

*A Project Report*

*On*

## **BLSS CARS USING PYTHON TKINTER**



Submitted in partial fulfilments of the requirements  
for the award of the degree of

**MASTER OF COMPUTER APPLICATIONS**

**SUBMITTED BY**

**M. Satyavathi (23A91F00G9)**

**K. Lavanya (23A91F00G3)**

**K. SumaSri (23A91F00F3)**

**G. Bhuvana Vilashitha (23A91F00E9)**

Under the Esteemed Guidance of

**Mr. A. Rajendra Prasad**



**DEPARTMENT OF MCA**

**ADITYA ENGINEERING COLLEGE**

**An Autonomous Institution**

(Approved by AICTE, Affiliated to JNTUK, Kakinada & Accredited by NBA, NAAC with 'A+')

Recognized by UGC under the sections 2(f) and 12(b) of the UGC act 1956 surampalem-533437, East Godavari

District, Andhra Pradesh. 2023-2025

# **ADITYA ENGINEERING COLLEGE**

## **An Autonomous Institution**

(Approved by AICTE, Affiliated to JNTUK, Kakinada & Accredited by NBA, NAAC with 'A+')  
Recognized by UGC under the sections 2(f) and 12(b) of the UGC act 1956 surampalem-533437, East  
Godavari District, Andhra Pradesh. **2023-2025**

### **DEPARTMENT OF MCA**



### **CERTIFICATE**

This is to certify that the project work entitled, “BLSS CARS USING PYTHON TKINTER” is a bonafide work of **M. Satyavathi (23A91F00G9), K. Lavanya (23A91F00G3), K. Suma Sri (23A91F00F3), G. Bhuvana Vilashitha (23A91F00E9)** submitted to the faculty of Master of Computer Applications, in partial fulfilment of the requirements for the award of the Degree of **MASTER OF COMPUTER APPLICATIONS** from Aditya Engineering College, Surampalem for the academic year 2024-2025.

#### **Project guide**

**Mr. A. Rajendra Prasad, MBA**

Department of MCA,  
Aditya Engineering College,  
Surampalem-533437

#### **Head of the Department**

**Mr. G. Hema Sundar Rao, MCA**

Department of MCA,  
Aditya Engineering College,  
Surampalem-533437

**External Examiner**

## **DECLARATION**

I hereby declare that the project entitled **“BLSS CARS USING PYTHON TKINTER”** done on my own and submitted to **Aditya Engineering College, Surampalem** has been carried out by me alone under the guidance of **Mr. A. Rajendra Prasad**

**Place: Surampalem**

**Date:**

**M. Satyavathi (23A91F00G9)**

**K. Lavanya (23A91F00G3)**

**K. Suma Sri (23A91F00F3)**

**G. Bhuvana Vilashitha (23A91F00E9)**

## ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them.

The first person I would like to thank is my guide **Mr. A. Rajendra Prasad, Aditya Engineering College, Surampalem**. His wide knowledge and logical way of thinking have made a deep impression on me. His understanding, encouragement and personal guidance have provided the basis for this project. He is a source of inspiration for innovative ideas and her kind of support is well known to all her students and colleagues.

I wish to thank **Mrs. G. Hema Sundar Rao, Head of the Department, Aditya Engineering College, Surampalem**, who has extended his support for the success of this project.

I also wish to thank **Sri. M. Sreenivasa Reddy, Principal, Aditya Engineering College, Surampalem**, who has extended his support for the success of this project.

I would like to thank all the **Management & Technical Supporting Staff** for their timely help throughout the project.

## ABSTRACT

Car showrooms today facing the challenges of staying efficient, organized, and customer-friendly in an increasingly competitive market. Most small and medium-sized showrooms still use outdated manual systems or struggle with expensive and complex software. These solutions don't meet the practical needs of showrooms, leading to inefficiency, errors, and missed opportunities.

To address all this issues, we developed **BLSS Cars**, a modern desktop application that is simple, affordable, and designed specifically for car showrooms. It is a Python-based project developed using the Tkinter library, aimed at optimizing and streamlining operations within car showrooms. The application is designed with two role-specific modules: **Admin** and **Salesperson**, each catering to distinct responsibilities and workflows to ensure efficient management and improved user productivity.

The **Admin module** provides features such as employee management, comprehensive car inventory tracking, detailed booking statistics, and an overview of showroom operations. This module empowers administrators to make data-driven decisions effortlessly.

The **Salesperson module**, on the other hand, allows users to edit their profiles, view car details, manage customer bookings, and access personalized booking records, thereby enhancing their ability to serve customers effectively.

The backbone of the application's data storage is **SQLite**, a lightweight, embedded database system. SQLite was chosen for its simplicity, reliability, and seamless integration with Python, making it ideal for a desktop application. All key data, including employee records, car details, booking information, and user credentials, are stored securely in SQLite.

**BLSS Cars** stands out by offering a cost-effective, user-friendly alternative to complex enterprise software. It bridges the gap for small to medium-sized showrooms that require tailored, localized solutions without the overhead of large-scale systems.

# CONTENTS

CHAPTER NO	PAGE NO
<b>Chapter 1: INTRODUCTION</b>	
1.1 Brief information about the project	01
1.2 Motivation and Contribution of project	01
1.3 Objective of the project	02
1.4 Organization of the project	02
<b>Chapter 2: SYSTEM ANALYSIS</b>	
2.1 Existing System	03
2.2 Proposed System	04
2.3 Feasibility Study	05
2.4 Functional Requirements	06
2.5 Non-Functional Requirements	07
2.6 Requirements Specification	
2.6.1 Minimum Hardware Requirements	07
2.6.2 Software Requirements	07
<b>Chapter 3: SYSTEM DESIGN</b>	
3.1 Introduction	08
3.2 System Architecture	09
3.3 Module Description	10
3.4 UML Diagrams	12
3.4.1 Use case Diagram	12
3.4.2 Class Diagram	13
3.4.3 Sequence Diagram	15
3.4.4 Activity Diagram	17
<b>Chapter 4: TECHNOLOGY DESCRIPTION</b>	
4.1 Introduction to Python	20
4.2 Applications of Python	23
4.3 Technologies Used	24

<b>Chapter 5: SAMPLE CODE</b>	<b>27</b>
<b>Chapter 6: TESTING</b>	
<b>6.1</b> Introduction	<b>31</b>
<b>6.2</b> System Testing and Implementation	<b>31</b>
<b>6.3</b> Testing Techniques	<b>32</b>
<b>6.4</b> Testing Strategies	<b>33</b>
<b>Chapter 7: SCREENSHOTS</b>	<b>35</b>
<b>Chapter 8: CONCLUSION</b>	<b>43</b>
<b>8.1</b> Future Scope	<b>43</b>

## **LIST OF TABLES**

<b>S.NO</b>	<b>TABLE NO</b>	<b>NAME OF THE TABLE</b>	<b>PAGE NO</b>
<b>01</b>	<b>3.4.1</b>	Use Case Diagram table	<b>13</b>
<b>02</b>	<b>3.4.2</b>	Class Diagram table	<b>15</b>
<b>03</b>	<b>3.4.3</b>	Sequence Diagram Symbols Representation	<b>16</b>
<b>04</b>	<b>3.4.4</b>	Activity Diagram table	<b>17</b>



## **LIST OF FIGURES**

<b>FIGURE NO</b>	<b>NAME OF THE FIGURE</b>	<b>PAGE NO</b>
<b>3.2</b>	System architecture	<b>9</b>
<b>3.4.1.1</b>	Use case diagram for system	<b>13</b>
<b>3.4.2.1</b>	Class diagram for system	<b>15</b>
<b>3.4.3.1</b>	Sequence diagram for System	<b>16</b>
<b>3.4.4.1</b>	Activity diagram for System	<b>19</b>
<b>4.1</b>	Python diagram	<b>22</b>

## **LIST OF SCREENS**

<b>S.NO</b>	<b>SCREEN NO</b>	<b>NAME OF THE SCREEN</b>	<b>PAGE NO</b>
<b>01</b>	<b>7.1</b>	Splash Screen	<b>35</b>
<b>02</b>	<b>7.2</b>	Welcome Screen	<b>36</b>
<b>03</b>	<b>7.3</b>	Admin Login	<b>37</b>
<b>04</b>	<b>7.4</b>	Admin Dashboard	<b>38</b>
<b>05</b>	<b>7.5</b>	SalesPerson Dashboard	<b>39</b>
<b>06</b>	<b>7.6</b>	Employee Registration	<b>40</b>
<b>07</b>	<b>7.7</b>	Car Models	<b>41</b>
<b>08</b>	<b>7.8</b>	Car Booking	<b>42</b>

# 1.INTRODUCTION

## 1.1 Brief Information about the project:

BLSS Cars is a desktop application created to help car showrooms manage their day-to-day operations more easily. Built using Python and SQLite, the app has two main sections: one for Admins and one for Salespeople. Admins can manage employees, keep track of car inventory, and view booking details. Salespeople can update their profiles, manage customer bookings, and see car information quickly, making their work more efficient.

The app is designed to be simple, affordable, and works offline, making it ideal for small to medium-sized showrooms. Unlike complicated and expensive systems, BLSS Cars provides the essential features needed for smooth showroom management in an easy-to-use and cost-effective way. It can also grow with the showroom as its needs expand.

## 1.2 Motivation and contribution of the project:

The motivation behind BLSS Cars is driven by the challenges faced by small and medium-sized car showrooms in managing daily operations efficiently. Many showrooms still rely on outdated manual systems or expensive, complex software that doesn't cater to their specific needs. These systems often lead to inefficiencies, errors, and missed opportunities. We wanted to create an affordable, easy-to-use solution that helps streamline showroom operations without the burden of complex features or high costs.

BLSS Cars contributes by offering a simple, cost-effective desktop application that makes key tasks like employee management, car inventory tracking, and customer booking more efficient. It empowers salespeople with tools to improve customer service, while providing administrators with insights to manage the showroom better. With both online and offline functionality, the app ensures reliable performance even in areas with limited internet access. By addressing the real-world challenges of car showrooms, BLSS Cars helps improve operational efficiency, enhance customer satisfaction, and reduce the time spent on manual tasks, making it a valuable asset for any showroom looking to modernize.

### 1.3 Objective of the project:

- Simplify tasks such as employee management, car inventory tracking, and booking management to reduce manual effort and errors.
- Equip salespeople with tools to efficiently manage customer bookings, update profiles, and access car details to improve customer service.
- Enable showroom administrators to monitor performance, view real-time statistics, and make informed decisions based on data.
- Provide an affordable alternative to complex software, making advanced showroom management tools accessible to small and medium-sized showrooms.
- Ensure offline functionality, making the application reliable in areas with limited or no internet access.
- Create a scalable system that can grow with the showroom, allowing for future features and expansions as the business evolves.

### 1.4 Organization of the project:

- **System Analysis:** This chapter mainly consists of description of current system, proposed system, and requirement specifications.
- **System Design:** This chapter mainly consists of module descriptions and algorithms with examples and unified modelling language diagrams: use case diagrams, class diagrams, sequence diagrams and activity diagrams.
- **Technology Description:** This chapter mainly consists of technology used in this project.
- **Sample code:** This chapter mainly consists of sample code for few modules.
- **Testing:** This chapter mainly consists of testing techniques and test modules.
- **Screenshots:** This chapter mainly consists of output screens of this project.
- **Conclusion:** This contains main conclusion of this project.

## 2. SYSTEM ANALYSIS

### 2.1 Existing system:

Currently, many car showrooms rely on either outdated manual systems or expensive, complex software to manage their operations. Smaller showrooms often use paper records or basic spreadsheet systems to handle inventory, employee data, and customer bookings. These methods are inefficient, prone to human error, and lack scalability. They make it difficult for showroom owners to track real-time performance, analyze sales trends, or make data-driven decisions. Additionally, managing large amounts of information through paper or spreadsheets can quickly become overwhelming as the showroom grows.

For larger showrooms or dealerships, cloud-based Dealer Management Systems (DMS) like CDK Global and Reynolds are commonly used. These systems offer a range of features, such as inventory management, customer relationship management (CRM), accounting, and reporting. While they provide comprehensive solutions, they are often expensive and require ongoing subscriptions, making them unsuitable for small or medium-sized showrooms. Furthermore, these systems require stable internet connectivity and technical expertise to set up and maintain, which can be a significant hurdle for many businesses.

Additionally, these systems typically lack offline functionality, making them less reliable in areas with poor internet access.

#### **Disadvantages:**

- Existing systems are often expensive and too complex for small showrooms, leading to difficulties in affordability and usability.
- Many solutions depend on stable internet access, which can cause disruptions in areas with unreliable connectivity.
- Manual systems increase the risk of human errors, such as incorrect data entry, which leads to inefficiencies and mistakes in inventory and bookings.

## 2.2 Proposed system:

The BLSS Cars system is a comprehensive and affordable solution designed to streamline car showroom management by integrating essential functions like employee management, inventory tracking, and customer bookings into a single platform. It ensures reliable performance with offline capabilities, making it effective even in areas with poor internet connectivity. The user-friendly interface and role-based access make it easy for showroom staff to manage daily tasks, while real-time reporting and analytics provide valuable insights for decision-making. This system is designed to be scalable, offering room for future growth and adaptation to business needs.

### Advantages:

- Cost-effective solution suitable for small and medium-sized showrooms.
- Operates offline, ensuring reliability even without internet access.
- Simple and intuitive interface, requiring minimal training for staff.
- Integrates multiple showroom functions into one platform, improving efficiency.
- Scalable, allowing the system to grow alongside the business.
- Provides real-time reports and analytics for better decision-making and performance tracking.

## 2.3 Feasibility Study:

Preliminary research examines the feasibility of the project and the potential of the system to Help the organization. The main goal of the feasibility study is to add new models and test the Technical, operational, and economic feasibility for debugging old running systems. With unlimited resources and unlimited time, any system is feasible.

The feasibility study part of the preliminary study has the following aspects:

- Technical Feasibility
- Operational Feasibility
- Economic Feasibility

**Technical Feasibility:**

This part of the study focuses on determining the technical requirements of the BLSS Cars system, ensuring it can operate efficiently within the existing infrastructure.

Key questions include:

- Do the available systems and hardware have the technical capacity to store and process the required data for the showroom management system?
- Will the proposed system be able to handle a significant number of users and provide fast responses to inquiries, no matter their location?
- Can the system be easily upgraded in the future to incorporate new features or handle growing data needs?

**Operational Feasibility:**

Operational feasibility focuses on how well the system can be integrated into the showroom's daily operations. It ensures the system supports user needs and organizational goals.

Key factors include:

- Will the system receive enough support from management and users to ensure its success?
- Can the system be used effectively after implementation, ensuring smooth daily operations for both admins and salespersons?
- Is the system designed in a way that ensures optimal use of available computer resources and enhances overall performance in managing car sales, bookings, and employee data?

**Economical Feasibility:**

Economic feasibility examines the financial viability of the BLSS Cars system, ensuring the cost of developing and maintaining the system is justified by the benefits it brings.

Key considerations include:

- Can the organization afford to develop and implement this system within its budget?
- The system is built using open-source technologies like Python and SQLite, minimizing development and ongoing operational costs.
- Only minimal expenditure is required for customized features, which ensures the system stays within budget and offers a good return on investment.

## 2.4 Functional Requirements:

Functional Requirements are the functions or features that must be in any system to satisfy the business and be acceptable to users. Based on this, the functional requirements that the system must require as follows:

- The system should have the capability to manage employee data. This includes adding, updating, and deleting employee records, along with assigning roles (e.g., admin, salesperson) and viewing employee performance.
- The system should be able to manage car inventory effectively. This includes adding new car models, updating details such as price, availability, and specifications, and removing outdated or sold cars from the system.
- The system should support the creation and management of customer bookings. Salespersons should be able to enter customer details, select car models, and finalize bookings, while admins should be able to manage and review all bookings.
- The system should provide real-time updates and notifications for car bookings. Users should receive notifications about booking confirmations, cancellations, and updates on car availability or customer status.
- The system should have a user-friendly interface to facilitate easy interaction. This may include intuitive dashboards, data visualization of car sales and booking statistics, and user-friendly navigation for both admins and salespeople.
- The system should support secure login and authentication for both admins and salespeople to ensure that only authorized personnel can access the system's features and sensitive data.



## 2.5 Non-Functional Requirements:

- **Performance** - The system should provide quick response times for all user actions to ensure a smooth and efficient user experience.
- **Scalability** - The system should be able to scale to accommodate a growing number of users, car models, and booking records without a significant impact on performance.
- **Reliability** - The system should be highly reliable, with minimal downtime. It must be available for use at all times, particularly during working hours for car showrooms.
- **Usability** - The system should have an intuitive and user-friendly interface that minimizes the learning curve for both admins and salespeople.
- **Security** - It should securely store user credentials and personal information, encrypting sensitive data both in transit and at rest.
- **Data Integrity** - The system should ensure the accuracy and consistency of data. All data related to car inventory, bookings, and employees should be kept up-to-date and accurate.

## 2.6 Requirements Specifications:

### 2.6.1 Minimum Software Requirements:

- **Domain:** Python Tkinter
- **Technologies :** Python & Sqlite3
- **Tools :** Idle & Command Prompt

### 2.6.2 Minimum Hardware Requirements:

- **Processor:** i3 or i5
- **Ram:** 8gb
- **Operating system:** Windows / Linux

### 3. SYSTEM DESIGN

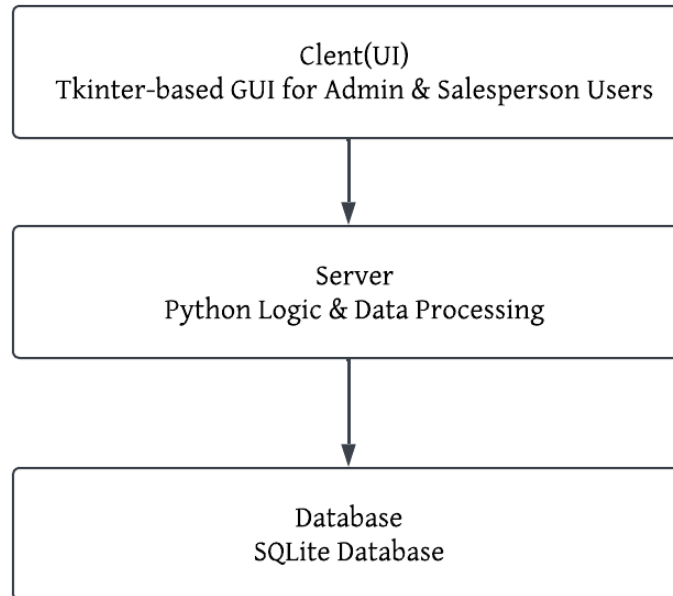
#### 3.1 Introduction:

The purpose of the design phase is to plan a solution to the problem specified in the requirements document. This phase is the first step on the path from the problem domain to the solution domain. In other words, start with what you need Design guides us how we can meet our needs System design is probably the most important factor in software quality. It will have a significant impact on later phases, especially testing and maintenance. The result of this phase is the design document. This document is similar to the solution blueprint and will be used later for implementation, testing, and maintenance.

Design activities are often divided into two separate phases: system design and detailed design System design, also known as top-level design, aims to identify the modules that need to be included in the system, the specifications of those modules, and how they interact to achieve the desired result At the end of the system design, all important data structures, file formats, output formats, and main modules and their specifications in the system are determined. The detailed design determines the internal logic of each module specified in the system design. In this phase, the data details of the module are usually specified in a high-level design description language that is independent of the target language in which the software is ultimately implemented System design focuses on the identification of modules, and detailed design focuses on the design of the logic of each module.

In other words, system design focuses on the required components, while detailed design focuses on how the components are implemented in the software The design involves identifying software components that specify the relationships between the components Providing blueprints for software structure specifications and document phase Modularity is one of the desired qualities for large systems This means that the system is divided into several parts In this way, the interaction between the parts is minimized and clearly specified During system design activities, developers bridge the gap between the requirements specifications created during requirements extraction and analysis and the systems provided to users Design is a place where quality is encouraged in development Software design is the process of translating requirements into software representations.

### 3.2 System Architecture:



#### Architecture Description:

The system architecture has following steps:

- Import necessary Python packages (Tkinter, SQLite, etc.).
- Initialize and set up the SQLite database for storing car, employee, and booking data.
- Validate user login credentials (Admin or Salesperson).
- Set up the user interface (Tkinter) based on user roles (Admin or Salesperson).
- Admin manages car inventory and employee details; Salesperson books cars for customers.
- Process car bookings, verify availability, and update database.
- Admin generates reports on sales, employees, and bookings.
- Log out the admin and end the session.

### 3.3 Modules Description:

Here's the Modules Description for the BLSS Cars system:

**3.3.1. Admin Module:**

- The Admin module is designed to manage and oversee the entire car showroom system.
- The admin has full control over the application and can perform administrative tasks such as adding new employees, updating existing employee details, and deleting employees as necessary.
- Admins can manage the car inventory, including adding new car models, updating details (like price, availability, or specifications), and removing old or sold-out cars.
- The admin has access to detailed reports that summarize sales statistics, track employee performance, and display booking details. These reports are useful for business analysis and decision-making.

**3.3.2. Salesperson Module:**

- Salespeople can book cars for customers by entering relevant information, such as the customer's details, car model selected, and the booking time. Once a booking is made, the salesperson can update or modify the booking record if necessary.
- Salesperson can book cars for customers and update booking records.
- Salespersons also have the ability to edit their personal profile (such as contact information or job title) and view their individual booking history.

**3.3.3. Authentication Module:**

- The Authentication module is responsible for ensuring secure access to the system. It handles user login by verifying usernames and passwords for both the admin and salesperson roles.
- Role-based access control is implemented to restrict functionality based on the user's role. Admins have access to all modules and features, while salespeople have limited access, only to the car booking and customer management features. This security ensures that users can only perform actions relevant to their role in the system.

**3.3.4. Database Module:**

- The Database module is critical for managing all data within the BLSS Cars system. It uses SQLite to store and organize information on cars, employees, customers, and bookings.
- The module is responsible for executing queries that retrieve necessary data, such as the list of available cars, employee details, or past booking information.

**3.3.5. Report Generation Module:**

- Provides functionality for the admin to generate various reports (e.g., car sales, employee performance, customer bookings).
- Displays the report in a user-friendly format, such as tables or charts.
- For example, reports can be generated to show how many cars were sold in a particular month, which employees have made the most sales, or which car models are most popular among customers.
- This module is essential for tracking performance metrics and evaluating the overall health of the business.

**3.3.6. User Interface Module:**

- The User Interface (UI) module is responsible for providing an intuitive and accessible interface for both admins and salespeople. Built using Tkinter, it ensures that all users can interact with the system efficiently.
- The goal of the UI is to create a seamless experience for users, allowing them to easily navigate through the system and complete their tasks. It is designed to be simple, yet powerful, with clear labels, helpful tooltips, and easy-to-understand controls.

### 3.4 UML Diagram




- The unified modelling language allows the software engineer to express an analysis model using the modelling notation that is governed by a set of syntactic semantic and pragmatic rules.
- A UML system is represented using five different views that describe the system from distinctly different perspective. UML is specifically constructed through two different domains.
- UML Analysis modelling, this focuses on the user model and structural model views of the system.
- UML design modelling, which focuses on the behavioural modelling, implementation modelling and environmental model views.
- These are divided into the following types:
  - Use case diagram.
  - Class diagram
  - Sequence diagram
  - Collaboration diagram
  - Activity diagram

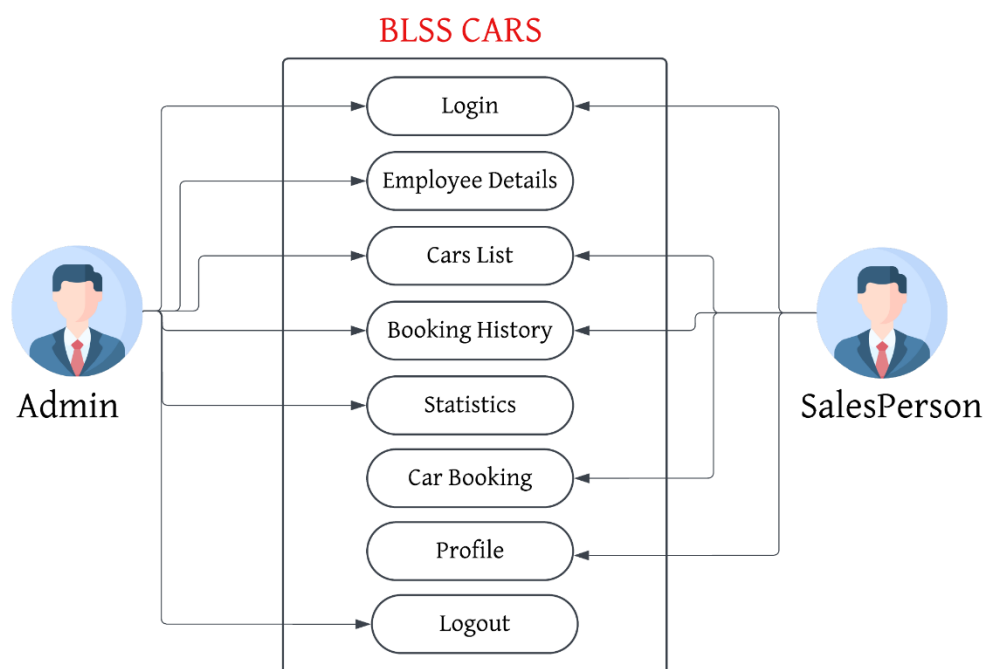
#### 3.4.1 Use case diagram

Use Case diagrams identify the functionality provided by the system (use cases), the users who interact with the system (actors), and the association between the users and the functionality. Use Cases are used in the Analysis phase of software development to articulate the high-level requirements of the system. The primary goals of Use Case diagrams include:

- Providing a high-level view of what the system does.
- Identifying the users ("actors") of the system.
- Determining areas needing human-computer interfaces.

**Graphical Notation:** The basic components of Use Case diagrams are the Actor, the Use Case, and the Association.

<b>Actor</b>	An Actor, as mentioned, is a user of the system, and is depicted using a stick figure. Written beneath the icon is the role of the user. Actors are not limited to humans. Application can also be considered an actor, if a system communicates with another application, and expects input or delivers output.	 <b>Actor Role Name</b>
	A Use Case represents a functionality of the system from the viewpoint of the user and describes the goals of their use. Use Cases are normally presented as ovals. The name of the use case is written within the ovals.	 <b>Use Case Name</b>
<b>Directed Association</b>	If an actor is involved in a use case – by starting a function of the system, for example, then this is controlled by a communication relationship, also called an association. It is identified in the use case diagram through a simple line or a line with arrow.	



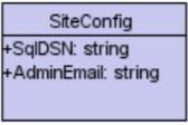
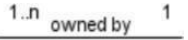
**Fig 3.4.1.1 Use Case Diagram**

### 3.4.2 Class Diagram

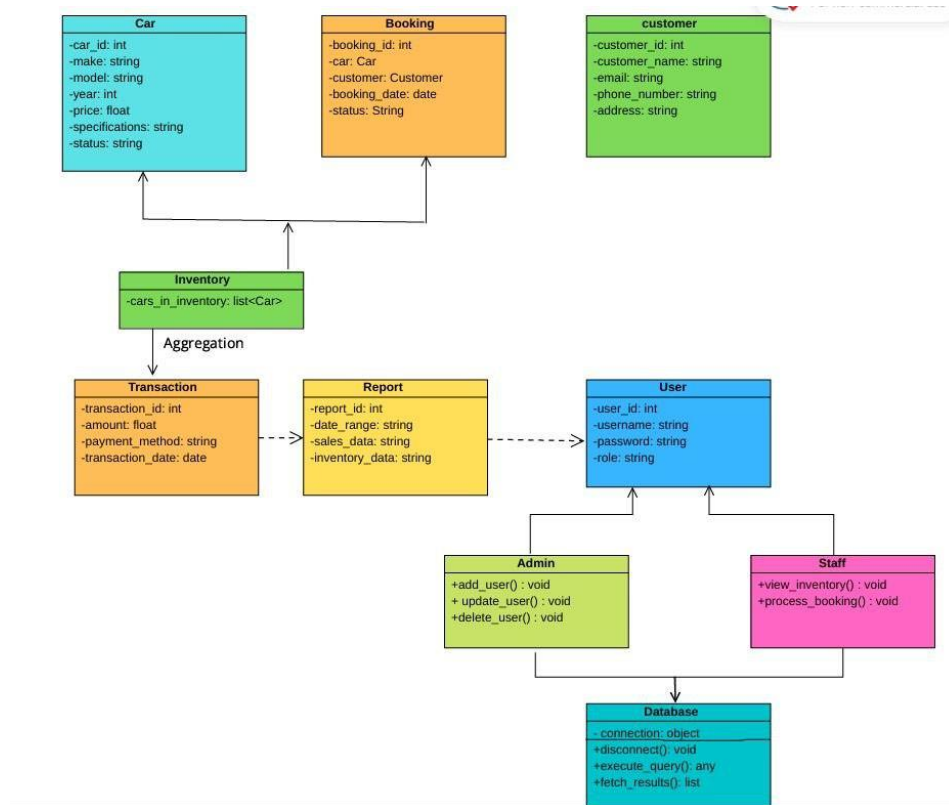
Class diagrams are one of the most commonly used diagrams in the Unified Modeling Language (UML). They help developers and stakeholders understand the static structure of an object-oriented system by focusing on the blueprint of the system.

A class represents a blueprint for objects and encapsulates the data (attributes) and behaviour (methods) that characterize those objects. In a class diagram, the emphasis is on classes, their relationships, and the overall structure of the system.

**Graphical Notation:** The elements on a Class diagram are classes and the relationships between them.

<p><b>Class</b></p>	<p>Generally, in a UML class diagram, classes are the most represented elements. The classes are usually grouped together in a class diagram which helps determine the static relations between those objects. Classes are represented with boxes containing three parts: The top part contains the name of the class. The middle part contains the attributes of the class. The bottom part contains the methods the class can execute.</p>	 <pre> classDiagram     class SiteConfig {         +SqlDSN: string         +AdminEmail: string     } </pre>
<p><b>Association</b></p>	<p>If two classes in a model need to communicate with each other, there must be a link between them. This link can be represented by an association. Associations can be represented in a class diagram by a line between these classes with an arrow indicating the navigation direction.</p>	 <pre> classDiagram     class1 "1..n" -- "1" class2 : owned by </pre>







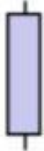
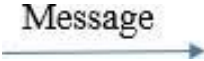
**3.4.2.1 Class Diagram for BLSS CARS**

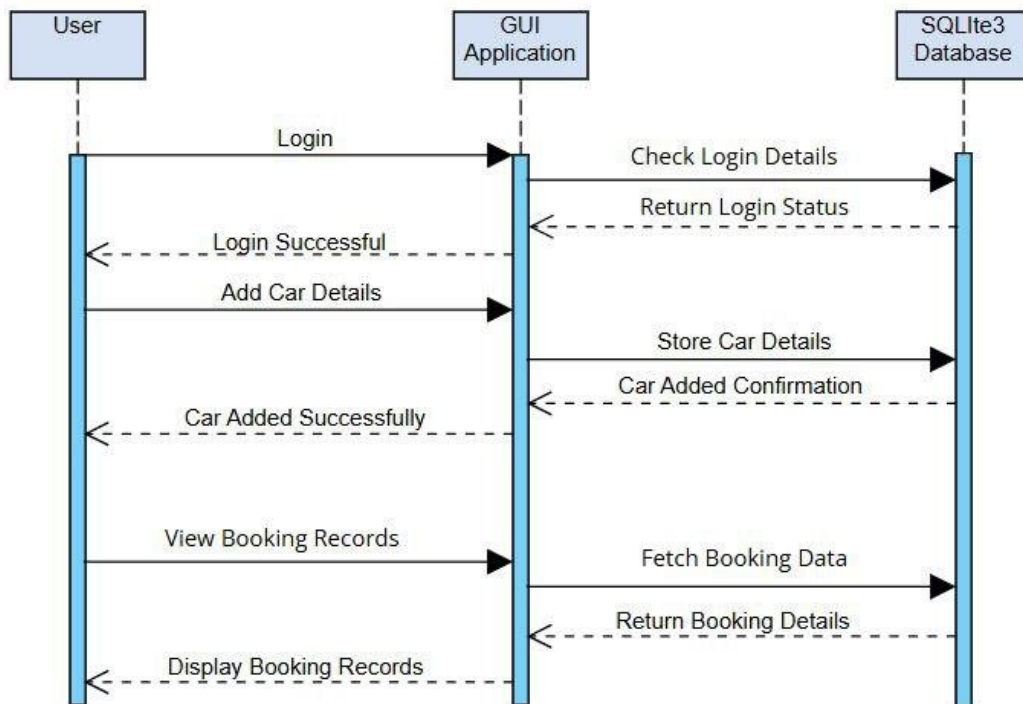
**Description:** The above diagram represents the class diagram of the project “BLSS CARS USING PYTHON TKINTER”. In the above class diagram, it describes the relation between User and Admin.

### 3.4.3 Sequence Diagram

Sequence diagrams document the interactions between classes to achieve a result, such as a use case. Because UML is designed for object-oriented programming, these communications between classes are known as messages. The Sequence diagram lists objects horizontally, and time vertically, and models these messages over time.

**Graphical Notation:** In a Sequence diagram, classes and actors are listed as columns, with vertical lifelines indicating the lifetime of the object over time.

<b>Object</b>	Each system/object instance and actor is placed on a lifeline - a vertical dotted line - going across the top of the sequence diagram. An Object is figured as a rectangular box, with the class name inside prefixed with the object name (optional) along with a semi-colon.	
<b>Lifeline</b>	The messages that pass between the lifelines are connectors - solid for an initial message or outgoing call and dotted for a return value.	
<b>Activation</b>	A rectangle shape placed on a lifeline that indicates the time or duration an object needs to finish a task.	
<b>Message</b>	Used to create an object in a sequence diagram example that is usually drawn using a dashed line and an open arrowhead pointing to the object created. It represents a symbol of a dashed line with an open arrowhead in response to the calls from the original lifeline.	




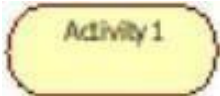


3.4.3.1 Sequence Diagram for BLSS Cars




**Description:**

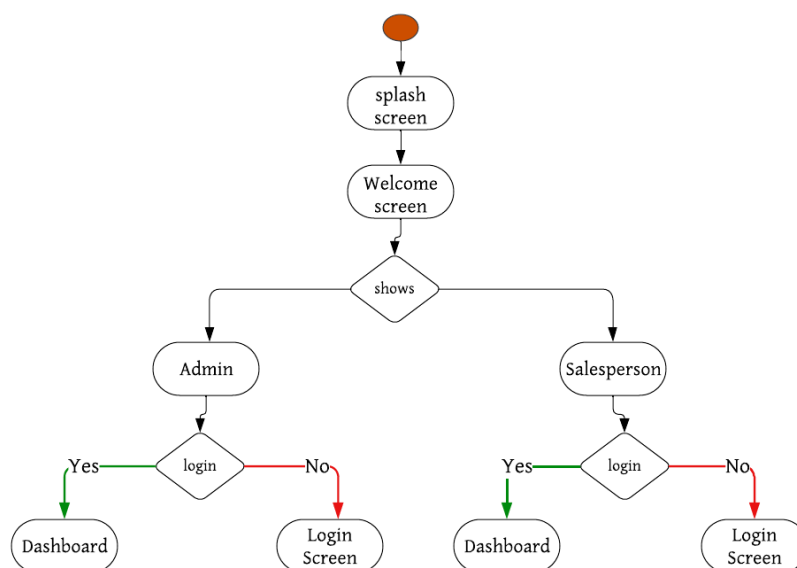
The above diagram represents the sequence diagram of the project “BLSS CARS USING PYTHON TKINTER”.

**3.4.4 Activity Diagram**

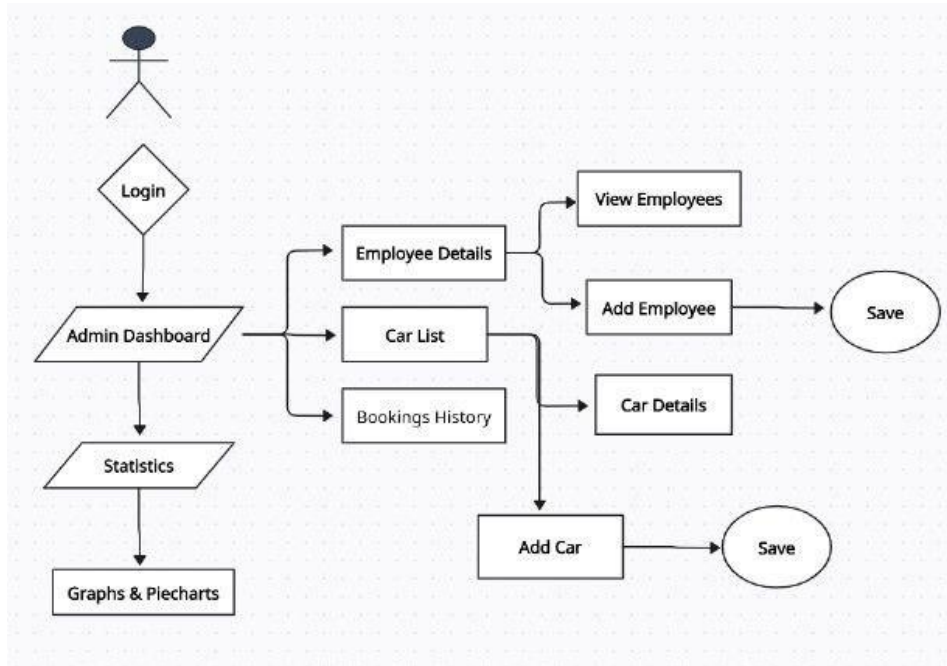
This shows the flow of events within the system. The activities that occur within a use case or within an objects behaviour typically occur in a sequence. An activity diagram is designed to be simplified look at what happens during an operation or a process. Each activity is represented by a rounded rectangle the processing within an activity goes to compilation and then an automatic transmission to the next activity occurs. An arrow represents the transition from one activity to the next. An activity diagram describes a system in terms of activities. Activities are the state that represents the execution of a set of operations. These are similar to flow chart diagram and dataflow.

<b>Starting point</b>	Represented by a fat black dot and there can be only one initial (starting) node	
<b>Action</b>	Represented by a rectangle with rounded corners (drawn in slightly different ways depending on the software used). Action nodes should have a label.	
<b>End point</b>	An open circle with a smaller, solid black dot in the middle, this is the end of the activity.	
<b>Decision</b>	Use diamonds whenever there is a choice. You can either include the decision as a question within the diamond or indicate the decision outcome on the outgoing arrows instead of simply using yes/no labels.	

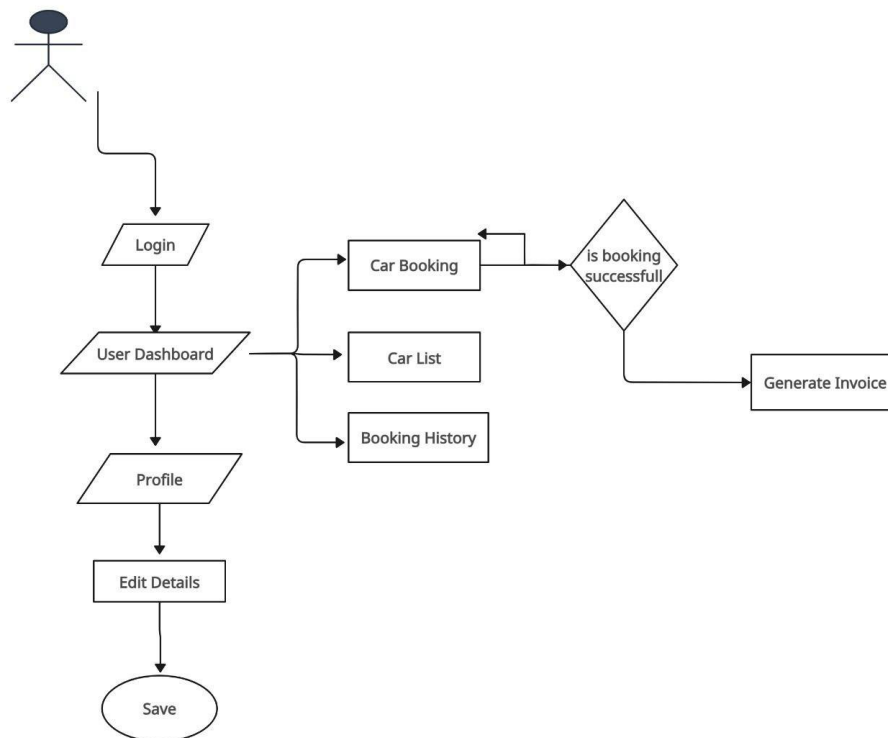
<b>Synchronization bar</b>	Multiple states are divided from the state by the fork and multiple states are merged into single states by the join.	
<b>Swim lane</b>	The business objects are divided by the swim lane into meaning full order. Actions and sub activities may be organized into swim lanes. Swim lanes are used to organize responsibility for actions and sub activities. They often correspond to organizational units in a business model.	
<b>Transition</b>	The relationship between a source state vertex and a target state vertex is directed by a transition. We use a line with an arrowhead to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.	



### 3.4.4.1 Activity Diagram for BLSS CARS



**3.4.4.2 Activity Diagram for Admin**



**3.4.4.3 Activity Diagram for User/Salesperson**

### Description

The above diagram represents the activity diagram of the project “BLSS CARS”.

## 4. TECHNOLOGY DESCRIPTION

### 4.1 Introduction to Python:

Python is a widely used, high-level general-purpose programming language. Created by Guido van Rossum in 1991 and further developed by the Python Software Foundation.

Designed with a focus on code readability, its syntax allows programmers to express concepts with fewer lines of code. Python is a dynamic, high-level, free, open source, interpreted programming language. It supports both object-oriented programming and procedural programming. Python is a programming language that allows you to work quickly and integrate your system more efficiently.

#### Features of Python:

There are many features in Python, some of which are described below:

- **Easy to code:**

Python is a widely used, high-level general-purpose programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. Designed with a focus on code readability, its syntax allows programmers to express concepts with fewer lines of code. Python is a dynamic, high-level, free, open source, interpreted programming language. It supports both object-oriented programming and procedural programming. Python is a programming language that allows you to work quickly and integrate your system more efficiently.

- **Free and Open Source:**

Python language is freely available, anyone can download it, use it and share it completely free of cost. Since it is open source, this means that source code is also available to the public.

- **Object-Oriented Language:**

Python supports object-oriented programming, allowing developers to define classes and create objects with attributes and methods. This paradigm promotes modular and reusable code, making it easier to manage and scale complex projects.

- **GUI Programming Support:**

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk(Tkinter) in python. PyQt5 is the most popular option for creating graphical apps with Python.

- **High-Level Language:**

Python is a high-level language. When programs are written in python, it is not necessary to remember the system architecture, or manage the memory.

- **Extensible feature:**

Python is an extensible language. Python code can be written into C or C++ language and compile that code in C/C++ language.

- **Python is Portable language:**

Python language is also a portable language. For example, if there is a python code for windows and in order to run this code on other platforms such as Linux, Unix, and Mac then there is no need to change it, the code can run on any platform.

- **Python is Integrated language:**

Python is also an integrated language because python can be easily integrated with other languages and technologies, then Python does have excellent integration capabilities. Python provides various mechanisms to interact with other languages, such as C/C++, Java, and .NET.

- **Interpreted Language:**

Python is an interpreted language, which means that code is executed line by line without the need for compilation. This enables quick prototyping and interactive experimentation, as Python's interactive shell provides immediate feedback.

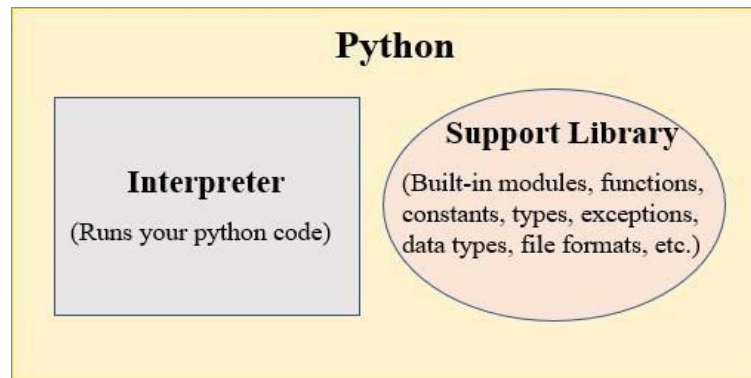
- **Dynamically Typed Language:**

Python is a dynamically typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature there is no need to specify the type of variable.

## How Python runs?

When the Python software is installed on your machine, minimally, it has:

- an interpreter
- a support library



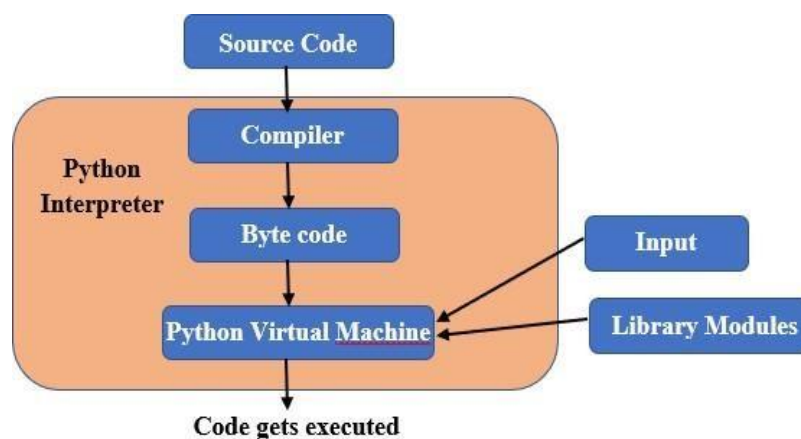
## The Interpreter

- Interpreter is nothing but a software which can run your Python scripts.
- Interestingly, it can be implemented in any programming language!
- C Python is the default interpreter for Python which is written in C programming language.
- python is another popular implementation of python interpreter written using java programming language.

## Python's view of Interpreter:

The Python source code goes through the following to generate an executable code:

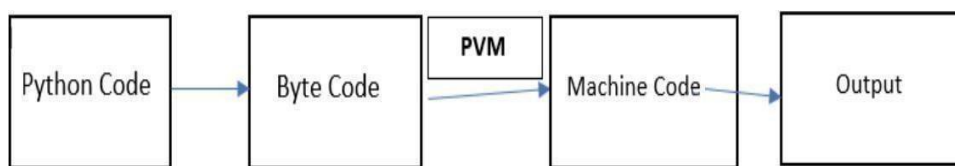
- Python source code or instruction will be read by the python compiler. Then it verifies that the instruction is well-formatted, that is, each line of the syntax will be checked. If it identifies an error, it immediately pauses the translation and shows an error message.
- If error is not found, i.e., if the python instruction or source code is well-formatted then it translates into its equivalent form in an intermediate language known as the "Byte code" by the compiler.
- Byte code is then sent to the Python Virtual Machine (PVM) which is the python interpreter. Now it converted to the python byte code into machine-executable code by the PVM. The conversion is halted with an error message while an error occurs during this interpretation.





## Python Virtual Machine

Python uses interchangeable code modules instead of a single long list of statements that was standard in functional programming languages. The default implementation of Python is called "C, python". This is a widely used implementation of the Python standard. Python does not translate that code into machine code. This is what the hardware understands. It actually translates it into what is called bytecode. That is, there is compilation inside Python, but it's not machine language. This is a bytecode (.pyc or .pyo) and this bytecode is not understood by the CPU. Therefore, to execute bytecode, you need an interpreter called a Python virtual machine.



The Python source code goes through the following to generate an executable code:

**Step 1:** The python compiler reads a python source code or instruction. Then it verifies that the instruction is well-formatted, i.e., it checks the syntax of each line. If it encounters an error, it immediately halts the translation and shows an error message.

**Step 2:** If there is no error, i.e., if the python instruction or source code is well-formatted then the compiler translates it into its equivalent form in an intermediate language called "Byte code".

**Step 3:** Byte code is then sent to the Python Virtual Machine (PVM) which is the python interpreter. PVM converts the python byte code into machine-executable code. If an error occurs during this interpretation, then the conversions halted with an error message.

## 4.2 Applications of Python:

**Web Applications:** You can create scalable Web Apps using frameworks and CMS (Content Management System) that are built on Python. Some of the popular platforms for creating Web Apps are Django, Flask, Pyramid, Plone, Django CMS. Sites like Mozilla, Reddit, Instagram and PBS are written in Python.

**Scientific and numeric computing:** There are numerous libraries available in Python for scientific and numeric computing. There are libraries like: SciPy and NumPy that are used in general purpose computing. And there are specific libraries like: EarthPy for earth science, AstroPy for Astronomy and so on. Also, the language is heavily used in machine learning, data mining and deep learning.

**Creating software prototypes:** Python is slow compared to compiled languages like C++ and Java. It might not be a good choice if resources are limited, and efficiency is a must. However, Python is a great language for creating prototypes.

**GUI based desktop applications:** Python has simple syntax, modular architecture; rich text processing tools and the ability to work on multiple operating systems create highly functional Graphical User Interface (GUI).

**Data Analysis and Visualization:** Data analysis is a process of collecting, inspecting, cleansing, transforming, and modeling data to discover useful information, make predictions, arrive at conclusions, support decision-making processes, and more.

**Image Processing and Graphic Design Applications:** Python has been used to make 2D imaging software such as Inks cape, GIMP, Paint Shop Pro and Scribes'. Further, 3D animation packages, like Blender, 3dsMax, Cinema 4D, Houdini, Light wave, and Maya, also use Python in variable proportions.

**Artificial Intelligence and Machine Learning:** AI/ML applications require a language that is stable, secure, flexible, and is equipped with tools that can handle the various unique requirements of such projects. Python is widely used in machine learning and AI. Libraries like TensorFlow, Keras, and PyTorch provide efficient implementations of deep learning algorithms. Python's simplicity and readability make it an excellent choice for prototyping and experimenting with machine learning models.

**Scientific and Computational Applications:** The higher speeds, productivity, and availability of tools, such as Scientific Python and Numeric Python, have resulted in Python becoming an integral part of applications involved in computation and processing of scientific data. 3D modeling software, such as Free CAD, and finite element method software, such as Abacus, are coded in Python.

**Games:** Python has various modules, libraries and platforms that support development of games. For example, PySoy is a 3D game engine supporting Python 3, and PyGame provides functionality and a library for game development. There have been numerous games built using Python including Civilization-IV, Disney's Toon town Online, and Vega Strike etc.

**Operating Systems:** Python is often an integral part of Linux distributions. For instance, Ubuntu's Ubiquity Installer, and Fedora's and Red Hat Enterprise Linux's Anaconda Installer are written in Python. Gentoo Linux makes use of Python for Portage, its package management system.

## 4.3 Technologies used:

### Tkinter:

Tkinter is Python's built-in library for creating Graphical User Interfaces (GUIs). It is based on the Tcl/Tk GUI framework and is widely used for developing desktop applications because of its simplicity, ease of use, and platform independence.

## Key Features of Tkinter:

- **Comprehensive Widget Library** - Tkinter offers a rich set of widgets, including buttons, labels, entry fields, text areas, menus, and canvas widgets, allowing developers to create interactive applications with ease.
- **Cross-Platform Compatibility** - Tkinter applications run seamlessly on Windows, macOS, and Linux, ensuring consistency and portability.
- **Event-Driven Programming** - User interactions like clicks, key presses, and window events are efficiently handled through Tkinter's event-driven architecture.
- **Built-in Layout Managers** - Tkinter provides powerful layout managers—pack, grid, and place—to arrange widgets dynamically or position them absolutely.
- **Customization and Styling** - Widgets can be styled using fonts, colors, and sizes, with advanced themes supported via ttk (Themed Tkinter).

## Advantages of Tkinter:

- **Beginner-Friendly**: Simple syntax and no external installation requirements make it accessible for newcomers to GUI development.
- **Rapid Prototyping**: Ideal for quickly creating small to medium-scale desktop applications.
- **Rich Community Support**: Extensive online documentation, tutorials, and forums are available for problem-solving and learning.
- **Versatility**: Suitable for a variety of applications, such as calculators, text editors, games, and educational tools.

## SQLite3:

SQLite3 is a lightweight, self-contained, serverless, and high-performance relational database management system (RDBMS). It is embedded within the Python standard library, which means no external installation is required to use it. SQLite3 is ideal for applications that need a simple database solution without the complexity of a full-fledged database server.

**Key Features of SQLite3:**

- **Serverless Architecture** - Unlike traditional databases, SQLite is serverless, meaning it does not require a separate server process to operate. All data is stored in a single file, making it highly portable.
- **Lightweight and Fast** - SQLite is small in size (less than 1MB) and optimized for efficiency, making it ideal for lightweight applications and embedded systems.
- **ACID Compliance** - SQLite3 adheres to the principles of Atomicity, Consistency, Isolation, and Durability, ensuring reliable transactions and data integrity.
- **Cross-Platform** - SQLite databases can be created and accessed on multiple platforms, such as Windows, macOS, Linux, and even mobile operating systems.
- **Self-Contained** - The SQLite engine is self-contained and requires no external dependencies.

**Basic SQLite3 Workflow**

- **Connecting to a Database** - Create a connection to the database file. If the file doesn't exist, SQLite will create it automatically.
- **Creating a Cursor** - Use a cursor object to execute SQL queries.
- **Executing SQL Queries** - Perform operations such as creating tables, inserting data, updating records, and retrieving information using SQL syntax.
- **Committing Changes** - Save changes to the database using the `commit()` method.
- **Closing the Connection** - Always close the connection after completing database operations.

## 5. SAMPLE CODE

### Opening screen:

```
import tkinter as tk

from tkinter import ttk

from PIL import Image, ImageTk

from main_first import main_screen

def center_window(window, width, height):

    screen_width = window.winfo_screenwidth()

    screen_height = window.winfo_screenheight()

    x = (screen_width - width) // 2

    y = (screen_height - height) // 2

    window.geometry(f'{width}x{height}+{x}+{y}')

def open_main_screen():

    splash.withdraw()

    main_screen()

splash = tk.Tk()

splash.title('Splash Screen')

center_window(splash, 650, 500)

splash.overrideredirect(True)

image = Image.open("logo.jpeg")

resized_image = image.resize((650, 500), Image.LANCZOS)

photo_image = ImageTk.PhotoImage(resized_image)

image_label = tk.Label(splash, image=photo_image, bd=0)

image_label.place(x=0, y=0)

progress = ttk.Progressbar(splash, orient="horizontal",

                           mode="determinate", length=650)
```

```

progress.place(x=0, y=480)

def load_progress():
    for i in range(101):
        progress['value'] = i
        splash.update_idletasks()
        splash.after(15)
    open_main_screen()
splash.after(100, load_progress)
splash.mainloop()

```

### **Admin Dashboard:**

```

import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk
from open_employee import open_new_window
from statistics import display_statistics
from show_bookings import display_bookings
from admin_cars_list import admin_car_showroom
def show_admin_dashboard():
    def center_window(window, width, height):
        screen_width = window.winfo_screenwidth()
        screen_height = window.winfo_screenheight()
        x = (screen_width - width) // 2
        y = (screen_height - height) // 2
        window.geometry(f'{width}x{height}+{x}+{y}')
    def create_emp():
        open_new_window()
    def car_list():
        admin_car_showroom()
    def booked_details():
        display_bookings()

```

```

def statistics():
    display_statistics()
root = tk.Toplevel()
root.title("Admin Dashboard")
center_window(root, 900, 600)
root.configure(bg="white")
style = ttk.Style()
style.theme_use("clam")
card_width, card_height = 190, 190
style.configure("Card.TFrame", background="#2B579A", relief="flat")
style.configure("CardText.TLabel", font=("Times New Roman", 16, "bold"),
foreground="white", background="white")
label_welcome = ttk.Label(root, text="Welcome Admin", font=("times new roman", 30,
"bold"), foreground="black", background="white")
label_welcome.pack(pady=(40, 20))
def create_card(parent, text, image_path, command):
    canvas = tk.Canvas(parent, width=card_width, height=card_height, bg="#2B579A",
highlightthickness=0, borderwidth=0)
    canvas.pack_propagate(False)
    card = ttk.Frame(canvas, style="Card.TFrame")
    card.place(relx=0.5, rely=0.5, anchor="center")
    try:
        icon = Image.open(image_path).resize((130, 120), Image.LANCZOS)
        icon_img = ImageTk.PhotoImage(icon)
        label_icon = tk.Label(card, image=icon_img, background="#2B579A",
borderwidth=0, highlightthickness=0, relief="flat")
        label_icon.image = icon_img
        label_icon.pack(pady=(5, 0))
    except FileNotFoundError:
        print(f"Image '{image_path}' not found")
    label_text = ttk.Label(card, text=text, style="CardText.TLabel",
background="#2B579A")
    label_text.pack(side="bottom", pady=(10, 5))
    canvas.bind("<Button-1>", lambda e: command())
    card.bind("<Button-1>", lambda e: command())

```

```

label_icon.bind("<Button-1>", lambda e: command())
label_text.bind("<Button-1>", lambda e: command())

return canvas

card_data = [
    ("Employee details", "employee_icon.png", create_emp),
    ("Cars List", "car_list_icon.png", car_list),
    ("Booked Details", "booked_icon.png", booked_details),
    ("Statistics", "statistics_icon.png", statistics),
]

frame_cards = tk.Frame(root, bg="white")
frame_cards.pack()

for i, (text, image, cmd) in enumerate(card_data):
    row, col = divmod(i, 2)
    card = create_card(frame_cards, text, image, cmd)
    card.grid(row=row, column=col, padx=10, pady=10)

def logout():
    root.destroy()

    from main_first import main_screen
    main_screen()

style.configure("LogoutButton.TButton",
                font=("times new roman", 18, "bold"),
                foreground="white",
                background="red",
                padding=(10, 5),
                relief="flat",
                width=20)

logout_button = ttk.Button(root, text="Logout", command=logout,
style="LogoutButton.TButton")

logout_button.pack(pady=(15, 10))

root.mainloop()

```



## 6. TESTING

### 6.1 Introduction

In general, software engineers distinguish between software bugs and software failures. If an error occurs, the software does not do what the user expects. Bugs are programming errors that may or may not actually appear as errors. The error can also be described as an error in the semantic accuracy of the computer program. If the exact calculation conditions are met, the error will be an error. One is that the faulty part of the computer software is running on the CPU. Bugs can also fail if the software is ported to another hardware platform or compiler, or if the software is extended. Software testing is a technical survey of the product under test to provide stakeholders with quality-related information.

### 6.2 System Testing and Implementation

System testing is the process of evaluating a software system's functionality and performance in a controlled environment. It involves testing the entire system as a whole, including its integration with external components or subsystems. The main objectives of system testing are to verify that the system meets the specified requirements, identify any defects or inconsistencies, and ensure the system operates correctly in different scenarios.

The purpose is to run different parts of the module code to detect coding errors. The modules are then sequentially integrated into the subsystem, which in itself integrates and finally forms the entire system. Integration tests are run during module integration. The goal is to identify design flaws while focusing on the connections between the modules. After assembling the system, system tests are run. Here, the system is tested against system requirements to ensure that all requirements are met, and that the system is functioning according to the requirements. Finally, an acceptance test is run to show the customer how the system works.

Choosing the right test case is essential for successful testing. There are two different approaches to choosing a test case. The software or module under test is treated as a black box and the test case is determined based on the specifications of the system or module.

For this reason, this form of testing is also known as a "black- box test. "The focus here is to test the external behaviour of the system. Structural testing determines test cases based on the logic of the module under test. The general approach here is to achieve some kind of statement coverage in your code. The two test forms complement each other. One tests the external behaviour and the other tests the internal structure. Testing is a very important and time-consuming task. This requires good planning for the entire testing process. The testing process often begins with a test plan. This plan identifies, schedules,

allocates resources, and establishes test guidelines for all test-related activities that need to be performed.

The test plan specifies the conditions that need to be tested. How to integrate modules with different units to test. Next, a test case specification is created for the various test units, listing all the different test cases, along with the expected results to be used for testing. When you test the unit, the specified test case is run, and the actual result is compared to the expected output. The final result of the test phase is a test report and a bug report, or a series of such reports. Each test report contains a set of test cases and the results of running code using the test cases. The error report describes the error that occurred, and the actions taken to correct the error.

### 6.3 Testing techniques

Testing is a process which reveals errors in the program. It is the major quality measure employed during software development. During testing, the program is executed with a set of conditions known as test cases and the output is evaluated to determine whether the program is performing as expected. In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at differing phases of software development are:

#### **Black box testing**

Black box testing is a software testing technique in which the internal structure, design, or implementation details of the system being tested are not known to the tester. The testing is conducted based on the system's specifications, requirements, and expected behavior. The primary focus of black box testing is to assess the system's functionality, inputs, and outputs without considering its internal workings.

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. This testing has been used to find errors in the following categories:

- Incorrect or missing functions.
- Interface errors.
- Errors in data structure or external database access.
- Performance errors.
- Initialization and termination errors.

## **White box testing**

White box testing, also known as clear box testing or structural testing, is a software testing technique that examines the internal structure, design, and implementation details of the system being tested. It helps to identify defects related to coding errors, missing or incomplete logic, and integration issues. By evaluating the system's internal structure, white box testing can improve code quality, identify performance bottlenecks, and ensure that all code paths are tested. In this testing, the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases. It has been used to generate the test cases in the following cases:

- Guarantee that all independent paths have been executed.
- Execute all logical decisions on their true and false sides.
- Execute all loops at their boundaries and within their operational.
- Execute internal data structures to ensure their validity.

## **6.4 Testing Strategies**

### **Unit testing**

Unit testing is a software testing method that focuses on testing individual units, or the smallest testable components, of a software application. The goal of unit testing is to verify that each unit of code, such as functions, methods, or classes, behaves as expected and produces the correct output for a given input. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration.

### **Integration testing**

Integration testing is a software testing technique that focuses on testing the interaction between different components or modules of a software application. It aims to uncover defects that may arise due to the integration or communication between these components.

These are designed to test integrated software components to determine if they actually run as one program.

## **System testing**

System testing ensures that the entire integrated software system meets the requirements. It tests a configuration to ensure known and predictable results. It verifies that all components, subsystems, and interfaces function correctly when integrated and interact with each other as expected. The primary goal of system testing is to ensure that the software meets the specified requirements and performs as intended in the target environment.

## **Functional testing**

It is designed to test the run-time performance of software within the context of an integrated system. Used to test the speed and effectiveness of the program. It is also called load testing. We check, what is the performance of the system in the given load.

Functional testing is centered on the following items:

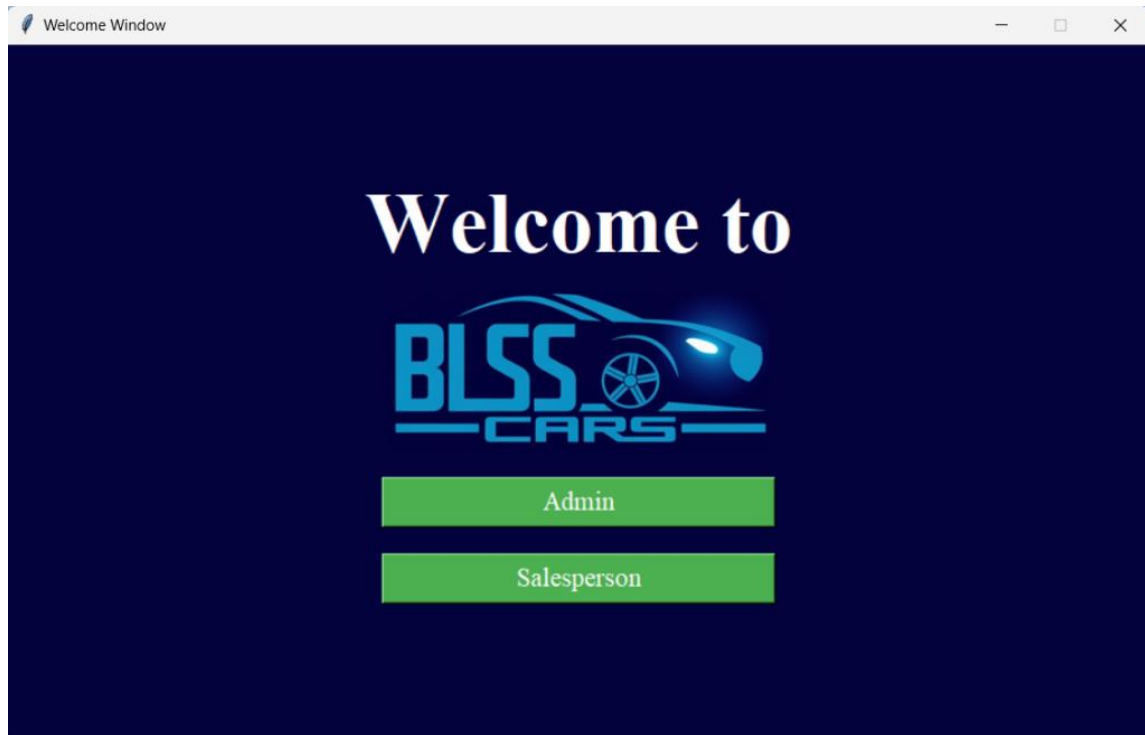
- Valid Input: identified valid input classes ought to be considered.
- Invalid Input: identified invalid input classes ought to be ignored.
- Functions: functions must be exercised which are identified.
- Output: identified classes of application outputs should be exercised
- Systems/Procedures: must invoke interfacing systems or procedures.

## 7. SCREENSHOTS



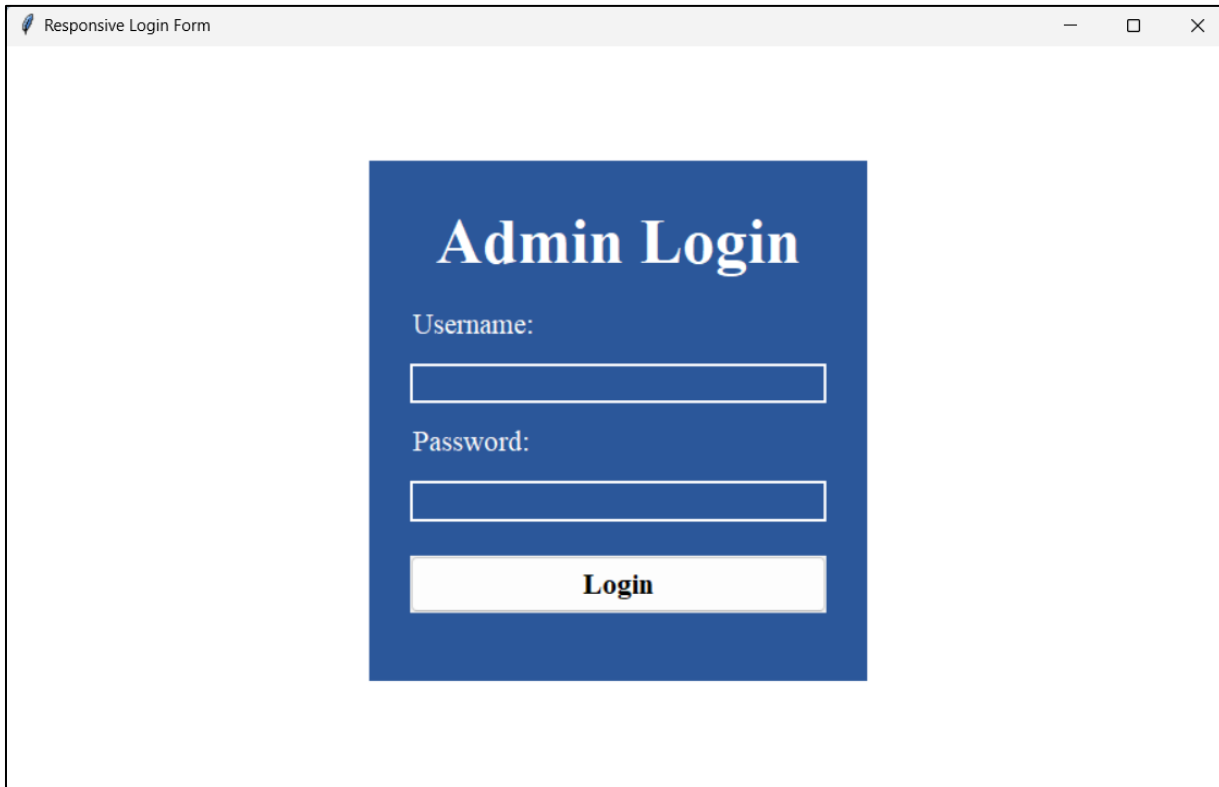
**Fig 7.1 Screenshot for splash screen**

**Description:** The above screenshot displays the Splash Screen to open the application



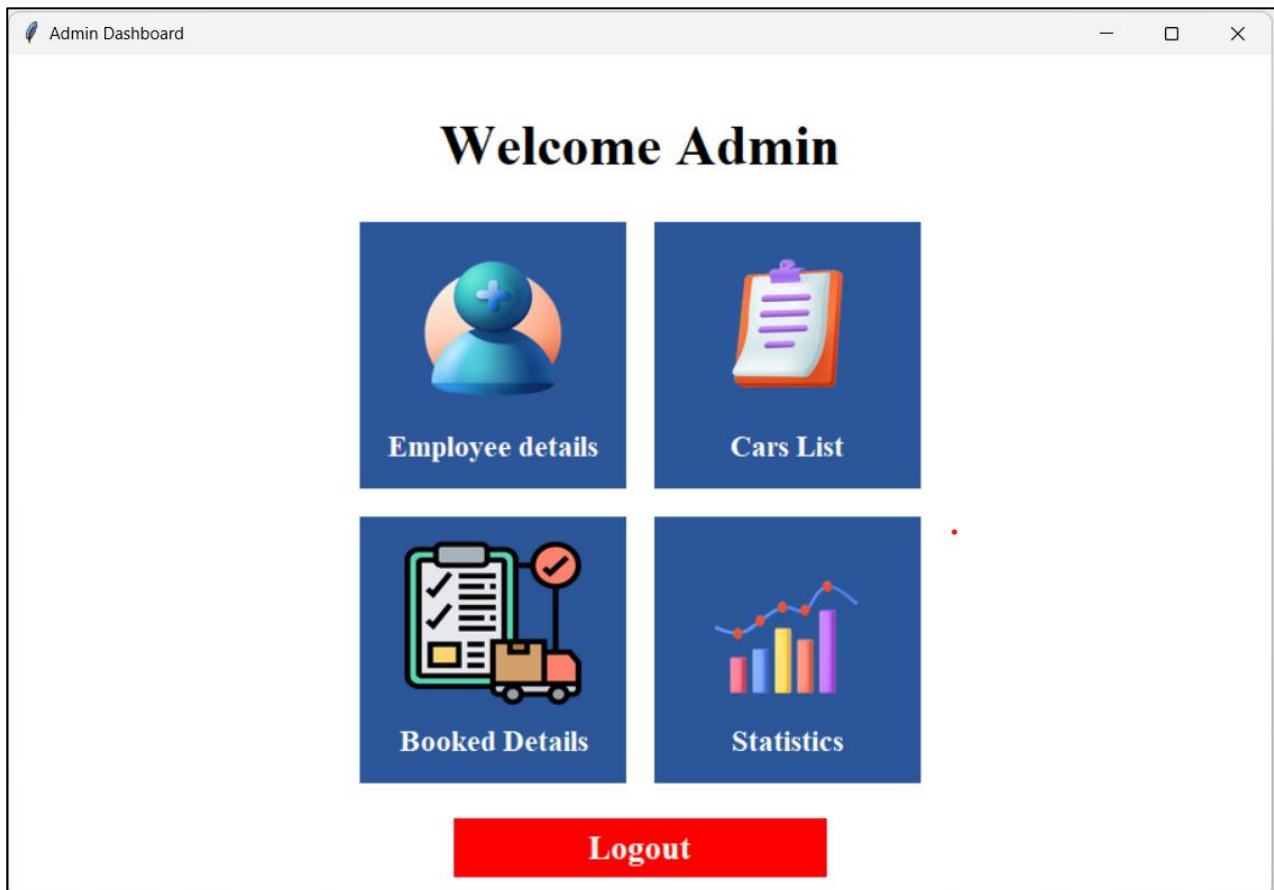
**Fig 7.2 Screenshot for Welcome Screen of the Application**

**Description:** The above screenshot displays the Welcome Screen to select whether an admin or Salesperson



**Fig 7.3 Screenshot for Login**

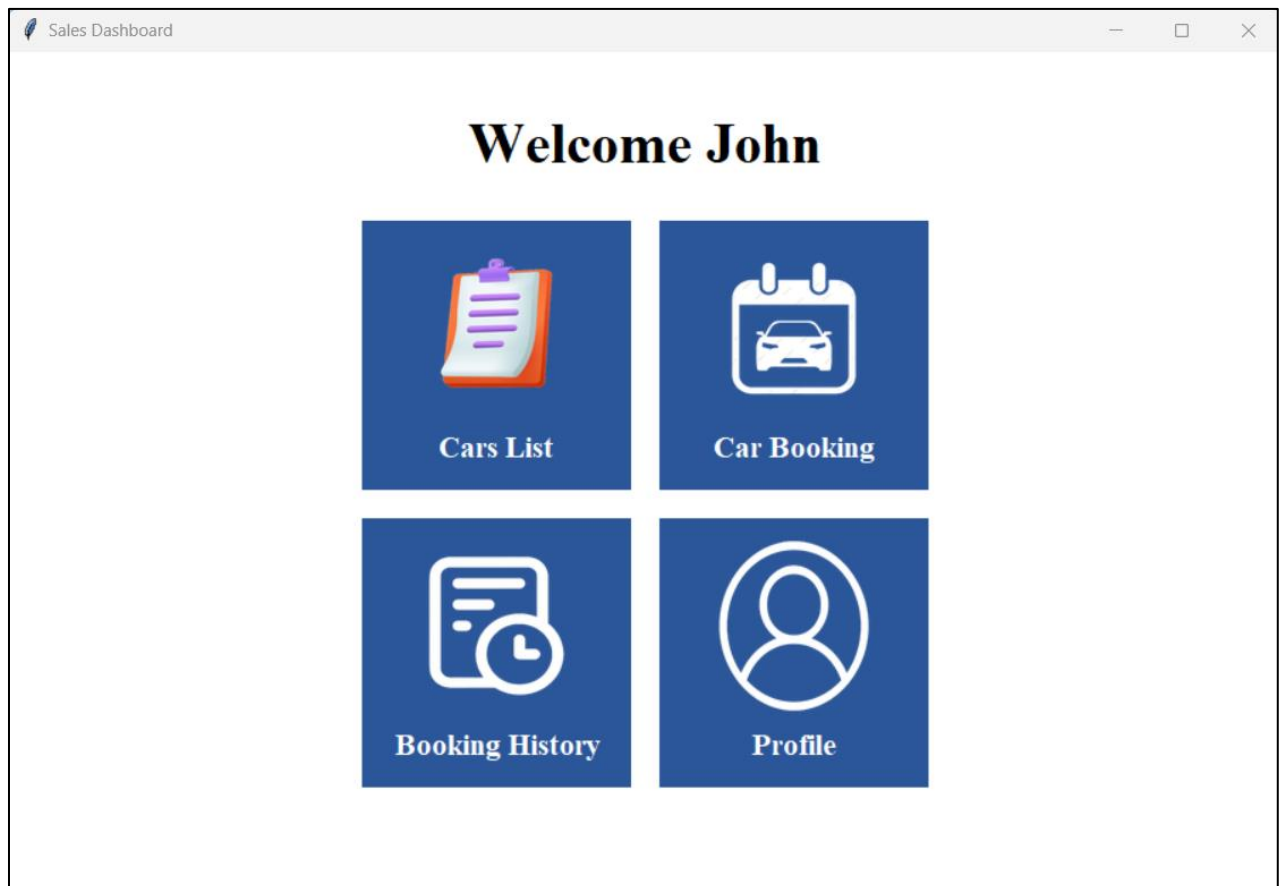
**Description:** The above screenshot displays the Admin Login where the admin enters username and Password to access the dashboard



**Fig 7.4 Screenshot for admin dashboard**

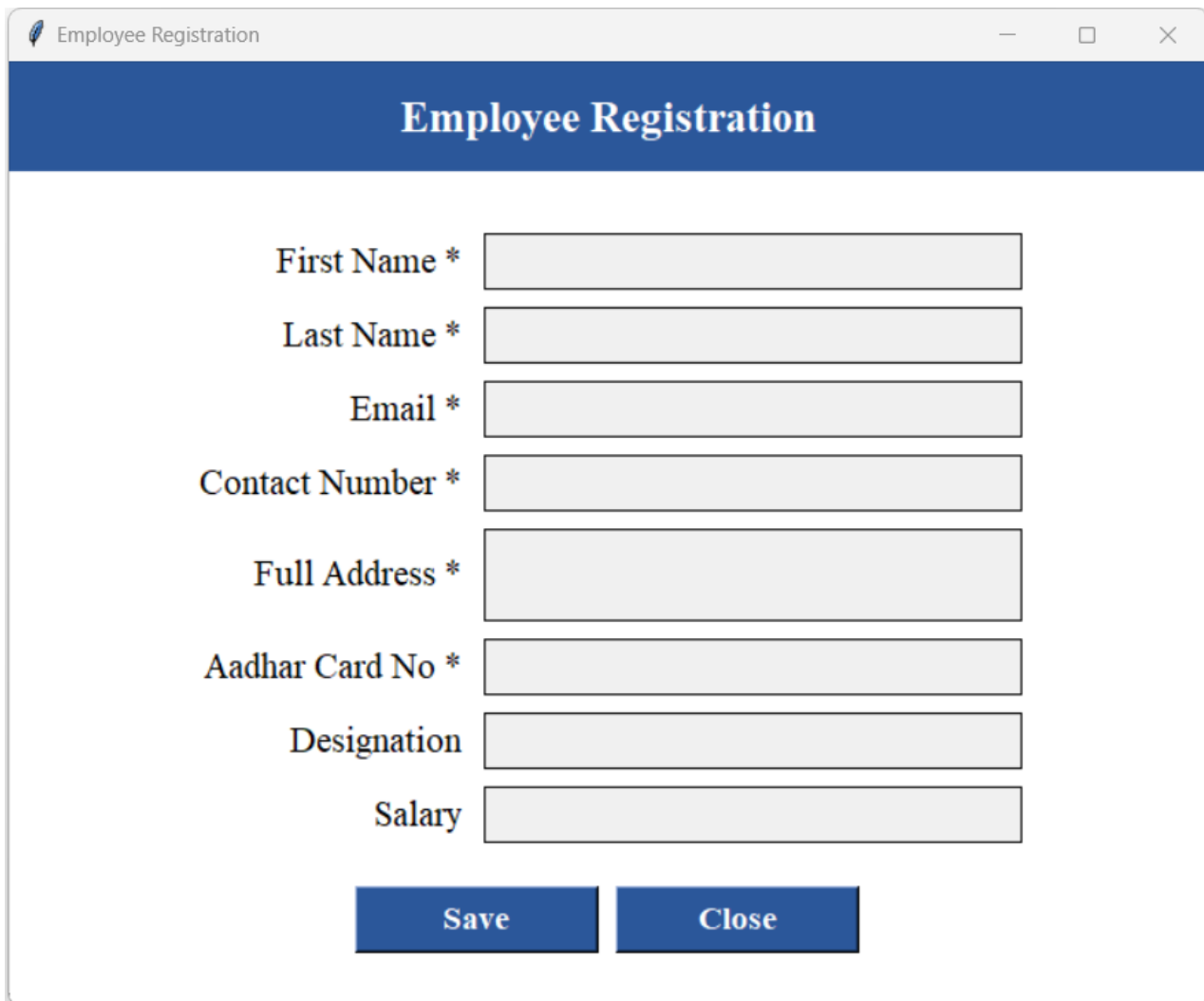
**Description:** The above screenshot displays Admin Dashboard for Admin to see employee details, cars list , booking details and statistics.





**Fig 7.5 Screenshot for salesperson dashboard**

**Description:** The above screenshot displays the salesperson dashboard

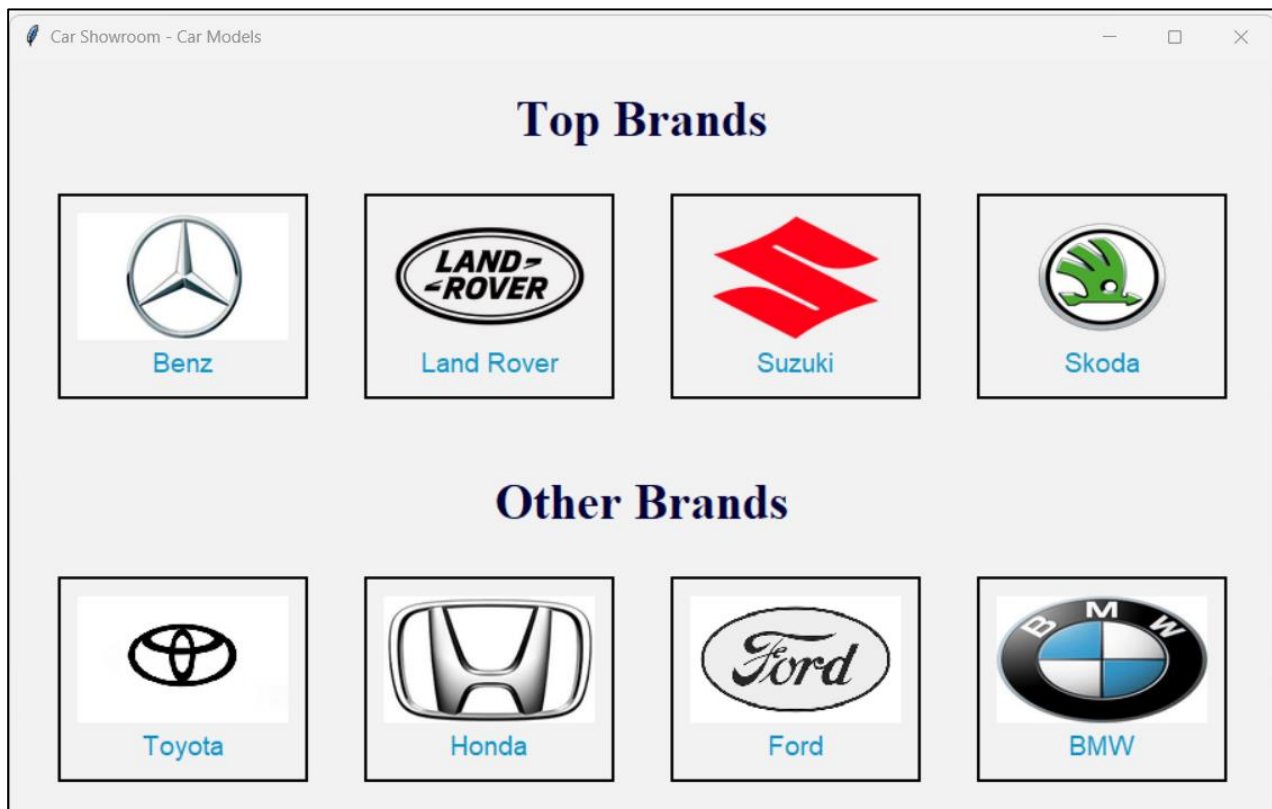


The screenshot shows a Tkinter window titled "Employee Registration". The window has a blue header bar with the title "Employee Registration" in white. Below the header, the form contains several input fields with labels and asterisks indicating required fields: "First Name \*", "Last Name \*", "Email \*", "Contact Number \*", "Full Address \*", "Aadhar Card No \*", "Designation", and "Salary". Each label is followed by a light gray rectangular input box. At the bottom of the form, there are two blue buttons with white text: "Save" and "Close".

Field	Label	Required
First Name	First Name *	Yes
Last Name	Last Name *	Yes
Email	Email *	Yes
Contact Number	Contact Number *	Yes
Full Address	Full Address *	Yes
Aadhar Card No	Aadhar Card No *	Yes
Designation	Designation	No
Salary	Salary	No

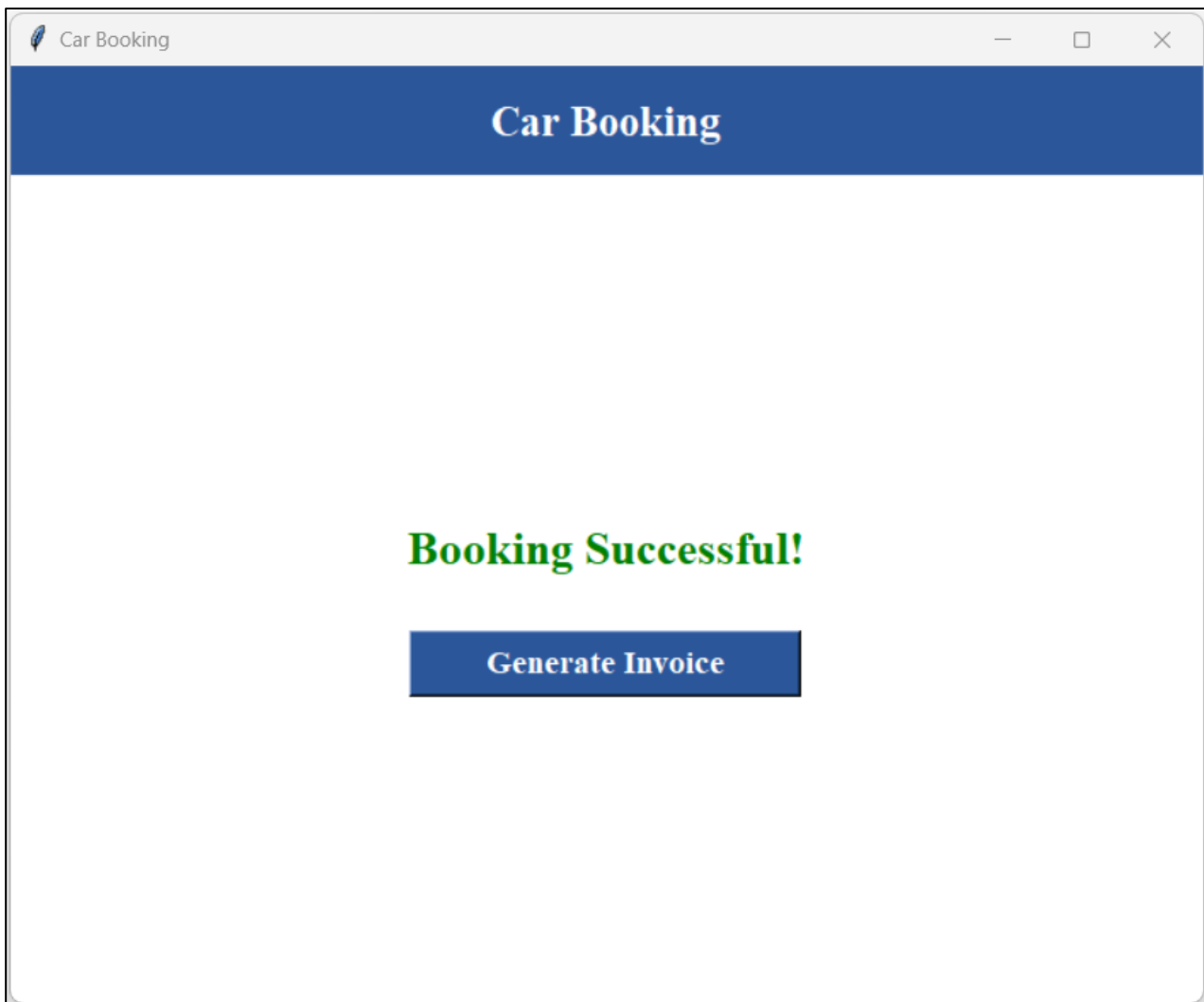
**Fig: 7.6 Screenshot about employee Registration**

**Description:** The above screenshot displays the registration for adding new employees



**Fig: 7.7 Screenshot about Car Models**

**Description:** The above screenshot displays car models available in the showroom



**Fig: 7.8 Screenshot about Car booking**

**Description:** The above Screenshot displays the Booking Successful screen

## 8. CONCLUSION

The BLSS Cars project stands as a comprehensive solution for car showroom management, addressing key challenges with modern technology. This desktop application efficiently integrates modules for administrators and salespersons, streamlining tasks such as managing car inventory, employee details, and customer bookings. By utilizing SQLite for reliable data storage and Python with Tkinter for an intuitive user interface, the system ensures a seamless and secure experience.

The project enhances productivity by empowering administrators to generate detailed reports and manage operations effectively, while salespersons can efficiently handle customer interactions and bookings. With features like role-based authentication, real-time updates, and a user-friendly design, BLSS Cars bridges the gap between traditional showroom practices and technological advancements.

In conclusion, BLSS Cars is more than just a management tool—it's a step forward in revolutionizing car showroom operations. Its scalability, efficiency, and innovative design make it a valuable asset for the automotive industry, offering the potential for future enhancements like online integration and advanced analytics. This project exemplifies how technology can transform businesses into smarter, more efficient enterprises.

### 8.1 Future Scope:

- Enable support for multiple branches within the same application, where each branch operates independently but can merge data offline using export/import functionalities.
- Add multilingual support to cater to showrooms in different regions, making it more accessible.
- Add functionality to scan barcodes or QR codes for quick retrieval of car or customer details, enhancing the efficiency of both admins and salespersons.
- Introduce advanced inventory tracking, such as notifications for low stock or upcoming car models.
- Integrate with popular payment gateways like Razorpay, Stripe, or PayPal to enable direct booking payments by customers.