

CONTENT

CHAPTER	TITLE	PAGE.NO
1.	ABSTRACT	
2.	SYSTEM ANALYSIS <ul style="list-style-type: none">• EXISTING SYSTEM• PROPOSED SYSTEM	
3.	PROJECT DESCRIPTION	
4.	SYSTEM REQUIREMENTS <ul style="list-style-type: none">• HARDWARE REQUIREMENTS• SOFTWARE REQUIREMENTS	
5.	SOFTWARE DESCRIPTION	
6.	SYSTEM DESIGN <ul style="list-style-type: none">• DATA FLOW DIAGRAM	
7.	SOURCE CODE	
8.	SCREEN LAYOUTS	
9.	SYSTEM TESTING <ul style="list-style-type: none">• BLACK BOX• WHITE TEST• UNIT TEST• INTEGRATION• VERIFICATION• VALIDATION	
10.	CONCLUSION	
11.	BIBLIOGRAPHY	

ABSTRACT

The **Fuel station management system** is a software solution designed to streamline and automate the billing process at fuel stations. The system is developed using Python and Tkinter, providing a user-friendly graphical interface for efficient customer transactions. It allows the Fuel station staff to input the fuel quantity, and price, calculating the total cost for customers.

Key features include real-time price updates, automatic calculation of total charges, a simple interface for easy operation, and the ability to generate detailed bills. This system also maintains a record of transactions, ensuring that both customers and station owners can track purchases efficiently. The solution helps reduce manual errors, enhance customer service, and optimize operational workflows at petrol stations.

Overall, the system aims to increase productivity and accuracy, providing a smooth experience for both users and administrators.

SYSTEM ANALYSIS

EXISTING SYSTEM

The existing system in most Fuel station relies heavily on manual processes and paper-based billing methods, which often lead to inefficiencies, errors, and delays in service.

Drawbacks of Existing Systems

1. Manual Billing Process:

The staff then manually generates a receipt or bill for the customer, which can lead to mistakes, delays, or lack of clarity.

2. Record-Keeping:

Fuel consumption and sales data may not be updated in real time, making it difficult for the management to monitor performance, inventory, or daily sales accurately.

3. Limited Customer Interaction Features:

The existing systems often lack customer-oriented features such as loyalty programs, discounts, or easy tracking of purchase history.

PROPOSED SYSTEM

The proposed Fuel station management System using Python Tkinter aims to automate and streamline the entire billing process at petrol stations, replacing the existing manual and paper-based methods with a more efficient, user-friendly digital solution. The system is designed to minimize errors, improve speed, enhance customer experience, and provide better inventory and transaction management. Here's an outline of the key features and improvements in the proposed system:

1. User-Friendly Interface
2. Automated Billing
3. Transaction and Billing Records

PROJECT DESCRIPTION

PROJECT DESCRIPTION

The **Fuel station management System** is an automated software application designed to simplify the billing process at fuel stations, allowing Fuel station staff to efficiently process fuel transactions with accuracy. Built using **Python** and the **Tkinter** library for the graphical user interface (GUI), this system aims to replace traditional, manual billing methods with a more efficient, error-free digital solution. The system automates key operations such as fuel price calculation, transaction logging, and inventory management, reducing manual work and improving service quality.

Scope of the Project:

The system is intended to streamline the operations at a Fuel station by automating the fuel billing process, tracking transactions, managing inventory, and providing reports.

Key Features of the System:

1. User Interface:

- The system's graphical interface, built with Tkinter, provides a clean and simple user experience, allowing staff to easily navigate through the various functionalities such as selecting entering quantities, and generating bills.
- Input fields are designed to capture staff Id, fuel rate, vehicle type, and quantity. Clicking on the 'calculate' button will generate the amount to be paid.

2. Automated Billing:

- When a customer purchases fuel, the system calculates the total cost by multiplying the quantity of fuel with the current price of the fuel. This eliminates the need for manual price calculations, reducing errors and ensuring accuracy.
- The system generates a bill automatically, detailing the fuel quantity, price per liter, and total cost.

3. Real-Time Price Updates:

- The fuel billing information are stored in the system and can be easily updated by authorized personnel (e.g., station managers) in real-time. This ensures that customers are always charged the correct price.
- Price updates are reflected immediately in the billing process, ensuring smooth transactions

4. Transaction Logging:

- Every transaction is automatically recorded in the system, storing information such as the staff Id, customer's fuel purchase, transaction date, quantity, and total cost. These records help in tracking daily sales for that particular staff and monitoring business performance.
- The system maintains a history of all completed transactions, making it easy for management to retrieve and review past sales.

5. Backup and Data Recovery:

- The system is designed to perform regular data backups, ensuring that transaction records and customer details are not lost in case of system failure or crashes.
- Backup features ensure smooth recovery, minimizing downtime and preserving business continuity.

6. Find option:

- This allows the user (typically a manager or supervisor) to search and view sales details based on certain criteria such as the staff member who processed the sale.

Benefits of the System:

- **Increased Accuracy:** Automated calculations eliminate human errors in billing, ensuring that customers are billed accurately.
- **Faster Service:** The system significantly speeds up the transaction process, reducing waiting times for customers and increasing overall throughput.

SYSTEM REQUIREMENTS

SYSTEM CONFIGURATION

HARDWARE REQUIREMENTS

Model	: IBM System X3650M4[791514A]
Processor	: Intel Xeon ES-2620@2.00GHZ
Ram	: 2GB
Hard Disk	: 300GB
Monitor	: Samsung LCD 15.6 Inch
Keyboard	: IBM
Mouse	: IBM

SOFTWARE REQUIREMENTS

Operating System	: Windows 10 Basic Edition
Language used	: Python
Framework Used	: GUI
Version	: IDLE (Python 3.12)
Database	: MYSQL

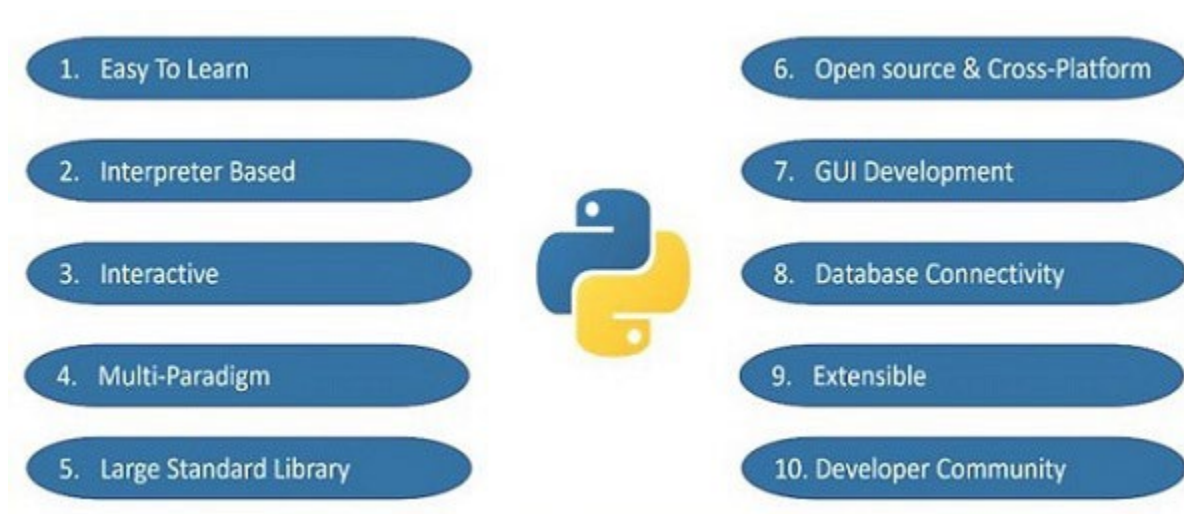
SOFTWARE DESCRIPTION

SOFTWARE DESCRIPTION

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). Python supports multiple programming paradigms, including Procedural, Object Oriented and Functional programming language. Python design philosophy emphasizes code readability with the use of significant indentation. This tutorial gives a complete understanding of Python programming language starting from basic concepts to advanced concepts.

PYTHON FEATURES

- Uses an elegant syntax, making the programs you write easier to read.
- Is an easy-to-use language that makes it simple to get your program working. This makes Python ideal for prototype development and other ad-hoc programming tasks, without compromising maintainability.
- Comes with a large standard library that supports many common programming tasks such as connecting to web servers, searching text with regular expressions, reading and modifying files.



- Python's interactive mode makes it easy to test short snippets of code. There's also a bundled development environment called IDLE.
- Is easily extended by adding new modules implemented in a compiled language such as C or C++.
- Can also be embedded into an application to provide a programmable interface.
- Runs anywhere, including Mac OS X, Windows, Linux, and UNIX, with unofficial builds also available for Android and iOS.

- Is free software in two senses? It doesn't cost anything to download or use Python, or to include it in your application. Python can also be freely modified and re-distributed because while the language is copyrighted it's available under an open-source license.

PYTHON CAREERS

- Game developer
- Web designer
- Python developer
- Full-stack developer
- Machine learning engineer
- Data scientist
- Data analyst
- Data engineer
- DevOps engineer
- Software engineer
- Many more other roles

CHARACTERISTICS OF PYTHON

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

PYTHON HISTORY

Guido Van Rossum, a Dutch programmer, created Python programming language. In the late 80's, he had been working on the development of ABC language in a computer science research institute named Centrum Wiskunde & Informatics (CWI) in the Netherlands. In 1991, Van Rossum conceived and published Python as a successor of ABC language. For many uninitiated people, the word Python is related to a species of snake. Rossum though attributes the choice of the name Python to a popular comedy series "Monty Python's Flying Circus" on BBC. Being the principal architect of Python, the developer community conferred upon him the title of "Benevolent Dictator for Life (BDFL)". However, in 2018, Rossum relinquished the title. Thereafter, the development and distribution of the reference implementation of Python is handled by a nonprofit organization Python Software Foundation. Important stages in the history of Python: Python 0.9.0 Python's first published version is 0.9. It was released in February 1991. It consisted of support for core object-oriented programming principles. Python 1.0 In January 1994, version 1.0 was released, armed with functional programming tools, features like support for complex numbers etc. Python 2.0 Next major version – Python 2.0 was launched in October 2000.

Many new features such as list comprehension, garbage collection and Unicode support were included with it. Python 3.0 Python 3.0, a completely revamped version of Python was released in December 2008. The primary objective of this revamp was to remove a lot of discrepancies that had crept in Python 2.x versions. Python 3 was backported to Python 2.6. It also included a utility named as python2to3 to facilitate automatic translation of Python 2 code to Python 3. EOL for Python 2.x even after the release of Python 3, Python Software Foundation continued to support the Python 2 branch with incremental micro versions till 2019. However, it decided to discontinue the support by the end of year 2020, at which time Python 2.

PYTHON FUNCTION

A function in Python is a reusable block of code that performs a specific task. It helps to make programs more organized, modular, and efficient.

Key Concepts of Python Functions

- Defining a Function
- Use the def keyword.
- Give the function a name.
- Add parameters (optional) inside parentheses.
- Write the function body with indentation.
- Use return (optional) to output a result.

PYTHON LAMBDA

Python, an anonymous function means that a function is without a name. As we already know that `def` keyword is used to define the normal functions and the `lambda` keyword is used to create anonymous functions.

Python lambda Syntax:

Lambda arguments: expression Python lambda properties:

- This function can have any number of arguments but only one expression, which is evaluated and returned.
- One is free to use lambda functions wherever function objects are required.
- You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.
- It has various uses in particular fields of programming, besides other types of expressions in functions.

Python input() Function

Python `input()` function is used to take user input. By default, it returns the user input in form of a string.

`input()` Function

Syntax:

`input(prompt)`

`prompt` [optional]: any string value to display as input message7.17 was the last version in the branch.

PYTHON MODULES

The concept of module in Python further enhances the modularity. You can define more than one related functions together and load required functions. A module is a file containing definition of functions, classes, variables, constants or any other Python object. Contents of this file can be made available to any other program. Python has the `import` keyword for this purpose.

Python GUI (Graphical User Interface) Concept

A GUI (Graphical User Interface) in Python allows users to interact with programs visually using buttons, menus, windows, and other graphical components instead of the command line. Python offers multiple GUI frameworks to build interactive applications.

1. Popular Python GUI Frameworks

Framework	Description
Tkinter	Built-in Python library for basic GUI development.
PyQt / PySide	Advanced GUI development using Qt framework. Kivy Used for multi-touch applications and mobile app development.
PyGTK	GUI framework based on GTK (used in Linux applications).
WxPython	Cross-platform GUI toolkit with a native look.

2. Basic Structure of a Python GUI Application (Using Tkinter)

- Steps to Create a Simple GUI
- Import the GUI library.
- Create the main application window.
- Add widgets (buttons, labels, text boxes, etc.).
- Define event handling (what happens when a user interacts).
- Run the application loop.

2. Key GUI Components (Widgets)

Widget	Description
Label	Displays text or images.
Button	Triggers an action when clicked.
Entry	Text input field.
Text	Multi-line text input.
Frame	Container to hold widgets.
Canvas	Draw shapes, images, or graphs.
Checkbutton	Checkbox for selection.
Radiobutton	Choose one option from multiple choices.
Listbox	Displays a list of selectable items.
Menu	Dropdown menus for navigation.

4. Advanced GUI Features

- Using Frames and Layouts (Grid, Pack, Place)
- Working with Images and Icons (PIL for image handling)
- Creating Dialog Boxes & Message Pop-ups (messagebox)
- Building Multi-Page Applications (ttk.Notebook)
- Styling with Themes (ttk module)

PYTHON DJANGO

Django, built with Python, is designed to help developers build secure, scalable, and feature-rich web applications quickly and efficiently. Whether you're a beginner looking to create your first dynamic website or an experienced developer aiming to enhance your skills, this tutorial will guide you through Django's core concepts.

This also tutorial provides you with Django projects to help you apply your knowledge and build some cool web applications. These projects not only provide you with experience in building with the Django framework but will also add value to your resume. This Python Django tutorial teaches basic to advanced Django concepts for backend development. Learn topics like forms, templates, views, ORM, etc.

DJANGO HISTORY

Django was created in the autumn of 2003, when the web programmers at the Lawrence Journal-World newspaper, Adrian Holovaty and Simon Willison, began using Python to build applications. Jacob Kaplan-Moss was hired early in Django's development shortly before Willison's internship ended. It was released publicly under a BSD license in July 2005. The framework was named after guitarist Django Reinhardt. Holovaty is a Romani jazz guitar player inspired in part by Reinhardt's music. In June 2008, it was announced that a newly formed Django Software Foundation (DSF) would maintain Django in the future.

Features of Django

- **Rapid Development:** Django's DRY principle accelerates development by reducing code repetition.
- **Admin Interface:** Comes with a ready-to-use, customizable admin panel for easy backend management.
- **Scalable:** Built to handle high traffic and complex applications, ideal for projects of any size.
- **Security:** Offers built-in protections against common security threats like XSS, CSRF, and SQL injection.
- **ORM:** Simplifies database interaction using Python, eliminating the need for raw SQL.
- **URL Routing:** Clean, readable URLs with easy mapping to views.
- **Template Engine:** Separates logic from presentation for dynamic, reusable web pages.
- **Extensive Documentation:** Well-organized resources for troubleshooting and learning.
- **Active Community:** Large community support with abundant third-party plugins and tools.

MySQL:

- **Introduction to MySQL:** MySQL is an open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) for managing and manipulating databases. It is one of the most widely used database systems, known for its reliability, performance, and flexibility. MySQL is a part of the **LAMP stack** (Linux, Apache, MySQL, and PHP/Perl/Python) and is often used in web development but can also be applied to other types of software applications, such as a **Fuel station management System**.
- In the context of a **Fuel station management System**, MySQL serves as the back-end database that stores and organizes essential data, such as fuel prices, transaction records, customer information, and inventory details. Using MySQL, the system can efficiently

manage large volumes of data, ensure data integrity, and allow easy retrieval and updating of information.

Key Features of MySQL:

- **Relational Database:** MySQL stores data in tables, which are related to each other through keys (e.g., primary and foreign keys).
- **SQL Language:** It uses SQL, a standard language for interacting with relational databases.
- **Open Source:** MySQL is free to use and has a large community of users and developers.
- **Cross-platform:** It works on different operating systems, including Windows, Linux, and macOS.
- **High Performance:** It is optimized for speed and can handle a large number of queries with low latency.
- **Scalability:** MySQL supports both small and large-scale applications, making it suitable for a wide range of projects.

Common Uses of MySQL:

1. **Web Development:** MySQL is often used in web applications, especially in combination with server-side scripting languages like PHP or frameworks like Laravel, Django, or Node.js. It stores data for dynamic websites (e.g., user data, product catalogs, etc.).
 - **Example:** WordPress, which powers many websites, uses MySQL as its database.
2. **E-commerce Platforms:** Online stores use MySQL to store product information, user accounts, orders, and transactions.
 - **Example:** Platforms like Magento or WooCommerce use MySQL to store e-commerce data.
3. **Data Warehousing:** MySQL can store and manage large datasets for analytics purposes. It's also used in smaller-scale data warehousing for aggregating and querying data.
4. **Content Management Systems (CMS):** MySQL is commonly used in CMS platforms like WordPress, Joomla, and Drupal, where it stores all content data, including blog posts, pages, and metadata.
5. **Customer Relationship Management (CRM):** MySQL can be used to store and manage customer data, sales, leads, and other CRM-related information.
6. **Mobile Applications:** Mobile apps with server-side databases can use MySQL to handle and store user data, which can be synchronized across devices or stored in a central location.

7. **Gaming:** Online games and social games often use MySQL to store player progress, scores, and game-related data.
8. **Enterprise Applications:** Large organizations use MySQL for managing business applications such as inventory management, HR systems, and financial records.

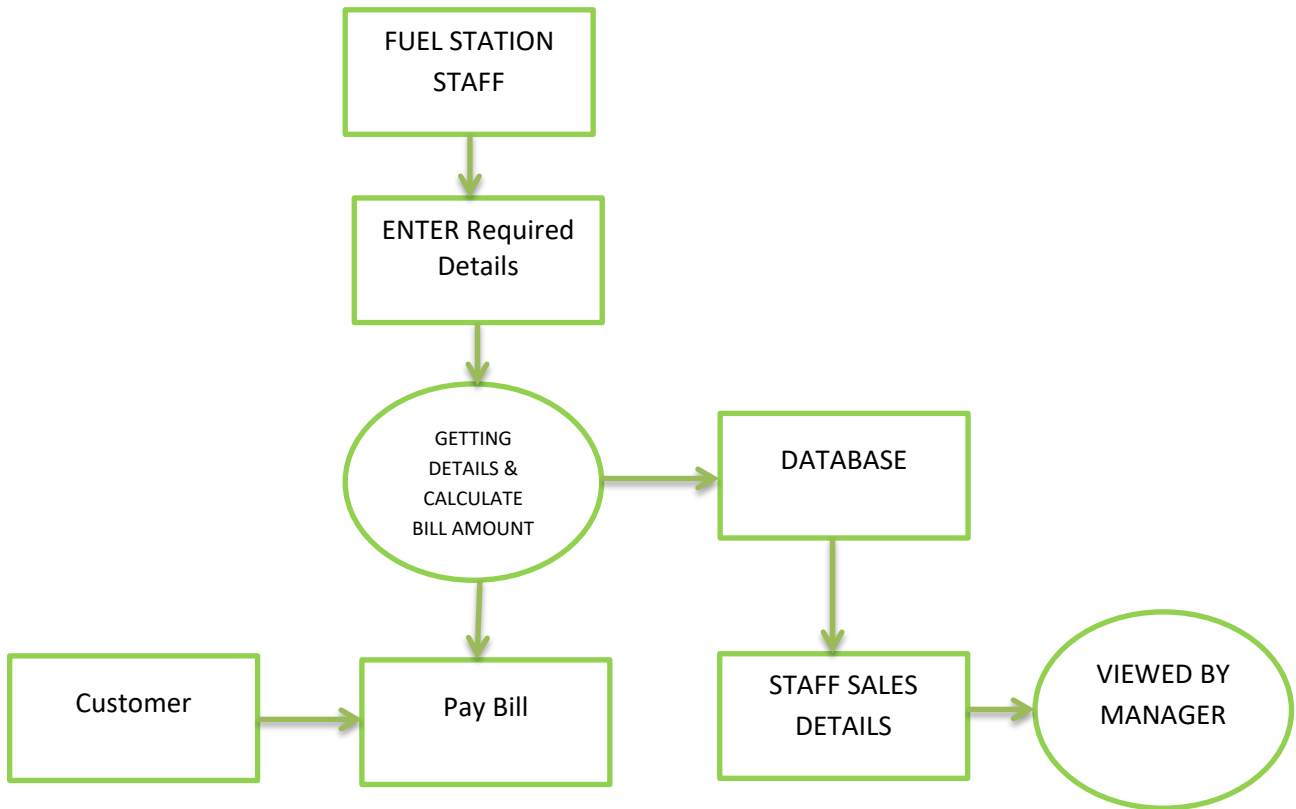
Advantages of MySQL:

- **Reliability:** It has built-in mechanisms to ensure data integrity and security.
- **Ease of Use:** MySQL is relatively easy to set up and use, especially for beginners.
- **Cost-effective:** Being open-source, MySQL can be used without licensing fees, making it a cost-effective option.
- **Security:** MySQL includes robust security features such as encryption, authentication, and access controls.
- **Backup and Recovery:** MySQL provides tools for backing up and recovering data, ensuring that information is safe in case of hardware failure.

SYSTEM DESIGN

SYSTEM DESIGN

DATA FLOW DIAGRAM



DATABASE SCHEMA DIAGRAM

submit		
Date		date
price		int(4)
empid		varchar(100)
vechicle_type		varchar(50)
litre		int(4)
total		int(4)

SOURCE CODE

```
import mysql.connector

from tkinter import*

from tkcalendar import DateEntry

from time import*

from tkinter import messagebox

from PIL import Image,ImageTk


db=mysql.connector.connect(host="localhost",user="root",password="",db="petrol")

cursor=db.cursor()

window=Tk()

window.title("fuel")

img=Image.open("p.png")

img1=ImageTk.PhotoImage(img)

lbl=Label(window,image=img1)

lbl.pack()


l=Label(window,text='FUEL STATION MANAGEMENT SYSTEM',font=("bauhaus
93",40,"bold"))

l.place(x=200,y=50)


time=Label(window,text='Date',font=("comic sans ms",18,"bold"))

time.place(x=950,y=150)

time1=Label(window,font=("comic sans ms",18))

time1.place(x=1050,y=150)

mod=strftime(" %d/%b/%Y")

time1.config(text=mod)

d=DateEntry(window)
```



```
l1=Label(window,text='Today price',font=("comic sans ms",18,"bold"))
```

```
l1.place(x=950,y=200)
```

```
pri=Entry(window,width=10,font=("comic sans ms",18,"bold"))
```

```
pri.place(x=1100,y=200)
```

```
l2=Label(window,text='Emp Id',font=("Arial",18,"bold"))
```

```
l2.place(x=200,y=180)
```

```
b=Entry(window,width=20,font=("Arial",18,"bold"))
```

```
b.place(x=550,y=180)
```

```
l4=Label(window,text='Vehicle type',font=("Arial",18,"bold"))
```

```
l4.place(x=200,y=250)
```

```
sel=StringVar()
```

```
b2=Radiobutton(window,text="2 wheeler",width=14,font=("lucida  
console",14),value="Two wheeler",variable=sel)
```

```
b2.place(x=550,y=250)
```

```
b2.deselect()
```

```
b21=Radiobutton(window,text="3 wheeler",width=14,font=("lucida  
console",14),value="Three wheeler",variable=sel)
```

```
b21.place(x=550,y=300)
```

```
b21.deselect()
```

```
b22=Radiobutton(window,text="4 wheeler",width=14,font=("lucida  
console",14),value="Four wheeler",variable=sel)
```

```
b22.place(x=550,y=350)
```

```
b22.deselect()
```

```
l5=Label(window,text='litre',font=("Arial",18,"bold"))
```

```
l5.place(x=200,y=400)

b3=Entry(window,width=20,font=("Arial",18,"bold"))

b3.place(x=550,y=400)


l6=Label(window,text='Bill Amount',font=("Arial",18,"bold"))

l6.place(x=200,y=450)

en=IntVar()

b4=Entry(window,width=20,textvariable=en,font=("Arial",18,"bold"))

b4.place(x=550,y=450)


def remove():

    b.delete(0,END)

    b3.delete(0,END)

    b4.delete(0,END)

    en.set(0)

def find():

    win=Tk()

    win.geometry("800x700")

    win.config(bg="#C1CDCD")

    inf=Label(win,text="Information",font=("lucida
calligraphy",40),bg="#C1CDCD",fg="#9C661F")

    inf.place(x=200,y=10)

    ve=Label(win,text='Emp_Id',font=("Arial",18,"bold"),bg="#C1CDCD")

    ve.place(x=100,y=150)

    Id=Entry(win,width=18,font=("Arial",18,"bold"),bg="#C1CDCD")

    Id.place(x=300,y=150)

    def result():
```

global c

if Id.get()=="":

c=Label(win,text="Please enter the
Emp_Id",font=("Arial",14),bg="#C1CDCD",fg="red")

c.place(x=300,y=200)

r=Label(win,text='
,font=("calibri",18),bg="#C1CDCD")

r.place(x=100,y=230)

r2=Label(win,text='
,font=("calibri",18),bg="#C1CDCD")

r2.place(x=100,y=280)

r4=Label(win,text='
,font=("calibri",18),bg="#C1CDCD")

r4.place(x=100,y=330)

r6=Label(win,text='
,font=("calibri",18),bg="#C1CDCD")

r6.place(x=100,y=380)

r8=Label(win,text='
,font=("calibri",18),bg="#C1CDCD")

r8.place(x=100,y=430)

else:

c=Label(win,text="Date :
,font=("Arial",14),bg="#C1CDCD",fg="red")

c.place(x=120,y=200)

r=Label(win,text='Date :',font=("calibri",18),bg="#C1CDCD")

r.place(x=100,y=230)

r1=Label(win,font=("Arial",18,"bold"),bg="#C1CDCD",fg="brown")

r1.place(x=400,y=230)

r2=Label(win,text='Today Petrol Price :',font=("calibri",18),bg="#C1CDCD")

```

r2.place(x=100,y=280)

r3=Label(win,font=("Arial",18,"bold"),bg="#C1CDCD")

r3.place(x=400,y=280)

r4=Label(win,text='Emp_Id :',font=("calibri",18),bg="#C1CDCD")

r4.place(x=100,y=330)

r5=Label(win,font=("Arial",18,"bold"),bg="#C1CDCD")

r5.place(x=400,y=330)

r6=Label(win,text=' Total Litre :',font=("calibri",18),bg="#C1CDCD")

r6.place(x=100,y=380)

r7=Label(win,font=("Arial",18,"bold"),bg="#C1CDCD")

r7.place(x=400,y=380)

r8=Label(win,text='Total Collection Amount:',font=("calibri",18),bg="#C1CDCD")

r8.place(x=100,y=430)

r9=Label(win,font=("Arial",18,"bold"),bg="#C1CDCD")

r9.place(x=400,y=430)


val=int(Id.get()),

sql=("select date_format(Date,'%d/%b/%Y'),price,empid,sum(litre),sum(total) from
submit where empid=%s")

cursor.execute(sql,val)

record=cursor.fetchall()

for i in record:

    r1.config(text=i[0])

    r3.config(text=i[1])

    r5.config(text=i[2])

    r7.config(text=i[3])

    r9.config(text=i[4])

```

```
btn2=Button(win,text="Find",width=5,command=result,font=("times new roman",14,"bold"))
```

```
btn2.place(x=600,y=150)
```

```
win.mainloop()
```

```
def calculate():
```

```
    if b.get()==" " or b3.get()==" " or sel.get()==" " or pri.get()==" ":
```

```
        messagebox.askquestion("error","please enter data")
```

```
    else:
```

```
        lit= int(b3.get())
```

```
        pay=lit*float(pri.get())
```

```
        en.set(pay)
```

```
def submit():
```

```
    if b.get()==" " or b3.get()==" " or sel.get()==" " or pri.get()==" ":
```

```
        messagebox.askquestion("error","please enter data")
```

```
    else:
```

```
        sql=("INSERT INTO
```

```
submit(Date,price,empid,vechicle_type,litre,total)values(%s,%s,%s,%s,%s,%s)")
```

```
        val=(d.get_date(),pri.get(),b.get(),sel.get(),b3.get(),b4.get())
```

```
        cursor.execute(sql,val)
```

```
        db.commit()
```

```
        messagebox.askquestion("successfull","Data was Stored")
```

```
bt=Button(window,text="Calculate",width=10,command=calculate,font=("times new roman",18,"bold"))
```

```
bt.place(x=150,y=600)
```

```
bt1=Button(window,text="Submit",width=10,command=submit,font=("times new  
roman",18,"bold"))
```

```
bt1.place(x=350,y=600)
```

```
bt2=Button(window,text="Remove",width=10,command=remove,font=("times new  
roman",18,"bold"))
```

```
bt2.place(x=550,y=600)
```

```
bt3=Button(window,text="Find",width=10,command=find,font=("times new  
roman",18,"bold"))
```

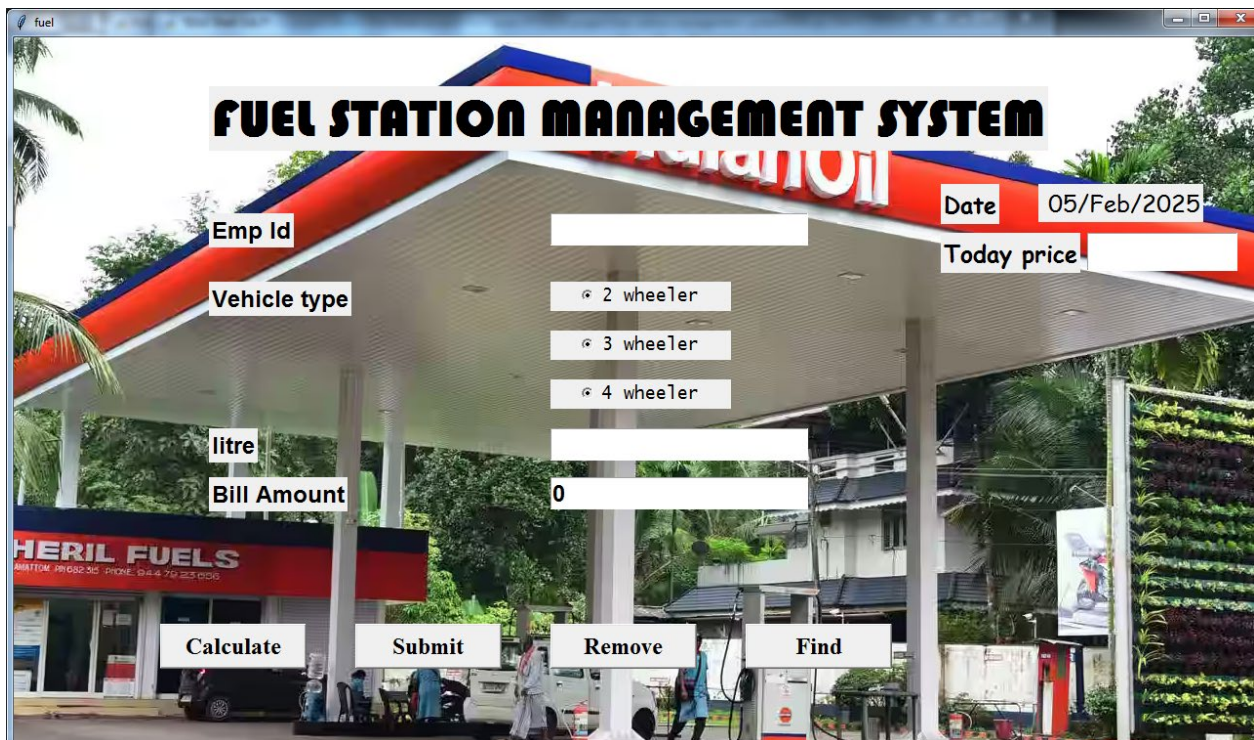
```
bt3.place(x=750,y=600)
```

```
window.mainloop()
```

SCREEN LAYOUT

FUEL STATION MANAGEMENT SYSTEM

Step 1:



The screenshot shows a web application interface for a fuel station management system. The background is a photograph of a gas station with a red and blue canopy. The application is titled "FUEL STATION MANAGEMENT SYSTEM" in large, bold, black letters at the top. Below the title, there are several input fields and buttons. On the left, there are labels for "Emp Id", "Vehicle type", "litre", and "Bill Amount". To the right of these labels are corresponding input fields. The "Date" field is pre-filled with "05/Feb/2025". The "Today price" field is empty. The "Vehicle type" field has three radio button options: "2 wheeler", "3 wheeler", and "4 wheeler". The "litre" field is empty. The "Bill Amount" field is pre-filled with "0". At the bottom of the form, there are four buttons: "Calculate", "Submit", "Remove", and "Find". The browser's address bar shows "fuel".

FUEL STATION MANAGEMENT SYSTEM

Emp Id

Date 05/Feb/2025

Today price

Vehicle type

☐ 2 wheeler

☐ 3 wheeler

☐ 4 wheeler

litre

Bill Amount 0

Step 2:

Fill the details and click on “CALCULATE” button

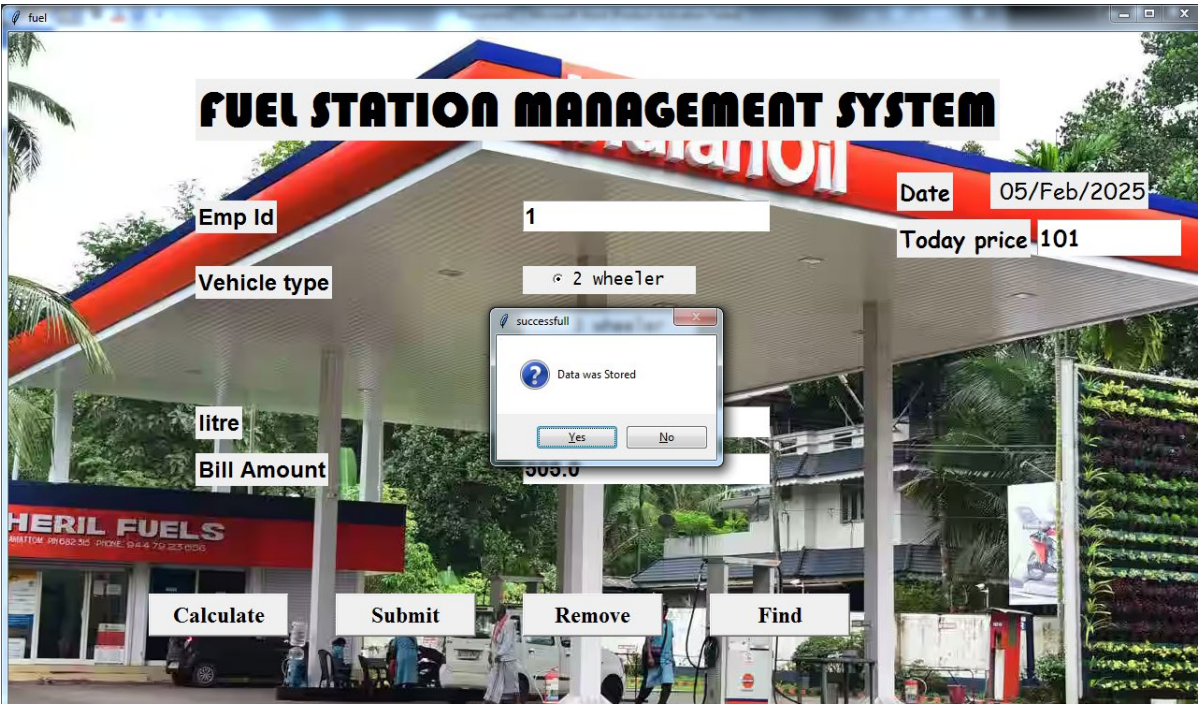
The image shows a web application titled "FUEL STATION MANAGEMENT SYSTEM" overlaid on a background image of a Shell gas station. The application has a title bar at the top with a "fuel" icon and standard window controls. The main content area includes:

- A title "FUEL STATION MANAGEMENT SYSTEM" in a large, bold, black font.
- A "Date" input field containing "05/Feb/2025".
- An "Emp Id" input field containing "1".
- A "Vehicle type" dropdown menu with three options: "2 wheeler", "3 wheeler", and "4 wheeler".
- A "litre" input field containing "5".
- A "Bill Amount" input field containing "505.0".
- A "Today price" input field containing "101".
- Four buttons at the bottom: "Calculate", "Submit", "Remove", and "Find".

The background image shows a Shell gas station with a red and blue canopy, a Shell logo, and a sign that reads "SHELL FUELS". There are also some people and cars visible in the background.

Step 3:

Click on “SUBMIT” button



Step 4:

Click on “REMOVE” button

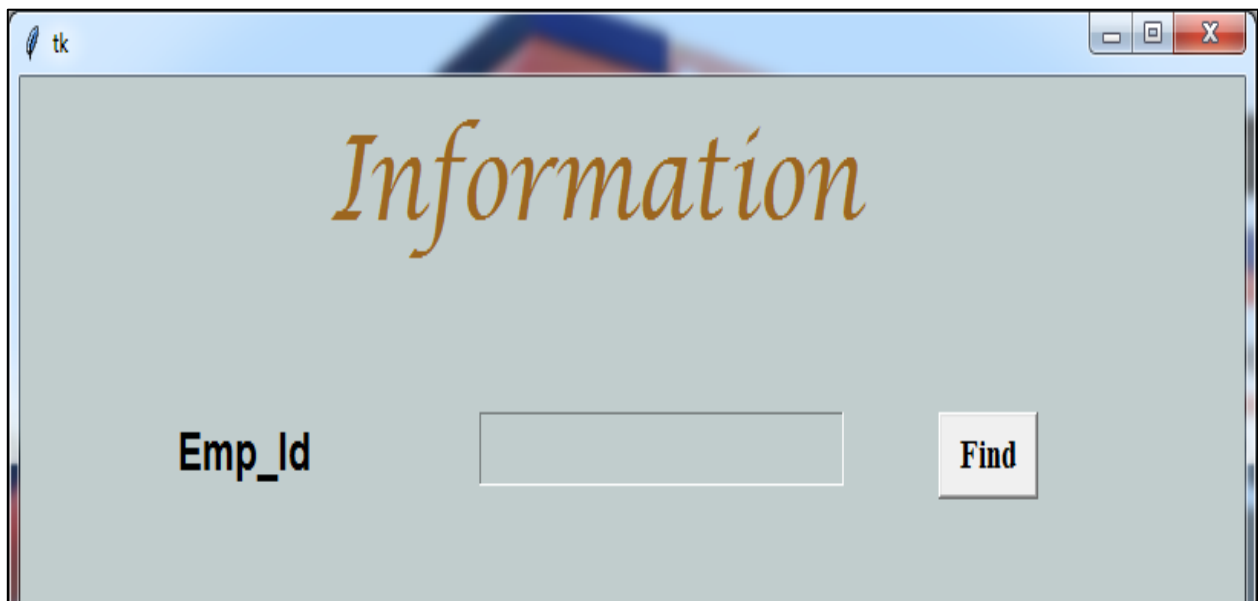
The image shows a web application interface for a 'FUEL STATION MANAGEMENT SYSTEM'. The interface is overlaid on a background image of a gas station with a red and blue canopy. The application features several input fields and buttons:

- Title:** A text input field containing 'Fuel Station Management System'.
- Date:** A text input field containing '05/Feb/2025'.
- Emp Id:** A text input field.
- Vehicle type:** A dropdown menu with three options: '2 wheeler', '3 wheeler', and '4 wheeler'.
- litre:** A text input field.
- Bill Amount:** A text input field containing '0'.
- Buttons:** Four buttons labeled 'Calculate', 'Submit', 'Remove', and 'Find' are positioned at the bottom of the form.

The background image shows a gas station with a red and blue canopy. A sign on the left reads 'HERIL FUELS' and 'MAYATEM 99 652 512 | PHONE: 94 47 72 25 006'. A person in a blue uniform is visible near a fuel pump.

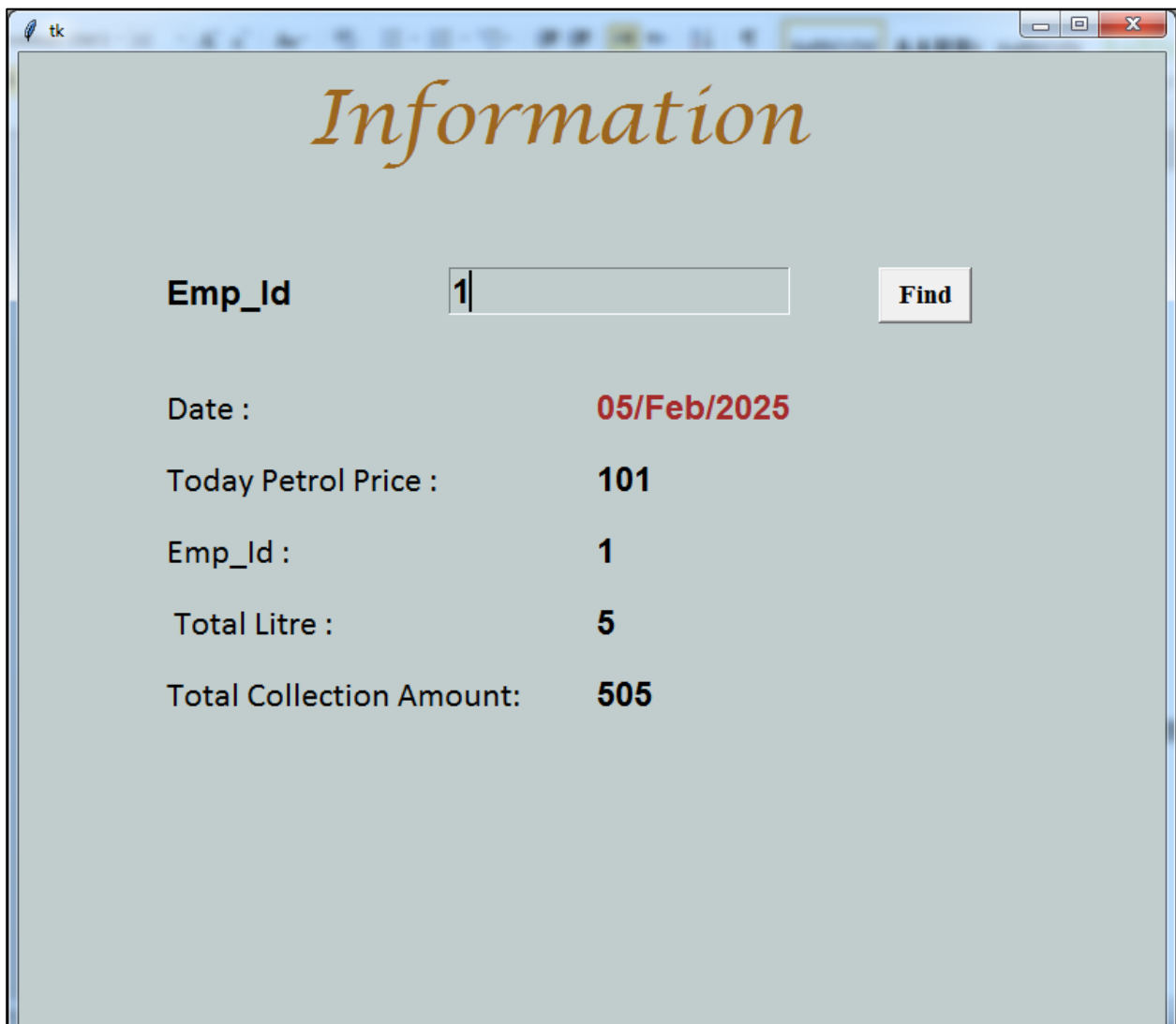
Step 5:

Click on “FIND” button



Step 6:

Fill the details and click “FIND” button in the above window



The screenshot shows a Tkinter window titled "Information" with a light blue background. At the top, the word "Information" is written in a large, brown, cursive font. Below this, there is a form with the following elements:

- A label "Emp_Id" followed by a text input field containing the number "1".
- A "Find" button to the right of the input field.
- A list of results displayed below the input field:
 - Date : 05/Feb/2025
 - Today Petrol Price : 101
 - Emp_Id : 1
 - Total Litre : 5
 - Total Collection Amount: 505

SYSTEM TESTING

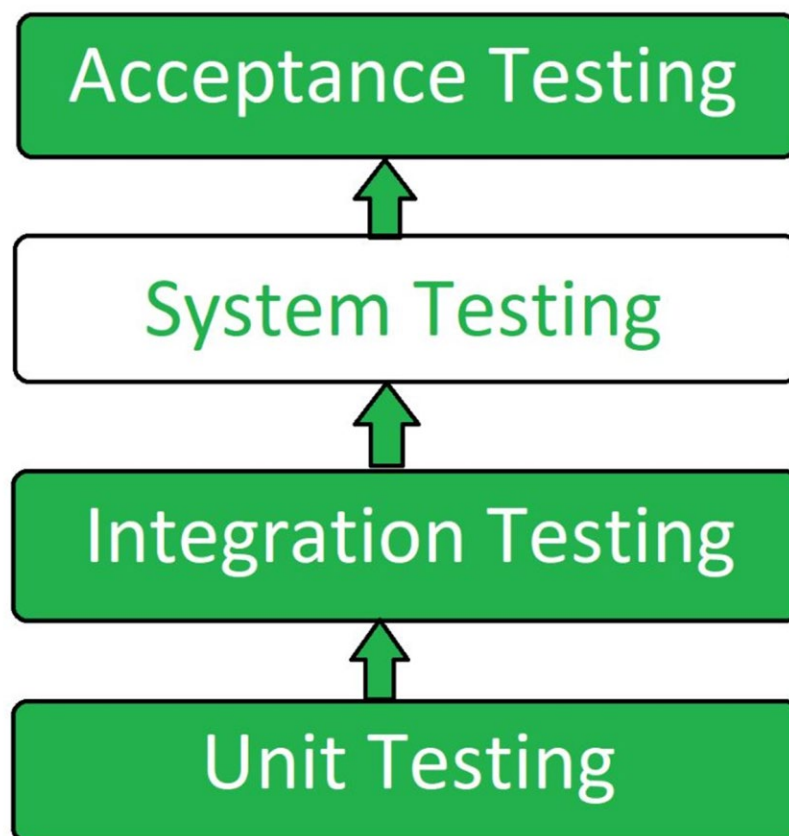
SYSTEM TESTING

System testing is a comprehensive testing process that evaluates the complete and integrated software system to verify that it meets specified requirements. It involves testing the software's functionality, reliability, and performance as a whole, rather than individual components, to ensure overall system quality and effectiveness.

It is indeed typically accomplished by software testing engineers. Its performance occurs in a context comparable to the one used in production, permitting the developers and other relevant parties to analyse user responses.

There are four levels of software testing: unit testing, integration testing, system testing and acceptance testing, all are used for the testing purpose. Unit Testing used to test a single software; Integration Testing used to test a group of units of software, System Testing used to test a whole system and Acceptance Testing used to test the acceptability of business requirements. Here we are discussing system testing which is the third level of testing levels.

The below flowchart shows where the System testing happens in the software development life-cycle.



It is important to follow a specific order when testing software. Following is a chronologically organized list of software testing categories.

1. Unit testing:

Unit testing is a software testing method that involves testing individual units of source code, typically a single function, method, or module, in isolation to ensure they behave as expected. This approach helps to verify the correctness of each unit, detect bugs or defects early in development, and improve overall code quality. By testing small, specific pieces of code, developers can identify and fix errors quickly, reducing the likelihood of downstream problems and making it easier to maintain and update the codebase over time.

2. Integration testing:

Integration testing is a software testing method that involves testing multiple individual units of code, such as functions, modules, or services, together as a combined system to ensure they interact and function correctly as a whole. This approach helps to verify that the interactions between different components or systems are working as expected, and that the integrated system meets the required functional and performance specifications. By testing the interactions and interfaces between components, integration testing helps to identify a fix issues related to data exchange, API calls, and system workflows, ultimately ensuring that the integrated system is stable, reliable, and works as intended.

3. System testing:

System testing is a software testing method that involves testing a complete, integrated software system to ensure it meets the specified requirements, works as expected, and is free from defects. This approach simulates real-world scenarios and usage conditions to evaluate the system's functionality, performance, security, and usability. System testing verifies that the software system operates correctly, from the user interface to the backend systems, and that it integrates seamlessly with external systems, hardware, and third-party components. By testing the entire system, end-to-end, and system testing helps to identify and fix defects, ensuring that the software system is reliable, stable, and meets the needs of its users.

4. Acceptance testing:

Acceptance testing, also known as user acceptance testing (UAT), is a software testing method that involves verifying that a software system meets the requirements and expectations of its end-users, stakeholders, or customers. This type of testing is typically performed by the users or stakeholders themselves, rather than by the development team, to ensure that the software system is functional, usable, and meets their acceptance criteria. Acceptance testing validates that the software system works as intended, is free from critical defects, and provides the required business value, ultimately giving stakeholder's confidence that the system is ready for deployment and use.

There are mainly two widely used methods for software testing, one is White box testing which uses internal coding to design test cases and another is black box testing which uses GUI or user perspective to develop test cases.

- **White box testing**
- **Black box testing**

System testing falls under Black box testing as it includes testing of the external working of the software. Testing follows user's perspective to identify minor defects.

System Testing includes the following steps:

- Verification of input functions of the application to test whether it is producing the expected output or not.
- Testing of integrated software by including external peripherals to check the interaction of various components with each other.
- Testing of the whole system for End to End testing.
- Behaviour testing of the application via a user's experience

SYSTEM TESTING IMPORTANT

- System Testing gives hundred percent assurance of system performance as it covers end to end function of the system.
- It includes testing of System software architecture and business requirements.
- It helps in mitigating live issues and bugs even after production.
- System testing uses both existing system and a new system to feed same data in both and then compare the differences in functionalities of added and existing functions so, the user can understand benefits of new added functions of the system.

CONCLUSION

CONCLUSION

The **Fuel station Management System** is a comprehensive solution that simplifies the billing process, enhances operational efficiency, and improves the customer experience at fuel stations. The combination of **Python**, **Tkinter**, and **MySQL** ensures that the system is both powerful and user-friendly, capable of handling everyday operations while allowing for future improvements. With its automated features, real-time data management, and ease of use, this system is well-suited for modern fuel stations looking to upgrade their billing and management processes.

In conclusion, the Fuel station management System is an essential tool that not only streamlines daily operations but also contributes to the overall growth and success of the Fuel station business. The system's flexible and robust design ensures that it can adapt to future needs, making it an investment worth considering for any petrol station.

BIBLIOGRAPHY

BIBLIOGRAPHY

BOOKS REFERED

1. "Python and Tkinter Programming" by John E. Grayson
2. "Python GUI Development with Tkinter" by David E. Wheeler

WEBSITES REFERED

1. <https://docs.python.org/3/library/tkinter.html>
2. <https://www.mysql.com/>