**AUTOMOBILE MANAGEMENT SYSTEM**

**Abstract**

The automobile management system is a powerful tool that streamlines the operations of automobile service centers. It helps to automate various tasks such as adding service men, assigning work, adding products, viewing payments, stocks, bills, and car work status. This system improves the efficiency of the service center by reducing manual labor and errors. It also provides real-time information and reports that enable the management to make informed decisions. With the automobile management system, the service center can provide better service to its customers, improve its reputation, and increase its revenue. Overall, the automobile management system is an indispensable tool that enhances the productivity and profitability of an automobile service center. In addition to automating various tasks, the automobile management system also helps to improve communication among the different users of the system. The system allows service men to update the status of the service, add bills, and view service details. This enables the management to have real-time updates on the status of the service and make necessary adjustments. The system also allows customers to apply for services, make payments, and view the status of their service. This provides transparency and improves customer satisfaction.Furthermore, the automobile management system is designed to be user-friendly, with an intuitive interface that makes it easy for users to navigate and perform various tasks.

**CHAPTER 1**

**INTRODUCTION TO THE PROJECT**

The automobile industry has evolved significantly over the years, and so have the automobile service centers. As the number of vehicles on the roads increases, the demand for efficient and effective automobile service centers also grows. To meet this demand, automobile service centers require advanced tools and systems to streamline their operations, improve their efficiency, and provide better service to their customers. One such tool is the Automobile Management System, which is a comprehensive platform that automates various tasks such as adding service men, assigning work, adding products, viewing payments, stocks, bills, and car work status. The system has different user types, each with specific functionalities, and it is designed to be user-friendly, secure, and provide real-time information and reports. In this project, we will develop an Automobile Management System that will help automobile service centers to manage their operations efficiently, improve their service quality, and gain a competitive edge in the market. The Automobile Management System is an advanced technology that simplifies the management of automobile service centers. It enables the service centers to track their inventory, schedule services, manage payments, and maintain customer records in one place. The system is designed to enhance communication among the different users of the system, including service men, admin, and customers. This way, the management can have a clear overview of the service center's operations and make informed decisions. Moreover, the Automobile Management System is a cost-effective solution that eliminates the need for manual labor, thereby reducing operational costs and increasing revenue. Overall, the system is a must-have for any automobile service center that wants to streamline its operations, improve its customer service, and stay ahead of the competition.

**CHAPTER 2**

**REQUIREMENT SPECIFICATION**

**2.1 EXISTING SYSTEM**

The existing system for automobile service centers usually involves a manual approach to managing the center's operations. This manual approach often results in inefficiencies, inaccuracies, and delays. The manual approach involves using physical books or registers to record customer information, services, and payment details. The service center staff also needs to manually update the status of each service and bill the customer accordingly. This manual process can be time-consuming and prone to errors, leading to delays in service delivery and poor customer experience.Moreover, the manual approach makes it difficult for service centers to keep track of their inventory, monitor stock levels, and place orders on time. This can lead to a shortage of essential spare parts, further delaying the service delivery time. The manual approach also makes it difficult to generate reports on the service center's operations, limiting the management's ability to make informed decisions.In conclusion, the existing system for automobile service centers is inefficient, prone to errors, and limits the service center's ability to provide quality service to its customers. Therefore, there is a need for an automated system that can streamline the service center's operations, improve its efficiency, and enhance customer satisfaction.

**DISADVANTAGE:**

* Inefficiency: The manual process of managing a service center is inefficient and time-consuming, which affects the service center's ability to provide quality service to its customers.
* Human error: The manual process is prone to human error, leading to mistakes in billing, inventory management, and service delivery, which can negatively impact customer satisfaction.
* Limited communication: The existing system has limited communication channels, which can lead to delays in service delivery and misunderstandings among different stakeholders.
* Poor inventory management: The manual process of managing inventory can lead to stock-outs, overstocking, and delayed service delivery, which negatively impacts customer satisfaction.
* Lack of real-time reporting: The existing system lacks real-time reporting, which limits the management's ability to make informed decisions and identify areas of improvement.
* Security issues: The manual process is prone to security issues such as unauthorized access to data, loss of data, and data breaches.

**2.2 PROPOSED SYSTEM**

The proposed system is an Automobile Management System that aims to automate and streamline the operations of automobile service centers. The system will have different user types, including service men, admin, and customers, each with specific functionalities. The proposed system will provide an easy-to-use interface that is designed to enhance user experience and enable users to perform their tasks efficiently.The Automobile Management System will automate various tasks such as adding service men, assigning work, adding products, viewing payments, stocks, bills, and car work status. The system will also allow service men to update the status of the service, add bills, and view service details. Customers can apply for services, make payments, and view the status of their service. The system will also allow the management to generate real-time reports on the service center's operations, enabling them to make informed decisions.Furthermore, the proposed system will allow service centers to track their inventory, monitor stock levels, and place orders on time. The system will have an automated inventory management module that will keep track of the inventory levels and alert the management when stock levels are low. This way, the service center can ensure that essential spare parts are always available, reducing the service delivery time and improving customer satisfaction.In conclusion, the proposed Automobile Management System is an advanced technology that will streamline the operations of automobile service centers, improve their efficiency, and enhance customer satisfaction. By using the proposed system, service centers can automate tasks, improve communication, manage inventory, and generate real-time reports on their operations, thereby providing better service to their customers and staying ahead of the competition.

Top of Form

**ADVANTAGES:**

* Improved efficiency: The system automates various tasks, reducing the need for manual labor, and improving the service center's efficiency.
* Enhanced customer service: The system allows customers to apply for services, make payments, and track their service status, enhancing customer satisfaction.
* Real-time reporting: The system provides real-time reporting on the service center's operations, enabling the management to make informed decisions.
* Inventory management: The system automates inventory management, enabling the service center to keep track of stock levels and place orders on time, reducing service delivery time and improving customer satisfaction.
* Cost-effective: The system eliminates the need for manual labor, reducing operational costs, and increasing revenue.
* Secure: The system provides security features such as user authentication, data encryption, and access control, protecting the system from unauthorized access and data breaches.
* Streamlined communication: The system enables streamlined communication among the different users of the system, including service men, admin, and customers, improving the service center's overall communication.

**2.3 HARDWARE REQUIREMENTS**

* Processors: Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM
* Disk space: 320 GB
* Operating systems: Windows® 10, macOS\*, and Linux\*

**2.4 SOFTWARE REQUIREMENTS**

* Server Side : Python 3.7.4(64-bit) or (32-bit)
* Client Side : HTML, CSS, Bootstrap
* IDE : Flask 1.1.1
* Back end : MySQL 5.
* Server : Wampserver 2i
* OS : Windows 10 64 –bit or Ubuntu 18.04 LTS “Bionic Beaver”

**2.5 SOFTWARE DESCRIPTION**

**Python 3.7.4**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level 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). This tutorial gives enough understanding on Python programming language.

****

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages. Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain.

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc. The biggest strength of Python is huge collection of standard library which can be used for the following:

* Machine Learning
* GUI Applications (like Kivy, Tkinter, PyQt etc. )
* Web frameworks like Django (used by YouTube, Instagram, Dropbox)
* Image processing (like OpenCV, Pillow)
* Web scraping (like Scrapy, BeautifulSoup, Selenium)
* Test frameworks
* Multimedia
* Scientific computing
* Text processing and many more..

**Data & Database**

Suppose a company needs to store the names of hundreds of employees working in the company in such a way that all the employees can be individually identified. Then, the company collects the **data** of all those employees. Now, when I say data, I mean that the company collects distinct pieces of information about an object. So, that object could be a real-world entity such as people, or any object such as a mouse, laptop etc.

**Database Management System & Types of DBMS**

A **Database Management System** (**DBMS**) is a software application that interacts with the user, applications and the database itself to capture and analyze data. The data stored in the database can be modified, retrieved and deleted, and can be of any type like strings, numbers, images etc.

**Types of DBMS**

There are mainly 4 types of DBMS, which are Hierarchical, Relational, Network, and Object-Oriented DBMS.

* **Hierarchical DBMS:**As the name suggests, this type of DBMS has a style of predecessor-successor type of relationship. So, it has a structure similar to that of a tree, wherein the nodes represent records and the branches of the tree represent fields.
* **Relational DBMS (RDBMS):** This type of DBMS, uses a structure that allows the users to identify and access data *in relation* to another piece of data in the database.
* **Network DBMS:**This type of DBMS supports many to many relations wherein multiple member records can be linked.
* **Object-oriented DBMS:**This type of DBMS uses small individual software called objects. Each object contains a piece of data, and the instructions for the actions to be done with the data.

**Structured Query Language (SQL)**

SQL is the core of a relational database which is used for accessing and managing the database. By using SQL, you can add, update or delete rows of data, retrieve subsets of information, modify databases and perform many actions. The different subsets of SQL are as follows:

* **DDL** (Data Definition Language) – It allows you to perform various operations on the database such as CREATE, ALTER and DELETE objects.
* **DML** (Data Manipulation Language) – It allows you to access and manipulate data. It helps you to insert, update, delete and retrieve data from the database.
* **DCL** (Data Control Language) – It allows you to control access to the database. Example – Grant or Revoke access permissions.
* **TCL**(Transaction Control Language) – It allows you to deal with the transaction of the database. Example – Commit, Rollback, Savepoint, Set Transaction.

**BACK END**

**MySQL**

MySQL tutorial provides basic and advanced concepts of MySQL. Our MySQL tutorial is designed for beginners and professionals. MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company. MySQL database that provides for how to manage database and to manipulate data with the help of various SQL queries. These queries are: insert records, update records, delete records, select records, create tables, drop tables, etc. There are also given MySQL interview questions to help you better understand the MySQL database.



MySQL is currently the most popular database management system software used for managing the relational database. It is open-source database software, which is supported by Oracle Company. It is fast, scalable, and easy to use database management system in comparison with Microsoft SQL Server and Oracle Database. It is commonly used in conjunction with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications. It is developed, marketed, and supported by MySQL AB, a Swedish company, and written in C programming language and C++ programming language. The official pronunciation of MySQL is not the My Sequel; it is My Ess Que Ell. However, you can pronounce it in your way. Many small and big companies use MySQL. MySQL supports many Operating Systems like Windows, Linux, MacOS, etc. with C, C++, and Java languages.

**The Apache Web Server**

In addition to PHP, MySQL, JavaScript, and CSS, there’s actually a fifth hero in the dynamic Web: the web server. In the case of this book, that means the Apache web server. We’ve discussed a little of what a web server does during the HTTP server/client exchange, but it actually does much more behind the scenes. For example, Apache doesn’t serve up just HTML files—it handles a wide range of files, from images and Flash files to MP3 audio files, RSS (Really Simple Syndication) feeds, and more. Each element a web client encounters in an HTML page is also requested from the server, which then serves it up. But these objects don’t have to be static files, such as GIF images. They can all be generated by programs such as PHP scripts. That’s right: PHP can even create images and other files for you, either on the fly or in advance to serve up later. To do this, you normally have modules either precompiled into Apache or PHP or called up at runtime. One such module is the GD library (short for Graphics Draw), which PHP uses to create and handle graphics.

Apache also supports a huge range of modules of its own. In addition to the PHP module, the most important for your purposes as a web programmer are the modules that handle security. Other examples are the Rewrite module, which enables the web server to handle a varying range of URL types and rewrite them to its own internal requirements, and the Proxy module, which you can use to serve up often-requested pages from a cache to ease the load on the server. Later in the book, you’ll see how to actually use some of these modules to enhance the features provided by the core technologies we cover. About Open Source Whether or not being open source is the reason these technologies are so popular has often been debated, but PHP, MySQL, and Apache are the three most commonly used tools in their categories. What can be said, though, is that being open-source means that they have been developed in the community by teams of programmers writing the features they themselves want and need, with the original code available for all to see and change. Bugs can be found and security breaches can be prevented before they happen. There’s another benefit: all these programs are free to use. There’s no worrying about having to purchase additional licenses if you have to scale up your website and add more servers. And you don’t need to check the budget before deciding whether to upgrade to the latest versions of these products.

**WampServer**

WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database. Alongside, PhpMyAdmin allows you to manage easily your database.



WAMPServer is a reliable web development software program that lets you create web apps with MYSQL database and PHP Apache2. With an intuitive interface, the application features numerous functionalities and makes it the preferred choice of developers from around the world. The software is free to use and doesn’t require a payment or subscription.

**Bootstrap 4**

Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.



It solves many problems which we had once, one of which is the cross-browser compatibility issue. Nowadays, the websites are perfect for all the browsers (IE, Firefox, and Chrome) and for all sizes of screens (Desktop, Tablets, Phablets, and Phones). All thanks to Bootstrap developers -Mark Otto and Jacob Thornton of Twitter, though it was later declared to be an open-source project.

**Easy to use**: Anybody with just basic knowledge of HTML and CSS can start using Bootstrap

**Responsive features**: Bootstrap's responsive CSS adjusts to phones, tablets, and desktops

**Mobile-first approach**: In Bootstrap, mobile-first styles are part of the core framework

**Browser compatibility**: Bootstrap 4 is compatible with all modern browsers (Chrome, Firefox, Internet Explorer 10+, Edge, Safari, and Opera)

**Using an IDE**

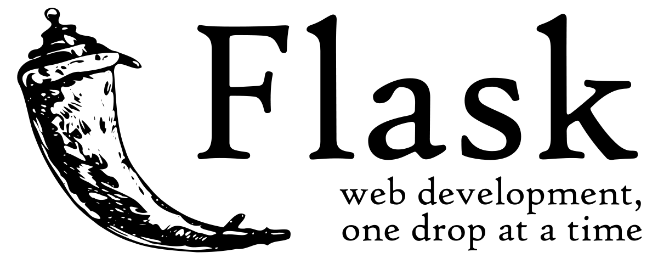
As good as dedicated program editors can be for your programming productivity, their utility pales into insignificance when compared to Integrated Developing Environments (IDEs), which offer many additional features such as in-editor debugging and program testing, as well as function descriptions and much more.

**Web Framework**

Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

**Flask**

[Flask](http://flask.pocoo.org/) is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.



Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have formed a validation support. Instead, Flask supports the extensions to add such functionality to the application.  Although Flask is rather young compared to most [Python](https://quintagroup.com/services/python) frameworks, it holds a great promise and has already gained popularity among Python web developers. Let’s take a closer look into Flask, so-called “micro” framework for Python.

Flask was designed to be easy to use and extend. The idea behind Flask is to build a solid foundation for web applications of different complexity. From then on you are free to plug in any extensions you think you need. Also you are free to build your own modules. Flask is great for all kinds of projects. It's especially good for prototyping.

Flask is part of the categories of the micro-framework. Micro-framework are normally framework with little to no dependencies to external libraries. This has pros and cons. Pros would be that the framework is light, there are little dependency to update and watch for security bugs, cons is that some time you will have to do more work by yourself or increase yourself the list of dependencies by adding plugins. In the case of Flask, its dependencies are:

**WSGI-**Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

**Werkzeug-**It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

**JSON**

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.



JSON consists of "name: object" pairs and punctuation in the form of brackets, parentheses, semi-colons and colons. Each object is defined with an operator like "text:" or "image:" and then grouped with a value for that operator. The simple structure and absence of mathematical notation or algorithms, JSON is easy to understand and quickly mastered, even by users with limited formal programming experience, which has spurred adoption of the format as a quick, approachable way to create interactive pages.

* 1. **MODULE DESCRIPTION**

User Management Module: This module allows administrators to manage user accounts, including creating, editing, and deleting user profiles.

Service Management Module: This module allows service managers to manage service requests, assign service technicians, and track service history.

Inventory Management Module: This module allows inventory managers to manage stock levels, track inventory movements, and generate inventory reports.

Billing Management Module: This module allows administrators to manage customer billing, generate invoices, and track payment status.

Product Management Module: This module allows administrators to manage product catalogs, add or remove products, and set product pricing.

Vehicle Management Module: This module allows service managers to manage vehicle records, including service history, maintenance records, and vehicle status.

Reporting Module: This module allows administrators to generate various reports related to service, inventory, and billing data.

Payment Gateway Module: This module integrates with payment gateways to enable online payments for service requests and product purchases.

Email Notification Module: This module sends automated email notifications to `customers and service managers regarding service requests, billing status, and product orders.

Security Management Module: This module allows administrators to manage user access levels, configure system security settings, and track system activity logs.

**CHAPTER 3**

**DESIGN**

**3.1 DATA FLOW DIAGRAM**

A two-dimensional diagram that explains how data is processed and transferred in a system. The graphical depiction identifies each source of data and how it interacts with other data sources to reach a common output. Individuals seeking to draft a data flow diagram must (1) identify external inputs and outputs, (2) determine how the inputs and outputs relate to each other, and (3) explain with graphics how these connections relate and what they result in. This type of diagram helps business development and design teams visualize how data is processed and identify or improve certain aspects.

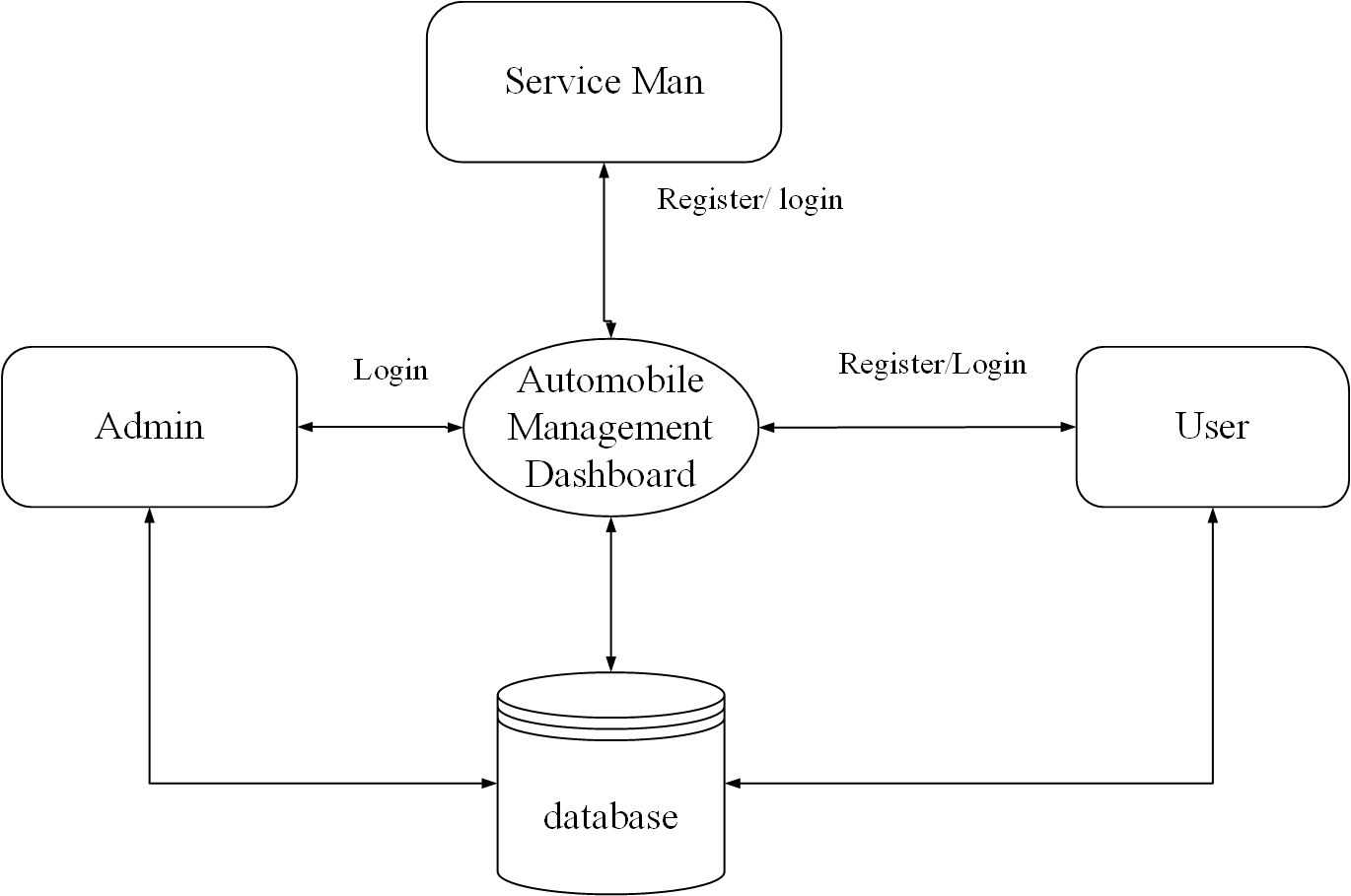
**Data flow Symbols:**

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| http://cpanel.stpaulsscience.org/gceict/specifications/ocr/unit3/sdlc/dfd/entity.jpg | An **entity**. A source of data or a destination for data. |
|  | A **process** or task that is performed by the system. |
| http://cpanel.stpaulsscience.org/gceict/specifications/ocr/unit3/sdlc/dfd/store.jpg | A **data store**, a place where data is held between processes. |
| http://cpanel.stpaulsscience.org/gceict/specifications/ocr/unit3/sdlc/dfd/flow.jpg | A **data flow**. |

**LEVEL 0**

The Level 0 DFD shows how the system is divided into 'sub-systems' (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

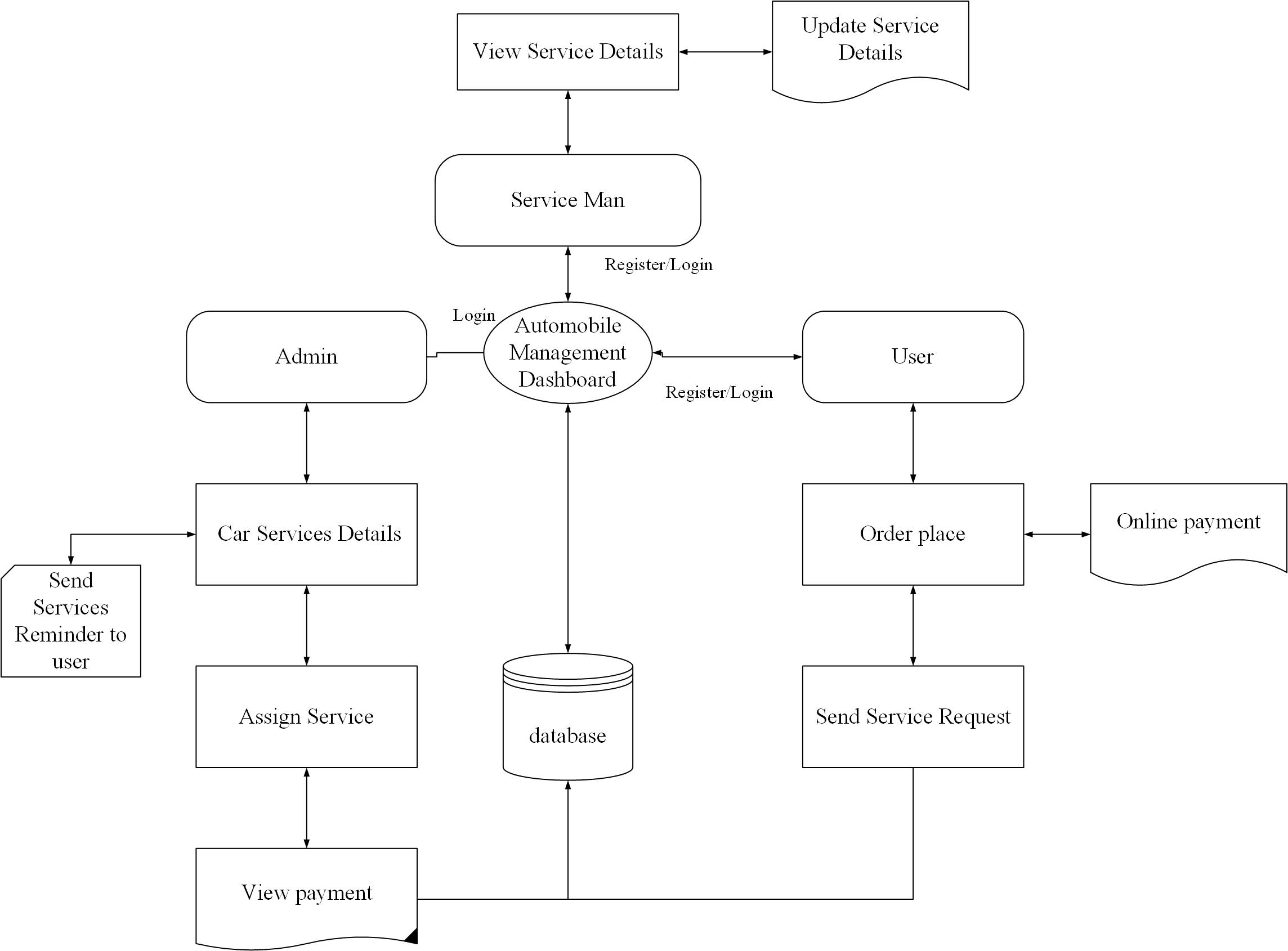
**DFD LEVEL-0**

****

**Figure 1.1 Data Flow Diagram Level 0**

**DFD LEVEL-1**

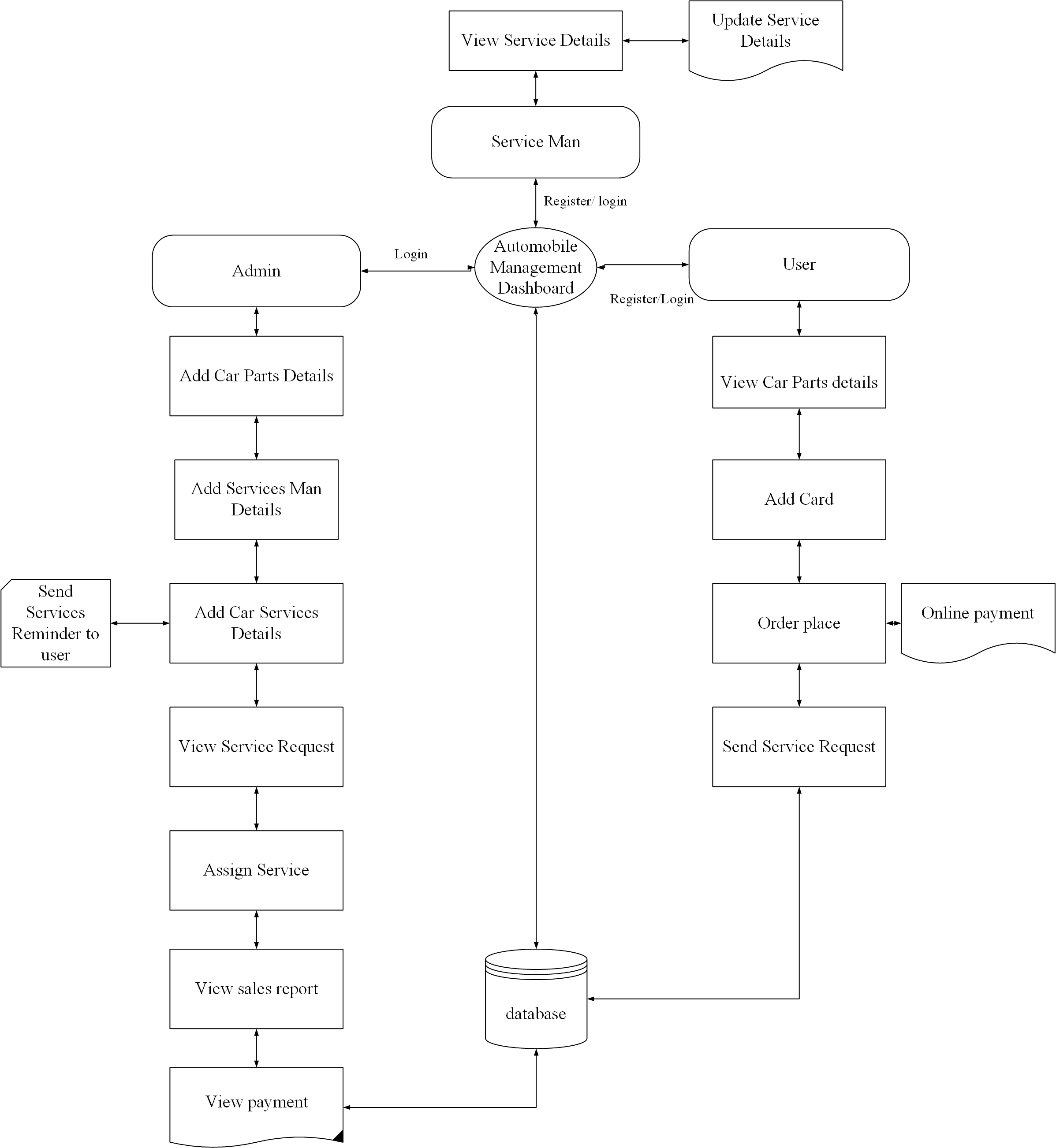
The next stage is to create the Level 1 Data Flow Diagram. This highlights the main functions carried out by the system. As a rule, to describe the system was using between two and seven functions - two being a simple system and seven being a complicated system. This enables us to keep the model manageable on screen or paper.

****

**Figure 1.2 Data Flow Diagram Level 1**

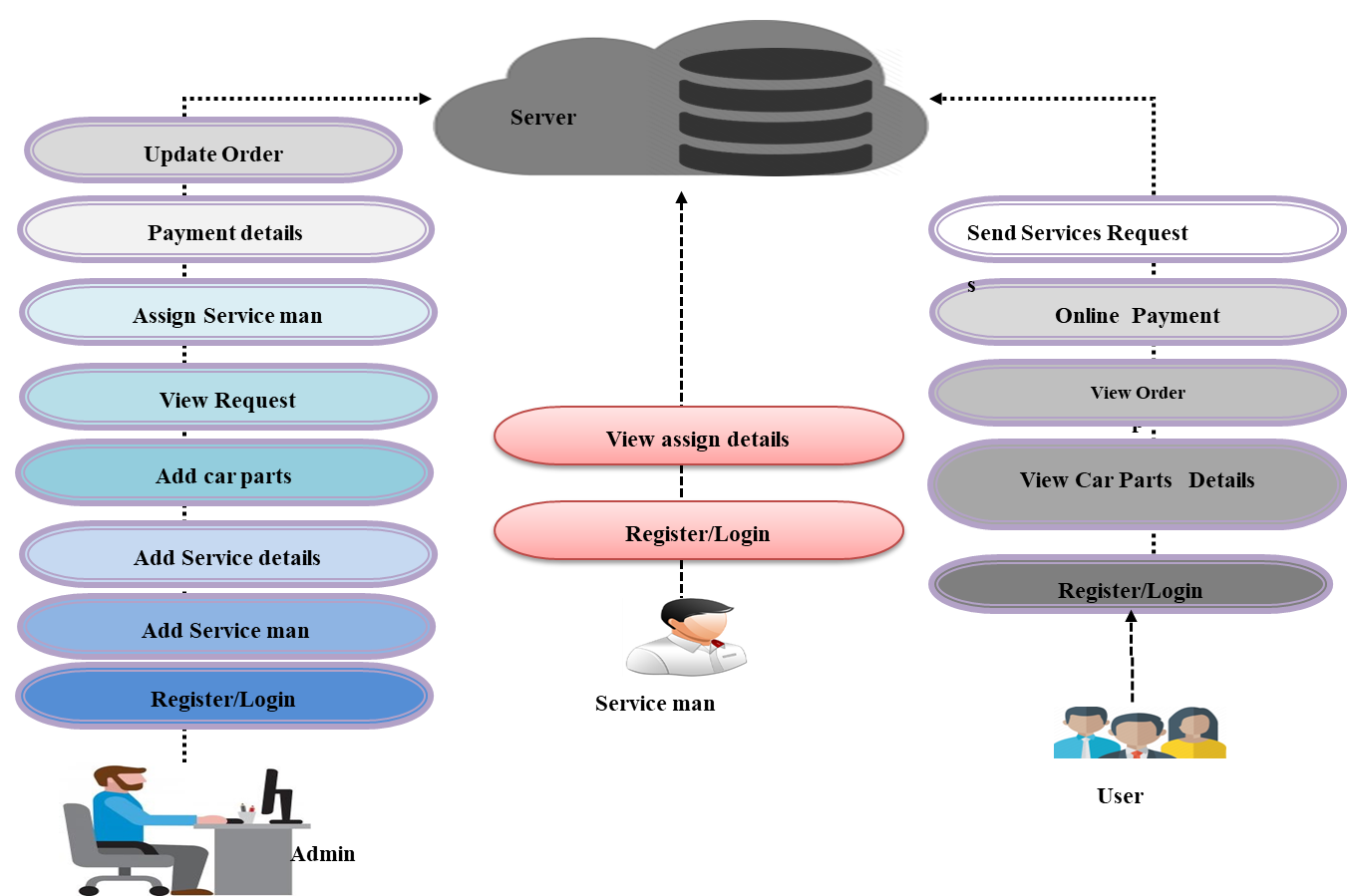
**DFD LEVEL-2**

A Data Flow Diagram (DFD) tracks processes and their data paths within the business or system boundary under investigation. A DFD defines each domain boundary and illustrates the logical movement and transformation of data within the defined boundary. The diagram shows 'what' input data enters the domain, 'what' logical processes the domain applies to that data, and 'what' output data leaves the domain. Essentially, a DFD is a tool for process modeling and one of the oldest



**Figure 1.3 Data Flow Diagram Level 2**

**3.2 ARCHITECTURAL DIAGRAMS**

****

**3.4 TABLE DESIGN**

## Table structure for table car\_cart

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** |
| id | int(11) | Yes | NULL |
| uname | varchar(20) | Yes | NULL |
| pid | int(11) | Yes | NULL |
| status | int(11) | Yes | NULL |
| rdate | varchar(20) | Yes | NULL |
| bill\_id | int(11) | Yes | NULL |
| price | int(11) | Yes | NULL |
| category | varchar(30) | Yes | NULL |
|  |

## Table structure for table car\_category

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** |
| id | int(11) | Yes | NULL |
| category | varchar(30) | Yes | NULL |

## Table structure for table car\_login

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** |
| username | varchar(20) | Yes | NULL |
| password | varchar(20) | Yes | NULL |

## Table structure for table car\_man

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** |
| id | int(11) | Yes | NULL |
| name | varchar(20) | Yes | NULL |
| mobile | bigint(20) | Yes | NULL |
| email | varchar(40) | Yes | NULL |
| city | varchar(20) | Yes | NULL |
| uname | varchar(20) | Yes | NULL |
| pass | varchar(20) | Yes | NULL |
| create\_date | varchar(20) | Yes | NULL |
|  |

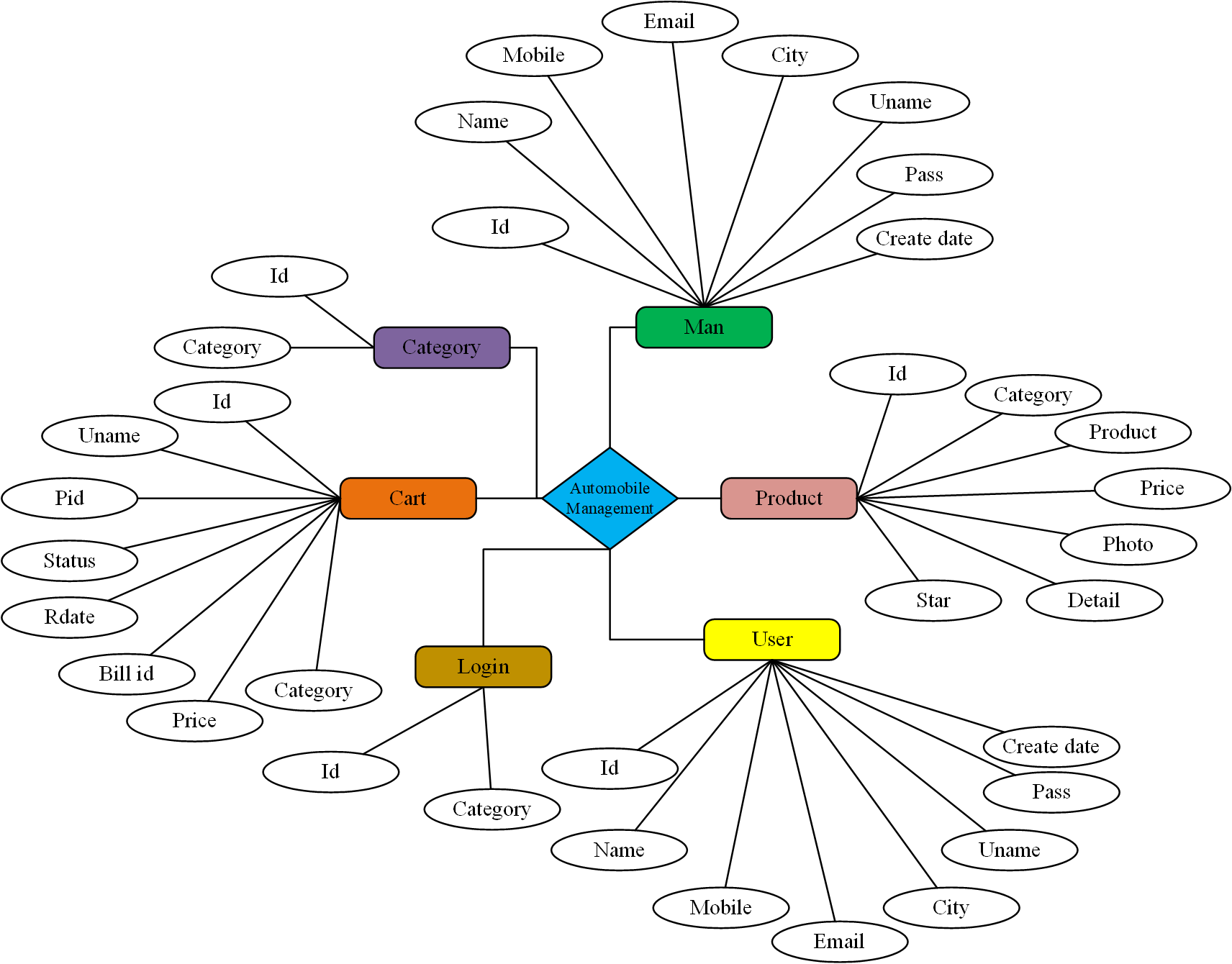
## Table structure for table car\_product

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** |
| id | int(11) | Yes | NULL |
| category | varchar(30) | Yes | NULL |
| product | varchar(50) | Yes | NULL |
| price | bigint(20) | Yes | NULL |
| photo | varchar(100) | Yes | NULL |
| detail | varchar(100) | Yes | NULL |
| star | int(11) | Yes | NULL |

## Table structure for table car\_user

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Default** |
| id | int(11) | Yes | NULL |
| name | varchar(20) | Yes | NULL |
| mobile | bigint(20) | Yes | NULL |
| email | varchar(40) | Yes | NULL |
| city | varchar(20) | Yes | NULL |
| uname | varchar(20) | Yes | NULL |
| pass | varchar(20) | Yes | NULL |
| create\_date | varchar(20) | Yes | NULL |

**3.5 ER Diagram**



**CHAPTER 4**

**IMPLEMENTATION**

**4.1 SOURCE CODE**

from flask import Flask

from flask import Flask, render\_template, Response, redirect, request, session, abort, url\_for

import os

import base64

from PIL import Image

from datetime import datetime

from datetime import date

import datetime

import random

from random import seed

from random import randint

from werkzeug.utils import secure\_filename

from flask import send\_file

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import threading

import time

import shutil

import hashlib

import urllib.request

import urllib.parse

from urllib.request import urlopen

import webbrowser

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

passwd="",

charset="utf8",

database="car\_automobile"

)

app = Flask(\_\_name\_\_)

##session key

app.secret\_key = 'abcdef'

UPLOAD\_FOLDER = 'static/upload'

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

#####

@app.route('/',methods=['POST','GET'])

def index():

cnt=0

act=""

msg=""

return render\_template('index.html',msg=msg,act=act)

@app.route('/login\_admin',methods=['POST','GET'])

def login\_admin():

cnt=0

act=""

msg=""

if request.method == 'POST':

username1 = request.form['uname']

password1 = request.form['pass']

mycursor = mydb.cursor()

mycursor.execute("SELECT count(\*) FROM car\_login where username=%s && password=%s",(username1,password1))

myresult = mycursor.fetchone()[0]

if myresult>0:

session['username'] = username1

#result=" Your Logged in sucessfully\*\*"

return redirect(url\_for('admin'))

else:

msg="Invalid Username or Password!"

return render\_template('login\_admin.html',msg=msg,act=act)

@app.route('/login',methods=['POST','GET'])

def login():

cnt=0

act=""

msg=""

if request.method == 'POST':

username1 = request.form['uname']

password1 = request.form['pass']

mycursor = mydb.cursor()

mycursor.execute("SELECT count(\*) FROM car\_user where uname=%s && pass=%s",(username1,password1))

myresult = mycursor.fetchone()[0]

if myresult>0:

session['username'] = username1

result=" Your Logged in sucessfully\*\*"

return redirect(url\_for('userhome'))

else:

msg="Invalid Username or Password!"

result="Your logged in fail!!!"

return render\_template('login.html',msg=msg,act=act)

@app.route('/login\_ser',methods=['POST','GET'])

def login\_ser():

cnt=0

act=""

msg=""

if request.method == 'POST':

username1 = request.form['uname']

password1 = request.form['pass']

mycursor = mydb.cursor()

mycursor.execute("SELECT count(\*) FROM car\_man where uname=%s && pass=%s",(username1,password1))

myresult = mycursor.fetchone()[0]

if myresult>0:

session['username'] = username1

result=" Your Logged in sucessfully\*\*"

return redirect(url\_for('ser\_home'))

else:

msg="Invalid Username or Password!"

return render\_template('login\_ser.html',msg=msg,act=act)

@app.route('/register', methods=['GET', 'POST'])

def register():

msg=""

act=request.args.get("act")

mycursor = mydb.cursor()

if request.method=='POST':

name=request.form['name']

city=request.form['city']

mobile=request.form['mobile']

email=request.form['email']

uname=request.form['uname']

pass1=request.form['pass']

mycursor.execute("SELECT count(\*) FROM car\_user where uname=%s",(uname,))

myresult = mycursor.fetchone()[0]

if myresult==0:

mycursor.execute("SELECT max(id)+1 FROM car\_user")

maxid = mycursor.fetchone()[0]

if maxid is None:

maxid=1

now = date.today() #datetime.datetime.now()

rdate=now.strftime("%d-%m-%Y")

sql = "INSERT INTO car\_user(id,name,mobile,email,city,uname,pass,create\_date) VALUES (%s,%s,%s,%s,%s,%s,%s,%s)"

val = (maxid,name,mobile,email,city,uname,pass1,rdate)

mycursor.execute(sql, val)

mydb.commit()

print(mycursor.rowcount, "Registered Success")

msg="success"

#if cursor.rowcount==1:

# return redirect(url\_for('index',act='1'))

else:

msg='Already Exist'

return render\_template('register.html', msg=msg)

@app.route('/admin', methods=['GET', 'POST'])

def admin():

msg=""

act=""

email=""

mess=""

data=[]

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM car\_service where service\_man='' order by id")

data = mycursor.fetchall()

return render\_template('admin.html',msg=msg,data=data)

@app.route('/assign', methods=['GET', 'POST'])

def assign():

msg=""

act=""

email=""

mess=""

sid = request.args.get('sid')

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM car\_man")

sdata = mycursor.fetchall()

if request.method=='POST':

service\_man=request.form['service\_man']

mycursor.execute('update car\_service set service\_man=%s,status=1 WHERE id = %s', (service\_man, sid))

mydb.commit()

msg="ok"

return render\_template('assign.html',msg=msg,sdata=sdata)

@app.route('/add\_product', methods=['GET', 'POST'])

def add\_product():

msg=""

act = request.args.get('act')

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM car\_category")

data = mycursor.fetchall()

if request.method=='POST':

category=request.form['category']

product=request.form['product']

price=request.form['price']

detail=request.form['detail']

file = request.files['file']

mycursor.execute("SELECT max(id)+1 FROM car\_product")

maxid = mycursor.fetchone()[0]

if maxid is None:

maxid=1

try:

if file.filename == '':

flash('No selected file')

return redirect(request.url)

if file:

fn=file.filename

fnn="P"+str(maxid)+fn

#fn1 = secure\_filename(fn)

file.save(os.path.join(app.config['UPLOAD\_FOLDER'], fnn))

except:

print("dd")

photo="P"+str(maxid)+fn

sql = "INSERT INTO car\_product(id,category,product,price,photo,detail) VALUES (%s, %s, %s, %s, %s, %s)"

val = (maxid,category,product,price,photo,detail)

mycursor.execute(sql, val)

mydb.commit()

#print(mycursor.rowcount, "Registered Success")

result="sucess"

if mycursor.rowcount==1:

return redirect(url\_for('view\_product'))

else:

msg='Already Exist'

mycursor.execute("SELECT \* FROM car\_product")

data2 = mycursor.fetchall()

return render\_template('add\_product.html',msg=msg,data=data,data2=data2)

@app.route('/view\_product', methods=['GET', 'POST'])

def view\_product():

msg=""

act = request.args.get('act')

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM car\_product")

data = mycursor.fetchall()

if act=="del":

did = request.args.get('did')

mycursor.execute('delete from car\_product WHERE id = %s', (did, ))

mydb.commit()

return redirect(url\_for('view\_product'))

return render\_template('view\_product.html',msg=msg,data=data)

@app.route('/add\_service\_man', methods=['GET', 'POST'])

def add\_service\_man():

msg=""

email=""

mess=""

act = request.args.get('act')

mycursor = mydb.cursor()

if request.method=='POST':

name=request.form['name']

mobile=request.form['mobile']

email=request.form['email']

city=request.form['city']

uname=request.form['uname']

pass1=request.form['pass']

now = date.today() #datetime.datetime.now()

rdate=now.strftime("%d-%m-%Y")

mycursor.execute("SELECT max(id)+1 FROM car\_man")

maxid = mycursor.fetchone()[0]

if maxid is None:

maxid=1

sql = "INSERT INTO car\_man(id,name,mobile,email,city,uname,pass,create\_date) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"

val = (maxid,name,mobile,email,city,uname,pass1,rdate)

mycursor.execute(sql, val)

mydb.commit()

#print(mycursor.rowcount, "Registered Success")

msg="success"

mycursor.execute("SELECT \* FROM car\_man")

data = mycursor.fetchall()

if act=="del":

did = request.args.get('did')

mycursor.execute('delete from car\_man WHERE id = %s', (did, ))

mydb.commit()

return redirect(url\_for('add\_service\_man'))

return render\_template('add\_service\_man.html',msg=msg,data=data,email=email,mess=mess)

@app.route('/userhome', methods=['GET', 'POST'])

def userhome():

msg=""

cnt=0

uname=""

act = request.args.get('act')

cat = request.args.get('cat')

if 'username' in session:

uname = session['username']

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM car\_user where uname=%s",(uname,))

usr = mycursor.fetchone()

mycursor.execute("SELECT \* FROM car\_category")

data2 = mycursor.fetchall()

cc=""

if cat is None:

cc=""

else:

cc="1"

if request.method=='POST':

getval=request.form['getval']

cat="%"+getval+"%"

prd="%"+getval+"%"

det="%"+getval+"%"

mycursor.execute("SELECT \* FROM car\_product where category like %s || product like %s || detail like %s order by star desc",(cat,prd,det))

data = mycursor.fetchall()

elif cc=="1":

mycursor.execute("SELECT \* FROM car\_product where category=%s order by star desc",(cat,))

data = mycursor.fetchall()

else:

mycursor.execute("SELECT \* FROM car\_product order by star desc")

data = mycursor.fetchall()

now = datetime.datetime.now()

rdate=now.strftime("%d-%m-%Y")

if act=="cart":

pid = request.args.get('pid')

mycursor.execute('SELECT count(\*) FROM car\_cart WHERE uname=%s && pid = %s && status=0', (uname, pid))

num = mycursor.fetchone()[0]

mycursor.execute("SELECT \* FROM car\_product where id=%s",(pid,))

pdata = mycursor.fetchone()

price=pdata[3]

cat=pdata[1]

if num==0:

mycursor.execute("SELECT max(id)+1 FROM car\_cart")

maxid = mycursor.fetchone()[0]

if maxid is None:

maxid=1

sql = "INSERT INTO car\_cart(id, uname, pid, status, rdate, price,category) VALUES (%s, %s, %s, %s, %s, %s, %s)"

val = (maxid, uname, pid, '0', rdate, price, cat)

mycursor.execute(sql,val)

mydb.commit()

msg="ok"

#return redirect(url\_for('userhome'))

mycursor.execute("SELECT count(\*) FROM car\_cart where uname=%s && status=0",(uname,))

cnt = mycursor.fetchone()[0]

if cnt>0:

msg="1"

else:

msg=""

return render\_template('userhome.html',msg=msg,usr=usr,data=data,cnt=cnt,data2=data2)

@app.route('/cart', methods=['GET', 'POST'])

def cart():

act=""

pid=""

did=""

amount=""

data=[]

st=""

ct=""

if 'username' in session:

uname = session['username']

cursor = mydb.cursor()

cursor.execute("SELECT count(\*) FROM car\_cart where uname=%s && status=0",(uname, ))

cnt = cursor.fetchone()[0]

if cnt>0:

act="1"

else:

act=""

cursor.execute('SELECT count(\*) FROM car\_cart where uname=%s and status=0', (uname, ))

cnt = cursor.fetchone()[0]

if cnt>0:

ct="1"

cursor.execute('SELECT c.id,p.product,p.price,p.detail,p.photo,c.rdate FROM car\_cart c,car\_product p where c.pid=p.id and c.uname=%s and c.status=0', (uname, ))

data = cursor.fetchall()

cursor.execute("SELECT \* FROM car\_cart where uname=%s && status=0",(uname, ))

dr = cursor.fetchall()

amt=0

if request.method=='POST':

rid=request.form.getlist('rid[]')

qty=request.form.getlist('qty[]')

print(qty)

i=0

for rr in rid:

cursor.execute("SELECT \* FROM car\_cart where id=%s",(rr, ))

dr2 = cursor.fetchone()

prc=dr2[6]

qt=int(qty[i])

amo=qt\*prc

amt+=amo

st="1"

cursor.execute("update car\_cart set amount=%s,qty=%s where id=%s",(amo,qt,rr))

mydb.commit()

i+=1

return render\_template('cart.html', data=data, amount=amt,act=act,st=st,ct=ct)

@app.route('/payment', methods=['GET', 'POST'])

def payment():

msg=""

mess=""

email=""

uname=""

amount=0

if 'username' in session:

uname = session['username']

now = datetime.datetime.now()

rdate=now.strftime("%d-%m-%Y")

cursor = mydb.cursor()

#print("uname="+uname)

cursor.execute("SELECT \* FROM car\_user where uname=%s",(uname, ))

rd=cursor.fetchone()

name=rd[1]

mob1=rd[2]

email=rd[3]

cursor.execute("SELECT \* FROM car\_cart where uname=%s && status=0",(uname, ))

d1=cursor.fetchall()

for r1 in d1:

aa=int(r1[6])\*int(r1[8])

print(aa)

amount+=aa

x=0

if request.method=='POST':

card=request.form['card']

cursor.execute("SELECT max(id)+1 FROM car\_purchase")

maxid = cursor.fetchone()[0]

if maxid is None:

maxid=1

mess="Dear "+name+", Amount Rs."+str(amount)+", paid and ordered, Bill ID: "+str(maxid)

cursor.execute('update car\_cart set status=1,bill\_id=%s WHERE uname=%s && status=0', (maxid, uname ))

mydb.commit()

sql = "INSERT INTO car\_purchase(id, uname, amount, rdate) VALUES (%s, %s, %s, %s)"

val = (maxid, uname, amount, rdate)

cursor.execute(sql,val)

mydb.commit()

msg="success"

return render\_template('payment.html', msg=msg, amount=amount,email=email,mess=mess)

@app.route('/user\_order', methods=['GET', 'POST'])

def user\_order():

msg=""

mess=""

email=""

uname=""

amount=0

data=[]

if 'username' in session:

uname = session['username']

cursor = mydb.cursor()

cursor.execute("SELECT \* FROM car\_user where uname=%s",(uname, ))

rd=cursor.fetchone()

cursor.execute("SELECT \* FROM car\_purchase where uname=%s order by id desc",(uname, ))

dd=cursor.fetchall()

for d1 in dd:

dt=[]

dt.append(d1[0])

dt.append(d1[1])

dt.append(d1[2])

dt.append(d1[3])

dt.append(d1[4])

dtt=[]

cursor.execute("SELECT \* FROM car\_cart where bill\_id=%s",(d1[0], ))

d3=cursor.fetchall()

for d4 in d3:

dt1=[]

cursor.execute("SELECT \* FROM car\_product where id=%s",(d4[2], ))

d2=cursor.fetchone()

dt1.append(d2[2])

dt1.append(d4[6])

dt1.append(d4[7])

dt1.append(d4[8])

dt1.append(d4[9])

dtt.append(dt1)

dt.append(dtt)

data.append(dt)

print(data)

return render\_template('user\_order.html', msg=msg, data=data)

@app.route('/view\_order', methods=['GET', 'POST'])

def view\_order():

msg=""

act=request.args.get("act")

mess=""

email=""

uname=""

amount=0

data=[]

if 'username' in session:

uname = session['username']

cursor = mydb.cursor()

cursor.execute("SELECT \* FROM car\_purchase order by id desc")

dd=cursor.fetchall()

for d1 in dd:

dt=[]

dt.append(d1[0])

dt.append(d1[1])

dt.append(d1[2])

dt.append(d1[3])

dt.append(d1[4])

dtt=[]

cursor.execute("SELECT \* FROM car\_cart where bill\_id=%s",(d1[0], ))

d3=cursor.fetchall()

for d4 in d3:

dt1=[]

cursor.execute("SELECT \* FROM car\_product where id=%s",(d4[2], ))

d2=cursor.fetchone()

dt1.append(d2[2])

dt1.append(d4[6])

dt1.append(d4[7])

dt1.append(d4[8])

dt1.append(d4[9])

dtt.append(dt1)

dt.append(dtt)

data.append(dt)

print(data)

if act=="yes":

did=request.args.get("did")

mycursor.execute("update car\_purchase set status=1 where id=%s",(did,))

mydb.commit()

return redirect(url\_for('view\_order'))

return render\_template('view\_order.html', msg=msg, data=data)

@app.route('/service\_req', methods=['GET', 'POST'])

def service\_req():

msg=""

email=""

mess=""

act = request.args.get('act')

uname=""

if 'username' in session:

uname = session['username']

mycursor = mydb.cursor()

if request.method=='POST':

details=request.form['details']

mobile=request.form['mobile']

city=request.form['city']

now = date.today() #datetime.datetime.now()

rdate=now.strftime("%d-%m-%Y")

mycursor.execute("SELECT max(id)+1 FROM car\_service")

maxid = mycursor.fetchone()[0]

if maxid is None:

maxid=1

sql = "INSERT INTO car\_service(id,uname,details,mobile,city,service\_man,reply,amount,status,create\_date) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"

val = (maxid,uname,details,mobile,city,'','','0','0',rdate)

mycursor.execute(sql, val)

mydb.commit()

#print(mycursor.rowcount, "Registered Success")

msg="success"

mycursor.execute("SELECT \* FROM car\_service where uname=%s order by id desc",(uname,))

data = mycursor.fetchall()

return render\_template('service\_req.html',msg=msg,data=data,email=email,mess=mess)

@app.route('/ser\_home', methods=['GET', 'POST'])

def ser\_home():

msg=""

cnt=0

uname=""

act = request.args.get('act')

if 'username' in session:

uname = session['username']

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM car\_man where uname=%s",(uname,))

usr = mycursor.fetchone()

mycursor.execute("SELECT \* FROM car\_service where service\_man=%s order by id desc",(uname,))

data2 = mycursor.fetchall()

return render\_template('ser\_home.html',msg=msg,usr=usr,data2=data2)

@app.route('/update', methods=['GET', 'POST'])

def update():

msg=""

cnt=0

uname=""

act = request.args.get('act')

sid = request.args.get('sid')

if 'username' in session:

uname = session['username']

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM car\_man where uname=%s",(uname,))

usr = mycursor.fetchone()

if request.method=='POST':

reply=request.form['reply']

amount=request.form['amount']

mycursor.execute("update car\_service set reply=%s,amount=%s,status=2 where id=%s",(reply,amount,sid))

mydb.commit()

msg="ok"

return render\_template('update.html',msg=msg,usr=usr)

@app.route('/pay', methods=['GET', 'POST'])

def pay():

msg=""

cnt=0

uname=""

act = request.args.get('act')

sid = request.args.get('sid')

if 'username' in session:

uname = session['username']

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM car\_user where uname=%s",(uname,))

usr = mycursor.fetchone()

if request.method=='POST':

upid=request.form['upid']

mycursor.execute("update car\_service set status=3 where id=%s",(sid,))

mydb.commit()

msg="ok"

return render\_template('pay.html',msg=msg,usr=usr)

@app.route('/view\_service', methods=['GET', 'POST'])

def view\_service():

msg=""

cnt=0

uname=""

act = request.args.get('act')

sid = request.args.get('sid')

if 'username' in session:

uname = session['username']

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM car\_service order by id desc")

data = mycursor.fetchall()

return render\_template('view\_service.html',msg=msg,data=data)

@app.route('/logout')

def logout():

# remove the username from the session if it is there

session.pop('username', None)

return redirect(url\_for('index'))

if \_\_name\_\_ == "\_\_main\_\_":

app.secret\_key = os.urandom(12)

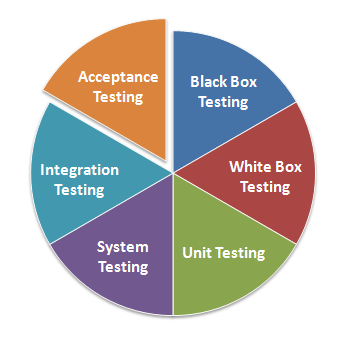
app.run(debug=True,host='0.0.0.0', port=5000)

**4.2 OUTPUT SCREEN**

**TESTING**

**Testing Approach**

A test case is an asset of data that the system will process as normal input. The strategies that we have used in our project are,



**SYSTEM TESTING**

Testing is the stage of implementation of which aimed at ensuring that the system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct the goal will be achieved. The candidates system subject to a variety of tests. Online response, volume, stress, recovery, security and usability tests. A series of testing are performed for the proposed system before the system is ready for user acceptance testing.

**UNIT TESTING**

The procedure level testing is made first. By giving improper inputs, the errors occurred are noted and eliminated .Then the web form level is made.

**INTEGRATION TESTING:**

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions. Thus the system testing is a confirmation that all its correct and an opportunity to show the user that the system works.

**VALIDATION TESTING**

Software Testing - Validation Testing. The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements. Validation Testing ensures that the product actually meets the client's needs.

**REGRESSION TESTING**

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

**CHAPTER 6**

**CONCLUSION**

In conclusion, the Automobile Management System is an essential tool for automobile service centers to manage their operations efficiently and provide quality service to their customers. The existing manual approach is inefficient, prone to errors, and limits the service center's ability to provide quality service to its customers. The proposed Automobile Management System aims to automate and streamline the service center's operations, improve its efficiency, and enhance customer satisfaction. The system will automate various tasks such as inventory management, service management, billing, and payment, among others. The system will have different user types, each with specific functionalities, and provide an easy-to-use interface that is designed to enhance user experience. The proposed system will also provide security features to protect the system from unauthorized access and data breaches. In summary, the Automobile Management System is a must-have tool for automobile service centers that want to streamline their operations, improve their customer service, and stay ahead of the competition.

**FUTURE ENHANCEMENT**

* Mobile app: Developing a mobile application for the system can improve the system's accessibility, enabling customers to access the system from their mobile devices.
* Artificial Intelligence: Implementing artificial intelligence (AI) in the system can improve its efficiency and accuracy, enhancing the quality of service delivery.
* Predictive analysis: Implementing predictive analysis in the system can enable the management to predict customer demand, identify trends, and make informed decisions.
* Integration with other systems: Integrating the Automobile Management System with other systems, such as accounting systems, can improve the system's functionality and efficiency.
* IoT integration: Integrating the system with IoT devices such as sensors, cameras, and other smart devices can improve the service center's efficiency, reduce service delivery time, and enhance customer satisfaction.
* Advanced reporting: Enhancing the system's reporting capabilities by implementing advanced reporting features such as data visualization, data analytics, and business intelligence tools can improve the management's ability to make informed decisions.
* Automated scheduling: Implementing an automated scheduling system can improve the service center's ability to manage service requests, prioritize tasks, and reduce service delivery time.

**CHAPTER 7**

**BIBLIOGRAPHY**

* "Automotive Industry: Opportunities and Challenges Ahead" by KPMG International: This report discusses the current state and future trends in the automotive industry, including the role of technology and innovation.
* "Automotive Industry: Digital Transformation and Disruptive Innovation" by Deloitte: This report discusses the impact of digital transformation and disruptive innovation on the automotive industry, including topics such as connected cars, autonomous vehicles, and mobility services.
* "Automotive Service Management: Principles into Practice" by Andrew Rezin: This book provides an overview of service management principles and practices in the automotive industry, including topics such as customer service, workflow management, and service quality.
* "Automotive Engineering: Powertrain, Chassis System and Vehicle Body" edited by David Crolla: This book provides an overview of automotive engineering, including topics such as powertrain, chassis systems, and vehicle body design.
* "Automotive Technology: Principles, Diagnosis, and Service" by James D. Halderman: This book provides an introduction to automotive technology, including topics such as vehicle systems, diagnosis, and repair.