = read_csv(ITEMS_FILE)

    if len(items) <= 1:
        print("No items found.")
        input("Press Enter to return to the admin dashboard...")
        return

    print("Existing Items:")
    for item in items[1:]:  # Skip the header row
        print(f"ID: {item[0]}, Name: {item[1]}, Price: {item[2]}")

    item_id = input("Enter the ID of the item you want to modify: ").strip()

    # Find item to modify
    for i, item in enumerate(items[1:], start=1):  # Skip header row
        if item[0] == item_id:
            print(f"Selected Item: ID: {item[0]}, Name: {item[1]}, Price: {item[2]}")
            new_name = input(f"Enter new name (leave blank to keep '{item[1]}'):
").strip()
            new_price = input(f"Enter new price (leave blank to keep '{item[2]}'):
").strip()
```

```
# Skip header row
        if item[0] == item_id:
            print("Selected Item: ID: {}, Name: {}, Price: {}".format(item[0], item[1],
item[2]))
            confirm = input("Are you sure you want to delete this item? (yes/no):
").strip().lower()
            if confirm == "yes":
                del items[i]  # Remove the selected item
                write_csv(ITEMS_FILE, items)
                print("Item deleted successfully.")
            else:
                print("Deletion canceled.")
            input("Press Enter to return to the admin dashboard...")
            return

    print("Item ID not found. No items were deleted.")
    input("Press Enter to return to the admin dashboard...")

# Add New Item
def add_new_item():
    clear_screen()
    items = read_csv(ITEMS_FILE)
    new_id = len(items)
    name = input("Enter item name: ").strip()
    price = input("Enter item price: ").strip()
    items.append([new_id, name, price])
    write_csv(ITEMS_FILE, items)
    print(f"Item '{name}' added successfully.")
    input("Press Enter to return to the admin dashboard...")

#Bill Generator function
def bill_generator():
```

```
clear_screen()

    # Read items from the items file
    items = read_csv(ITEMS_FILE)
    if len(items) <= 1:
        print("No items available for billing.")
        input("Press Enter to return to the main menu...")
        return

    # Display available items
    print("Available Items:")
    for item in items[1:]:  # Skip header row
        print(f"ID: {item[0]}, Name: {item[1]}, Price: {item[2]}")

    # Cart and item selection
    cart = []
    while True:
        item_id = input("Enter item ID to add to the bill (or 'done' to finish):
").strip()
        if item_id.lower() == 'done':
            break

        quantity = input("Enter quantity: ").strip()
        for item in items[1:]:
            if item[0] == item_id:
                cart.append((item[1], float(item[2]), int(quantity)))
                print(f"Added {quantity} x {item[1]} to the cart.")
                break
        else:
            print("Invalid item ID. Please try again.")
```

```
if not cart:
        print("No items added to the bill.")
        input("Press Enter to return to the main menu...")
        return

    # Get customer details
    customer_name = input("Enter customer's name: ").strip()
    customer_phone = input("Enter customer's phone number: ").strip()

    # Get discount (if any)
    discount_input = input("Enter discount (in % or 'n' for no discount): ").strip()
    discount = 0
    if discount_input.lower() != 'n':
        try:
            discount = float(discount_input)
        except ValueError:
            print("Invalid discount value. No discount will be applied.")

    # Generate bill summary
    subtotal = sum(price * qty for _, price, qty in cart)
    cgst = subtotal * cgst_rate / 100
    sgst = subtotal * sgst_rate / 100
    discount_amount = subtotal * discount / 100
    grand_total = subtotal + cgst + sgst - discount_amount

# Clear screen and print bill summary
    clear_screen()
    bill_no = len(read_csv(TRANSACTIONS_FILE))  # Generate Bill No.
```

```
print("--- Bill Summary ---")
    print(f"Bill No. {bill_no}")
    print(f"Time: {datetime.now().strftime('%H:%M:%S')}")
    print(f"Date: {datetime.now().strftime('%d-%m-%Y')}")
    print(f"Customer Name: {customer_name}")
    print(f"Phone Number: {customer_phone}")

    for name, price, qty in cart:
        print(f"{qty} x {name} @ {price:.2f} = {price * qty:.2f}")

    print(f"Subtotal: {subtotal:.2f}")
    print(f"CGST ({cgst_rate}%): {cgst:.2f}")
    print(f"SGST ({sgst_rate}%): {sgst:.2f}")
    if discount > 0:
        print(f"Discount: {discount}% = {discount_amount:.2f}")
    else:
        print("Discount: No discount available")
    print(f"Grand Total: {grand_total:.2f}")

# Save the transaction to the file
    transactions = read_csv(TRANSACTIONS_FILE)
    transaction_details = [{"Item": name, "Price": price, "Quantity": qty} for name,
price, qty in cart]
    transactions.append([
         bill_no,
        datetime.now().strftime("%Y-%m-%d"),
        datetime.now().strftime("%H:%M:%S"),
        "Sale",
        f"{grand_total:.2f}",
        str(transaction_details),
        customer_name,
        customer_phone ])
```

```
write_csv(TRANSACTIONS_FILE, transactions)

    print("Generated and saved successfully!")
    input("Press Enter to return to the main menu...")

# Clear Screen Function
def clear_screen():
    import os
    os.system('cls' if os.name == 'nt' else 'clear')

# View History Function
def view_history():
    clear_screen()
    transactions = read_csv(TRANSACTIONS_FILE)

    if len(transactions) <= 1:
        print("No transaction history found.")
        input("Press Enter to return to the main menu...")
    else:
        while True:
            clear_screen()
            print("Transaction History")
            print("1. View Detailed Bill of a Specific Transaction")
            print("2. View Summary of All Bills")
            print("3. Return to Main Menu")
            choice = input("Select an option: ").strip()

if choice == "1":
                while True:
                    clear_screen()
```

```
# Show list of all bills before selecting one
            print("---- Available Bills ----")
            print("{:<10} {:<15} {:<15} {:<20} {:<10}".format("Bill No", "Date",
"Time", "Customer Name", "Total Amount"))
            print("-" * 70)
            for row in transactions[1:]:
                print(f"{row[0]:<10} {row[1]:<15} {row[2]:<15} {row[6]:<20}
{row[4]:<10}")
            print("-" * 70)

            bill_no = input("Enter the Bill No to view full details (or type 'exit'
to go back): ").strip()
            if bill_no.lower() == "exit":
                break

            # Display details of the selected bill
            for row in transactions[1:]:
                if row[0] == bill_no:
                    clear_screen()
                    print(f"---- Detailed Bill for Bill No. {row[0]} ----")
                    print(f"Date: {row[1]}")
                    print(f"Time: {row[2]}")
                    print(f"Customer Name: {row[6]}")
                    print(f"Transaction Type: {row[3]}")
                    print("\n--- Purchased Items ---")

                    details = eval(row[5]) # Convert string representation of list
to actual list
```

```
subtotal = 0
                    total_cgst = 0
                    total_sgst = 0

                    for idx, item in enumerate(details, start=1):
                        product_total = item['Price'] * item['Quantity']
                        item_cgst = product_total * cgst_rate / 100
                        item_sgst = product_total * sgst_rate / 100
                        subtotal += product_total
                        total_cgst += item_cgst
                        total_sgst += item_sgst

                        print(f"{idx}. {item['Item']} | Price: {item['Price']:.2f} |
Quantity: {item['Quantity']}")
                        print(f"   Product price x quantity: {item['Price']} x
{item['Quantity']} = {product_total:.2f}")
                        print(f"   GST (CGST + SGST): {item_cgst:.2f} +
{item_sgst:.2f} = {item_cgst + item_sgst:.2f}")
                        print(f"   Total (with GST): {product_total + item_cgst +
item_sgst:.2f}\n")

                    grand_total = subtotal + total_cgst + total_sgst

                    print("--- Bill Summary ---")
                    print(f"Subtotal: {subtotal:.2f}")
                    print(f"CGST ({cgst_rate}%): {total_cgst:.2f}")
                    print(f"SGST ({sgst_rate}%): {total_sgst:.2f}")
                    print(f"Grand Total: {grand_total:.2f}")

                    input("\nPress Enter to return to the history menu...")
                    break
                else:
```

```
print("Bill No not found. Please try again.")
                    input("Press Enter to continue...")
        elif choice == "2":
            # Display summary of all bills
            clear_screen()
            print("---- Bill Summary ----")
            print("{:<10} {:<15} {:<15} {:<20} {:<10}".format("Bill No", "Date",
"Time", "Customer Name", "Total Amount"))
            print("-" * 70)
            for row in transactions[1:]:
                print(f"{row[0]:<10} {row[1]:<15} {row[2]:<15} {row[6]:<20}
{row[4]:<10}")
            print("-" * 70)
            input("\nPress Enter to return to the history menu...")

        elif choice == "3":
            break
        else:
            print("Invalid option. Please try again.")

# Update CGST Rate
def update_cgst_rate():
    global cgst_rate
    try:
        cgst_rate = float(input(f"Enter new CGST rate (current: {cgst_rate}%):
").strip())
        print("CGST rate updated successfully.")
    except ValueError:
        print("Invalid rate. No changes made.")
    input("Press Enter to return to the admin dashboard...")
```

```
# Update SGST Rate
def update_sgst_rate():
    global sgst_rate
    try:
        sgst_rate = float(input(f"Enter new SGST rate (current: {sgst_rate}%):
").strip())
        print("SGST rate updated successfully.")
    except ValueError:
        print("Invalid rate. No changes made.")
    input("Press Enter to return to the admin dashboard...")

# Global variables for CGST and SGST rates
cgst_rate = 6.0
sgst_rate = 6.0

# Main execution
if __name__ == "__main__":
    initialize_files()
    main_menu()
```

Testing

FUNCTIONAL TESTING:

• Verify that basic functions work correctly, such as adding and deleting items, generating bills, applying discounts, calculating taxes, and viewing transaction history.

USER INTERFACE TESTING:

• Ensure that the user interface is intuitive and user-friendly.

• Test the navigation, menu options, and responsiveness of the system for seamless operation.

BILLING PROCESS:

• Test the process of generating bills, ensuring accuracy in calculating totals, applying discounts, and adding CGST/SGST.

• Verify that all transactions are recorded properly in the history for future reference.

USER MANAGEMENT:

• Validate the admin login functionality to ensure only authorized users can access sensitive operations like modifying items or tax rates.

TRANSACTION HISTORY:

• Test the system’s ability to display a detailed summary of past transactions.

• Verify that all recorded bills are complete and match their respective transaction details.

ITEM MANAGEMENT:

• Test the system's ability to handle item management, including adding new items, updating item details (e.g., name, price), and deleting items from the list.

SEARCH AND RETRIEVAL:

• Test the ability to search for items based on their ID or name to ensure accurate and efficient retrieval of item details.

RECOVERY AND MAINTENANANCE

RECOVERY:

DATA BACKUP:

- Regularly back up the system's critical files, including items.csv and transactions.csv.
- Since the program doesn't automate backups, take manual backups by copying these files to a secure location.

DISASTER RECOVERY PLAN:

- Develop a simple plan to restore functionality by replacing corrupted or lost files (items.csv and transactions.csv) with backed-up versions.
- Keep a separate, labeled backup folder to organize backups effectively.

VERSION CONTROL:

- Maintain a copy of your program script and CSV files in a versioned folder or repository.
- Manually save stable versions of the system to revert easily if errors occur after changes.

REDUNDANCY AND FAILOVER:

- To minimize downtime, keep redundant copies of the key files (items.csv and transactions.csv) on external storage or cloud services.
- Ensure the backups are accessible for quick restoration.

MONITORING AND ALERTS:

- Periodically review the CSV files to ensure data integrity and check for issues like missing or incorrect records.
- Set reminders for manual backups to avoid unintentional data loss.

MAINTENANCE:

BUG TRACKING AND RESOLUTION:

- Log and track issues in the program, such as incorrect data entries or crashes during execution.
- Address and resolve bugs promptly to maintain system stability.

SECURITY UPDATES:

- While the program lacks robust database security, restrict access to critical files (e.g., items.csv, transactions.csv) by limiting file permissions to authorized users.

PERFORMANCE OPTIMIZATION:

- Periodically review and optimize the code for efficiency, such as simplifying redundant logic and improving file handling.
- Remove unnecessary operations that may slow down the program.

PERFORMANCE OPTIMIZATION:

• Periodically review and optimize the code for efficiency, such as simplifying redundant logic and improving file handling.
• Remove unnecessary operations that may slow down the program.

USER FEEDBACK AND FEATURE REQUESTS:

• Collect feedback from users (e.g., administrators) on the ease of use and potential improvements.
• Implement enhancements like better menu options, improved reporting, or easier item management.

DOCUMENTATION UPDATES:

• Keep documentation updated, including instructions for manual backup, restoration, and program usage.
• Ensure any program changes are reflected in the documentation for easy reference.

Implications

1)REQUIREMENTS ANALYSIS:

• Identify Key Requirements: The billing system must manage items, process transactions, and maintain a history of bills.
• Understand Workflow: Key workflows include item addition, price modification, bill generation, applying discounts, calculating taxes, and storing transaction logs for future reference.

2) DESIGN:

• Database Schema: Create a structure to store data about items (e.g., ID, name, price), transactions (e.g., bill number, total amount, and itemized details), and tax rates (CGST, SGST).
• User Interface: Design a menu-driven interface with modules for the admin dashboard, bill generation, and transaction history viewing.
• System Architecture: Maintain a desktop-based system using Python's command-line interface.

3) DEVELOPMENT:

- Programming Language and Framework: Use Python for both backend and user interface development.
- Backend Logic: Implement item management, bill generation, tax calculations, and discount handling.
- Frontend Components: Provide a clear command-line interface for administrators to manage items and transactions.
- Security Measures: Ensure restricted access with admin login credentials.

4) DATABASE IMPLEMENTATION:

- Database Setup: Use CSV files to store item details, transactions, and tax rates.
- CRUD Operations: Implement operations to create, read, update, and delete items and transactions through the admin interface.

5) INTEGRATION OF MODULES:

- Integrate the item management and billing modules to ensure seamless operations, such as reflecting item changes in generated bills.
- Link transaction history to bill generation to allow viewing of detailed past transactions.

6) USER AUTHENTICATION AND AUTHORIZATION:

- Implement a secure login system for administrators.
- Restrict access to sensitive operations like item modification and tax rate updates.

7) TESTING:

- Unit Testing: Test individual modules such as item management, bill generation, and history viewing.
- Integration Testing: Verify the seamless interaction between item updates, billing, and transaction logging.
- User Acceptance Testing: Collect feedback from administrators to refine the system and improve usability.

Conclusion

The Billing System streamlines billing operations and ensures accurate transaction records using CSV files for efficient data management without complex databases. Its intuitive Admin Dashboard supports inventory management, including adding, modifying, deleting items, and updating CGST and SGST tax rates.

The system automates tax calculations, generates precise bills, and provides detailed transaction summaries for transparency. Its transaction history feature enables easy review and analysis of past bills and reports.

Designed for small to medium-sized businesses, the user-friendly Billing System reduces errors, streamlines tasks, and allows businesses to focus on exceptional customer service.

BIBLOGRAPHY

We gathered information for our project report from the following sources:

Reference Books:

1. Informatics Practices by Sumita Arora
2. Complete Reference with Python
3. Data Analysis with Python
4. NCERT Textbook

Websites:

- https://www.python.org
- https://www.google.com
- https://www.cbseacademic.nic.in
- https://www.kaggle.com
- https://www.geeksforgeeks.org
- https://www.w3schools.com/python/
- https://realpython.com