

Introduction

Project Title : Garage Management System

College name : Ideal Institute Of Technology, Kakinada.

Team ID : LTVIP2025TMID28899

Team size : 2

Team Leader : A.Sony

Roll no. : 236K5A0407

Branch : Electronics & Communicatin Engineering

Mail ID : amaladasusony322005@gmail.com

Team Member : K.Prashanthi

Roll no. : 216K1A0428

Branch : Electronics & Communicatin Engineering

Mail ID : kolaprashanthi1@gmail.com

#Garage Management System#

Overview:

A **Garage Management System (GMS)** is a specialized software solution designed to streamline and automate the daily operations of automotive repair shops and service centers. It enhances efficiency, reduces manual errors, and improves customer satisfaction by integrating various functionalities into a unified platform.

Key Features of a Garage Management System

Appointment Scheduling & Online Booking

Enables customers to book, reschedule, or cancel appointments online.

Sends automated SMS/email confirmations and reminders to reduce no-shows.

Job Card Creation & Workflow Automation

Automatically generates job cards with customer and vehicle details.

Assigns tasks to mechanics with estimated timelines, ensuring efficient workflow management.

Vehicle History Tracking

Maintains comprehensive service records for each vehicle.

Facilitates better diagnostics and upselling opportunities based on past services.

Inventory and Spare Parts Management

Tracks stock levels and automates reordering of parts.

Links spare part usage with individual jobs to monitor inventory effectively.

Implementing a robust Garage Management System can significantly transform the operations of automotive service centers, leading to increased efficiency, customer satisfaction, and business growth.

The Garage Management System (GMS) is a custom application built on the **Salesforce Platform (Lightning or Classic)** to streamline operations in an automobile service garage. It leverages Salesforce CRM capabilities, automation tools, and Apex development to provide a cloud-based solution for customer interaction and garage workflow management.

Core Features

1. Customer Management

- Track customer details using **Leads** and **Accounts**
- Store vehicle ownership info using **Custom Objects**
- Manage communication via **Chatter** or **Email-to-Case**

2. Vehicle Management

- Custom Object: `Vehicle__c`
 - Fields: Make, Model, Year, VIN, Registration Number, Owner (Lookup to Account)

3. Service Booking

- Custom Object: `Service_Appointment__c`
 - Related to: `Vehicle__c`, Account
 - Tracks appointment date, services requested, mechanic assigned

4. Job Card and Invoice Management

- Create a `Job_Card__c` object with status tracking (Open, In Progress, Completed)
- Link services used and parts installed (via related lists or junction objects)
- Auto-generate invoices using **Flow** or **Apex Triggers**

5. Inventory Management (Optional)

- Track parts used for services

- Monitor stock levels, set reorder alerts

6. Reports & Dashboards

- Track daily bookings, revenue, service types
- Mechanic performance metrics
- Customer retention rates

Tools & Technologies Used

- **Salesforce Lightning App Builder**
- **Custom Objects, Fields, and Relationships**
- **Apex Classes & Triggers**
- **Process Builder / Flows (Automation)**
- **Validation Rules**
- **Reports & Dashboards**
- **Visualforce Pages / Lightning Web Components (optional)**

Development Steps

1. **Plan the data model**
2. **Create custom objects and fields**
3. **Build page layouts and Lightning Pages**
4. **Configure Flows or Process Builder for automation**
5. **Develop Apex triggers for advanced logic (e.g. auto-create Job Cards)**
6. **Set up Reports & Dashboards**

7. Test and deploy using Change Sets

#Objectives:

Creating a Salesforce Developer account:

Change Your Password

Enter a new password for lead@sb.oom.
Make sure to include at least:

- ✓ 8 characters
- ✓ 1 letter
- ✓ 1 number

* New Password
..... Good

* Confirm New Password
..... Match

Security Question
▼ In what city were you born?

* Answer
asdfghjkl

Change Password

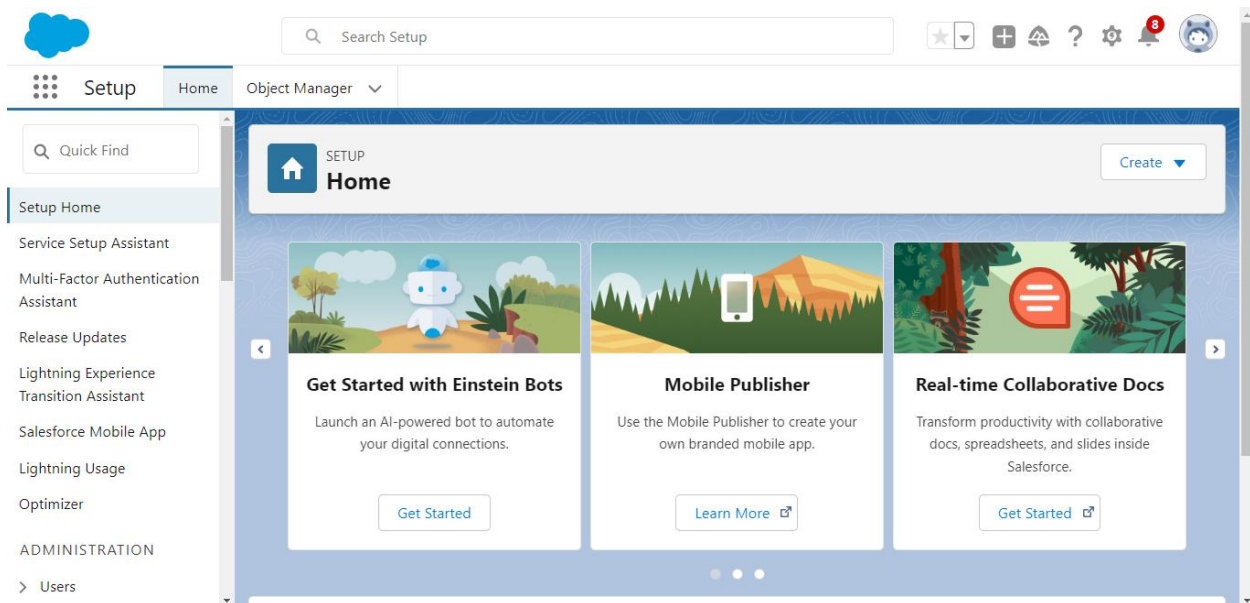
Sample Use Case

A customer visits the garage and requests an oil change. A service appointment is created, linked to their vehicle. A job card is generated and parts are logged. Once the service is completed, the system generates an invoice and updates inventory.

Possible Extensions

- Customer self-service portal (using **Experience Cloud**)
- SMS or email reminders for service dates
- Integration with payment gateway (via **REST API**)
- Mobile version using **Salesforce Mobile SDK**

Activating Developer Account:



#Objects

In a **Garage Management System (GMS)** built on Salesforce, **Objects** represent the core entities of the business—customers, vehicles, services, appointments, parts, etc. Salesforce uses both **Standard Objects** (like Account, Contact, Case) and **Custom Objects** (which you define) to model these entities.

Creating objects:

Optional Features

- ☒ Allow Reports
- ☐ Allow Activities
- ☐ Track Field History
- ☐ Allow in Chatter Groups
- ☐ Enable Licensing

Object Classification

When these settings are enabled, this object is classified as an Enterprise Application object. When these settings are disabled, this object is classified as a Light Application object. [Learn more](#)

- ☒ Allow Sharing
- ☒ Allow Bulk API Access
- ☒ Allow Streaming API Access

Deployment Status

[What is this?](#)

- ☐ In Development
- ☒ Deployed

Search Status

When this setting is enabled, your users can find records of this object type when they search. [Learn more](#)

- ☒ Allow Search

Object Creation Options (Available only when custom object is first created)

- ☐ Add Notes and Attachments related list to default page layout
- ☐ Launch New Custom Tab Wizard after saving this custom object

Buttons: Save, Save & New, Cancel

Key Objects in Garage Management System

1. Account (Standard Object)

Represents the customer (vehicle owner or corporate client).

- **Fields:** Name, Phone, Email, Address
- May relate to multiple vehicles and appointments.

2. Contact (Standard Object)

Linked to the Account to store the details of individuals.

- **Fields:** First Name, Last Name, Phone, Email
- Useful if the customer is part of a company account.

3. Vehicle__c (Custom Object)

Stores vehicle details owned by a customer.

- **Lookup to:** Account
- **Fields:**
 - Vehicle Name (Auto Number or Text)
 - Make (Picklist)

#Tabs

Creating Tabs

The screenshot shows a form titled "New Custom Object Tab" with a "Help for this Page" link. It is "Step 1 of 3". The instructions say: "Choose the custom object for this new custom tab. Fill in other details." The form contains the following fields:

- "Select an existing custom object or create a new custom object now" with a dropdown menu showing "Customer Details".
- "Tab Style" with a color selection bar and a preview icon.
- "(Optional) Choose a Home Page Custom Link to show as a splash page the first time your users click on this tab." with a "Splash Page Custom Link" dropdown menu showing "--None--".
- "Enter a short description." with a "Description" text area.

At the bottom right are "Next" and "Cancel" buttons.

Creating **tabs** in a **Garage Management System** typically refers to organizing the user interface (UI) into multiple sections or views using tabbed navigation. Tabs help improve usability by grouping related functionalities like **Vehicle Info**, **Customer Info**, **Invoices**, **Inventory**, etc.

Here's a breakdown of how to implement tabs in a Garage Management System, both from a **UI/UX** and **code** perspective:

The screenshot shows "Step 3 of 3" titled "Add to Custom Apps". The instructions say: "Choose the custom apps for which the new custom tab will be available. You may also examine or alter the visibility of tabs from the detail and edit pages of each Custom App." Below is a table with a list of custom apps and checkboxes to include the tab for each.

Custom App	<input type="checkbox"/> Include Tab
Platform (standard__Platform)	<input type="checkbox"/>
Sales (standard__Sales)	<input type="checkbox"/>
Service (standard__Service)	<input type="checkbox"/>
Marketing (standard__Marketing)	<input type="checkbox"/>
Sample Console (standard__ServiceConsole)	<input type="checkbox"/>
High Volume Customer Portal User	<input type="checkbox"/>
Authenticated Website User	<input type="checkbox"/>
App Launcher (standard__AppLauncher)	<input type="checkbox"/>

A red arrow points to the "Include Tab" checkbox header.

Purpose of Tabs in a Garage Management System

Tabs allow you to separate different modules or features for better user experience. Common tab categories might include:

1. **Dashboard**

2. **Customer Management**

3.Vehicle Records

4.Job Cards / Work Orders

5.Inventory Management

6.Billing & Invoicing

7.Reports

8.Settings / User Management

Lightning App

A **Lightning App** in the context of a **Garage Management System (GMS)**, especially if you're referring to **Salesforce Lightning**, is a modular, responsive, and component-based app that can be tailored to streamline operations like vehicle servicing, customer handling, invoicing, and parts inventory.

Garage Management System with Lightning App (Salesforce Context)

The screenshot shows the 'New Lightning App' setup page in Salesforce. The main heading is 'App Details & Branding'. Below this, there are two columns: 'App Details' and 'App Branding'. In the 'App Details' column, the 'App Name' field is highlighted with a red box and a red arrow pointing to it. Below it are fields for 'Developer Name' and 'Description'. In the 'App Branding' column, there is an 'Image' field with an 'Upload' button, and a 'Primary Color Hex Value' field with a dropdown menu and a text input showing '#0070D2'. At the bottom right, there is a 'Next' button highlighted with a red box and a red arrow pointing to it. The page also includes a progress bar at the bottom and an 'App Launcher Preview' section.

A **Lightning App** is a modern, dynamic app built on the **Salesforce Lightning Experience** platform. It's composed of **Lightning components** and **pages** that deliver a rich user interface and enhanced productivity features.

Features of a Garage Management System Lightning App

1. Customer Management

Track customer profiles, vehicle ownership, and service history.

Integration with contacts and accounts.

2. Vehicle Records

Log vehicle information (VIN, make, model, mileage).

Service schedules and status tracking.

3. Work Orders / Job Cards

Create and manage service orders.

Assign tasks to mechanics.

Track service progress.

Technologies Used

Lightning Web Components (LWC) – For building custom UI

Apex – Server-side logic (e.g., for calculating totals, managing workflows)

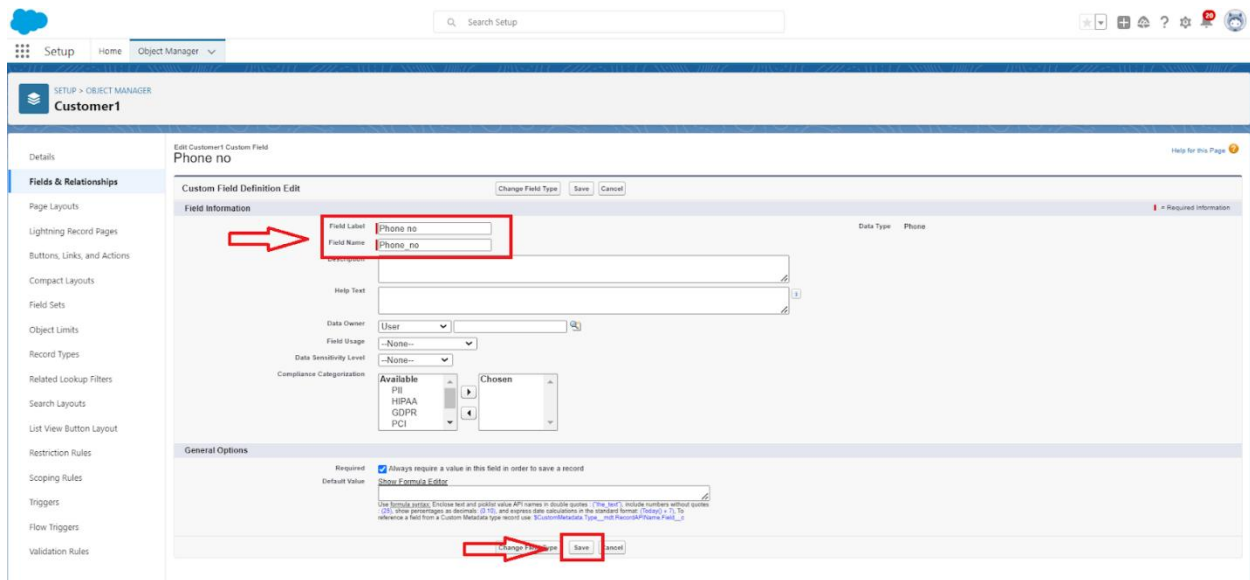
Salesforce Objects – Custom and standard (e.g., Account, Case, Opportunity)

Flows & Process Builder – Automate notifications, task assignments

Reports & Dashboards – Analytics

#Fields:

Creating Field objects:



Common Fields in a Garage Management System

I'll organize them by **module (table/entity)** — for each, I'll list useful fields and how you might create them conceptually in a database or form.

1 Customer Module

Purpose: Track customer details.

Fields:

Creation Example (SQL):

```
CREATE TABLE customers (
  customer_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100),
  phone_number VARCHAR(20),
  email VARCHAR(100),
  address TEXT
);
```

Creation of Lookup Fields:

Lookup Filter

Optionally, create a filter to limit the records available to users in the lookup field. [Tell me more!](#)

[Hide Filter Settings](#)

Filter Criteria [Insert Suggested Criteria](#) [Clear Filter Criteria](#)

Field	Operator	Value / Field
Appointment: Appointment Date	less than	Field Appointment: Created Date
ADD Begin typing to search for a field...	--None--	Value Clear

[Add Filter Logic...](#)

Filter Type ☒ **Required.** The user-entered value must match filter criteria.
If it doesn't, display this error message on save:
Value does not exist or does not match filter criteria.
[Reset to default message](#)

☐ **Optional.** The user can remove the filter or enter values that don't match criteria.

Lookup Window Text Add this informational message to the lookup window.

Active ☒ **Enable this filter.**

[Change Field Type](#) [Save](#) [Cancel](#)

2 Vehicle Module

Purpose: Store vehicle details.

Fields:

Creation Example:

```
CREATE TABLE vehicles (  
    vehicle_id INT PRIMARY KEY AUTO_INCREMENT,  
    customer_id INT,  
    registration_number VARCHAR(20),  
    make VARCHAR(50),  
    model VARCHAR(50),  
    year INT,  
    color VARCHAR(30),  
    chassis_number VARCHAR(50),  
    last_service_date DATE,  
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)  
);
```

3 Job/Service Module

Purpose: Track jobs or services done for a vehicle.

Fields:

Creation Example:

```
CREATE TABLE jobs (
  job_id INT PRIMARY KEY AUTO_INCREMENT,
  vehicle_id INT,
  job_description TEXT,
  start_date DATE,
  end_date DATE,
  status VARCHAR(20),
  total_cost DECIMAL(10,2),
  FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
);
```

Creation of Checkbox Field on Appointment Object :

SETUP > OBJECT MANAGER
Appointment

Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Search Layouts
List View Button Layout
Restriction Rules

Specify the type of information that the custom field will contain.

Data Type

Select one of the data types below:

- ☐ None Selected
- ☐ Auto Number
- ☐ Formula
- ☐ Roll-Up Summary
- ☐ Lookup Relationship
- ☐ Master-Detail Relationship
- ☐ External Lookup Relationship
- ☒ **Checkbox**
- ☐ Currency

Next Cancel

4 Parts Module

Purpose: Manage parts inventory and usage.

Fieldunit_price

Creation Example:

```
CREATE TABLE parts (
  part_id INT PRIMARY KEY AUTO_INCREMENT,
  part_name VARCHAR(100),
  stock_quantity INT,
  unit_price DECIMAL(10,2),
  supplier VARCHAR(100),
  category VARCHAR(50)
);
```

Creation of Text Fields:

Step 2. Enter the details

Step 2 of 4

Previous

Next

Cancel

Field Label

Vehicle number plate

Please enter the maximum length for a text field below.

Length

10

Field Name

Vehicle_number_plate

Description

Help Text

Required

☒

Always require a value in this field in order to save a record

Unique

☒

Do not allow duplicate values

☒ Treat "ABC" and "abc" as duplicate values (case insensitive)
 ☐ Treat "ABC" and "abc" as different values (case sensitive)

External ID

☐

Set this field as the unique record identifier from an external system

Auto add to custom report type

☒

Add this field to existing custom report types that contain this entity

Creating Formula Field in Service records Object:

Step 3. Enter formula

Step 3 of 5

Previous

Next

Cancel

Enter your formula and click Check Syntax to check for errors. Click the Advanced Formula subtab to use additional fields, operators, and functions.

Example: `Reminder Date = CloseDate - 7` [More Examples...](#)

Simple Formula

Advanced Formula

Insert Field

Insert Operator

service dates (Date) =

CreatedDate

Functions

-- All Function Categories --

ABS

ACOS

ADDMONTHS

AND

ASCII

Quick Tips

Getting Started

Operators & Functions

A Garage Management System typically manages **Customers, Vehicles, Jobs, Parts, Billing, and optionally Staff.**

You can create these fields via **SQL tables** in a database or **form inputs** in your application UI, ensuring good relational design with appropriate primary and foreign keys.

Ideation Phase

Brainstorm & Idea Prioritization


Date	29 June 2025
Team ID	LTVIP2025TMID28899
Project Name	Garage Management System
Maximum Marks	4 Marks

Brainstorm & Idea Prioritization Template:

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Step-1: Team Gathering, Collaboration and Select the Problem Statement



Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

🕒 10 minutes to prepare
🕒 1 hour to collaborate
👤 2-8 people recommended

➔

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

🕒 10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

[Open article](#) ➔

1


Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

🕒 5 minutes

PROBLEM

How might we [your problem statement]?



Key rules of brainstorming

To run a smooth and productive session

- 🕒 Stay in topic.
- 💡 Encourage wild ideas.
- 🕒 Defer judgment.
- 👂 Listen to others.
- 🗨️ Go for volume.
- 👁️ If possible, be visual.

Step-2: Brainstorm, Idea Listing and Grouping

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

🕒 10 minutes

TIP

You can select a sticky note and hit the pencil icon to start drawing!

Amar

Yuktesh

Person 3

Person 4

Person 5

Person 6

Person 7

Person 8

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

🕒 20 minutes

Person 4

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

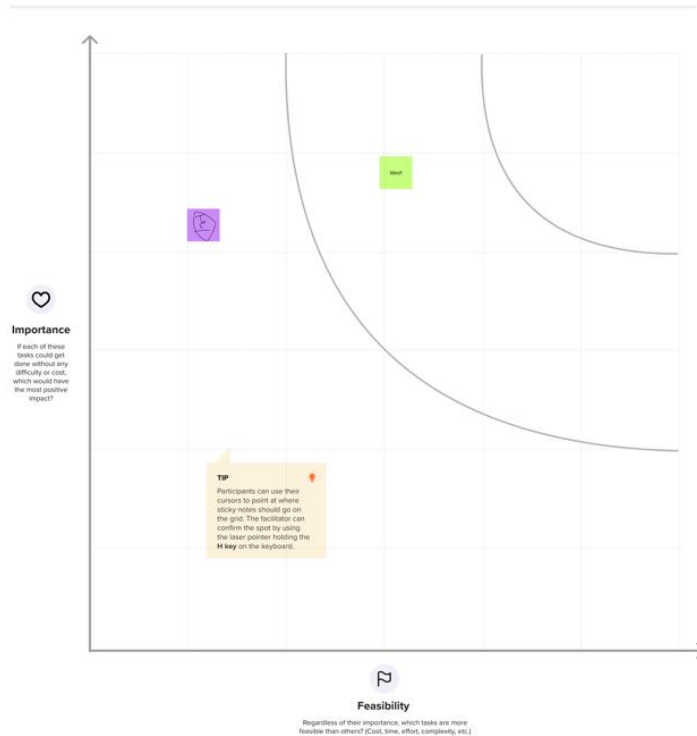
Step-3: Idea Prioritization

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

⌚ 20 minutes



Idea Prioritization (Using Value vs. Effort Matrix)

Let's map these ideas by **Value to Business** and **Effort to Implement**:

Idea	Value	Effort	Priority
Customer & Vehicle Management	High	Low	☑ Must-have
Job Card Management	High	Low	☑ Must-have
Billing & Invoicing	High	Low	☑ Must-have
Appointment Scheduling	High	Medium	☑ Must-have

Inventory Management	Medium	Medium	Good to have
Reports & Analytics	Medium	Medium	Good to have
Automated Service Reminders	High	Medium	Good to have
Mobile App for Customers	Medium	High	Optional/Phase 2
AI-based Repair Cost Estimator	High	High	Future
Predictive Maintenance Suggestions	Medium	High	Future
Online Payment Integration	Medium	Medium	Good to have
Digital Inspection Forms	Medium	Medium	Good to have

Empathize & Discover

Date	28 June 2025
Team ID	LTVIP2025TMID28899
Project Name	TO GARAGE MANAGEMENT SYSTEM
College Name	Ideal institute of Technology
Team members	*Amaladasu Sony Amaladasusony322005@gmail.com *Kola.Prashanthi kolaprashanthi1@gmail.com

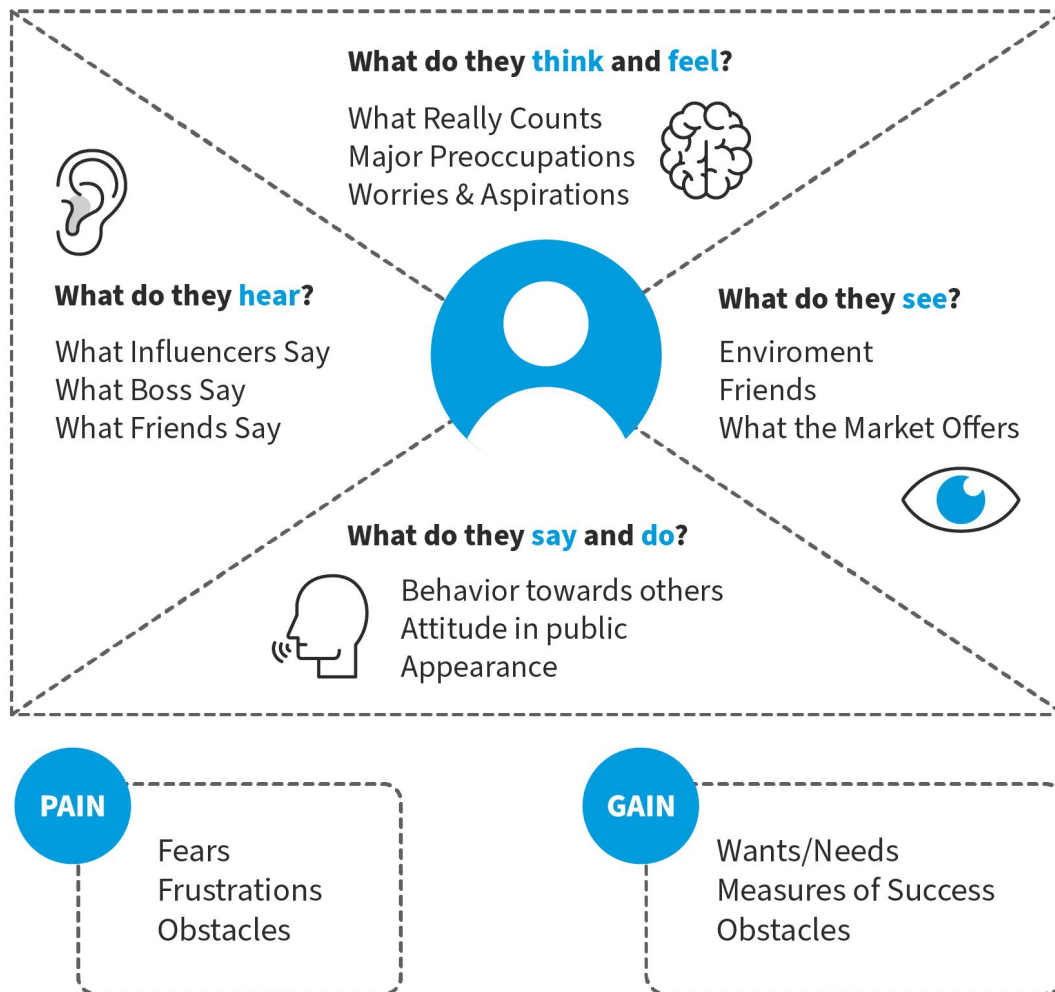
Empathy Map Canvas:

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.

It is a useful tool to help teams better understand their users.

Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the

Empathy Map



Project Design Phase
Problem – Solution Fit

Date	28 June 2025
Team ID	LTVIP2025TMID28899
Project Name	TO GARAGE MANAGEMENT SYSTEM
College Name	Ideal institute of Technology

Problem – Solution Fit Template:

The Problem-Solution Fit simply means that you have found a problem with your customer and that the solution you have realized for it actually solves the customer's problem. It helps entrepreneurs, marketers and corporate innovators identify behavioral patterns and recognize what would work and why

Purpose:

Solve complex problems in a way that fits the state of your customers.

Succeed faster and increase your solution adoption by tapping into existing mediums and channels of behavior.

Sharpen your communication and marketing strategy with the right triggers and messaging.

Increase touch-points with your company by finding the right problem-behavior fit and building trust by solving frequent annoyances, or urgent or costly problems.

Understand the existing situation in order to improve it for your target group.

Template:

Define CS, fit into CC	1. CUSTOMER SEGMENT(S) Who is your customer? I.e. working parents of 0-5 y.o. kids	CS	6. CUSTOMER CONSTRAINTS What constraints prevent your customers from taking action or limit their choices of solutions? I.e. spending power, budget, no cash, network connection, available devices.	CC	5. AVAILABLE SOLUTIONS Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? I.e. pen and paper is an alternative to digital notetaking.	AS	Explore AS, differentiate
Focus on J&P, tap into BE, understand RC	2. JOBS-TO-BE-DONE / PROBLEMS Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.	J&P	9. PROBLEM ROOT CAUSE What is the real reason that this problem exists? What is the back story behind the need to do this job? I.e. customers have to do it because of the change in regulations.	RC	7. BEHAVIOUR What does your customer do to address the problem and get the job done? I.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (I.e. Greenpeace)	BE	Focus on J&P, tap into BE, understand RC
Identify strong TR & EM	3. TRIGGERS What triggers customers to act? I.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news.	TR	10. YOUR SOLUTION If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.	SL	8. CHANNELS of BEHAVIOUR 8.1 ONLINE What kind of actions do customers take online? Extract online channels from #7	CH	Extract online & offline CH of BE
	4. EMOTIONS: BEFORE / AFTER How do customers feel when they face a problem or a job and afterwards? I.e. lost, insecure > confident, in control - use it in your communication strategy & design.	EM			8.2 OFFLINE What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.		

**Project Design Phase
Proposed Solution**

Date	28 June 2025
Team ID	LTVIP2025TMID28899
Project Name	TO GARAGE MANAGEMENT SYSTEM
College Name	Ideal institute of Technology
Team members	*Amaladasu Sony Amaladasusony322005@gmail.com *Kola.Prashanthi kolaprashanthi1@gmail.com

Proposed Solution Template:

Project team shall fill the following information in the proposed solution template.

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Many existing AI systems struggle with natural conversational flow—they often lose context, can't handle follow-up questions, make unsupported inferences, and may produce incorrect or irrelevant answers. This makes human–computer interaction feel rigid, clunky, or even misleading.
1.	Idea / Solution description	A Garage Management System is a centralized software platform designed to digitize and streamline every key operation of an auto repair business—from appointment booking to invoicing, inventory, and analytics.
1.	Novelty / Uniqueness	Unlike traditional workshop tools, this next-gen GMS blends AI, IoT, cloud, and automation into a seamless, user-friendly, and efficient platform . It reduces manual workloads, enhances transparency, boosts customer loyalty, and empowers garage owners with actionable insights—all delivered through an intuitive visual interface that works across mobile and desktop.
1.	Social Impact / Customer Satisfaction	By enabling efficient parts ordering, cooperative inventory, and business training, GMS empowers independent garages and mechanics—especially in underserved communities—to access needed parts faster, reduce costs, and offer competitive pricing

1.	Business Model (Revenue Model)	Most GMS platforms operate on a recurring subscription model. Garages pay monthly or annual fees that scale with user count or feature tiers—e.g., Basic, Professional, and Premium. This ensures predictable revenue for the provider and manageable costs for clients
----	-----------------------------------	---

Solution Architecture

High-Level Overview

Users:

Customers

Mechanics/Technicians

Receptionist/Service Advisor

Garage Owner/Admin

Inventory Manager

Core Components:

Web/Mobile Application

Backend Server / API Layer

Database

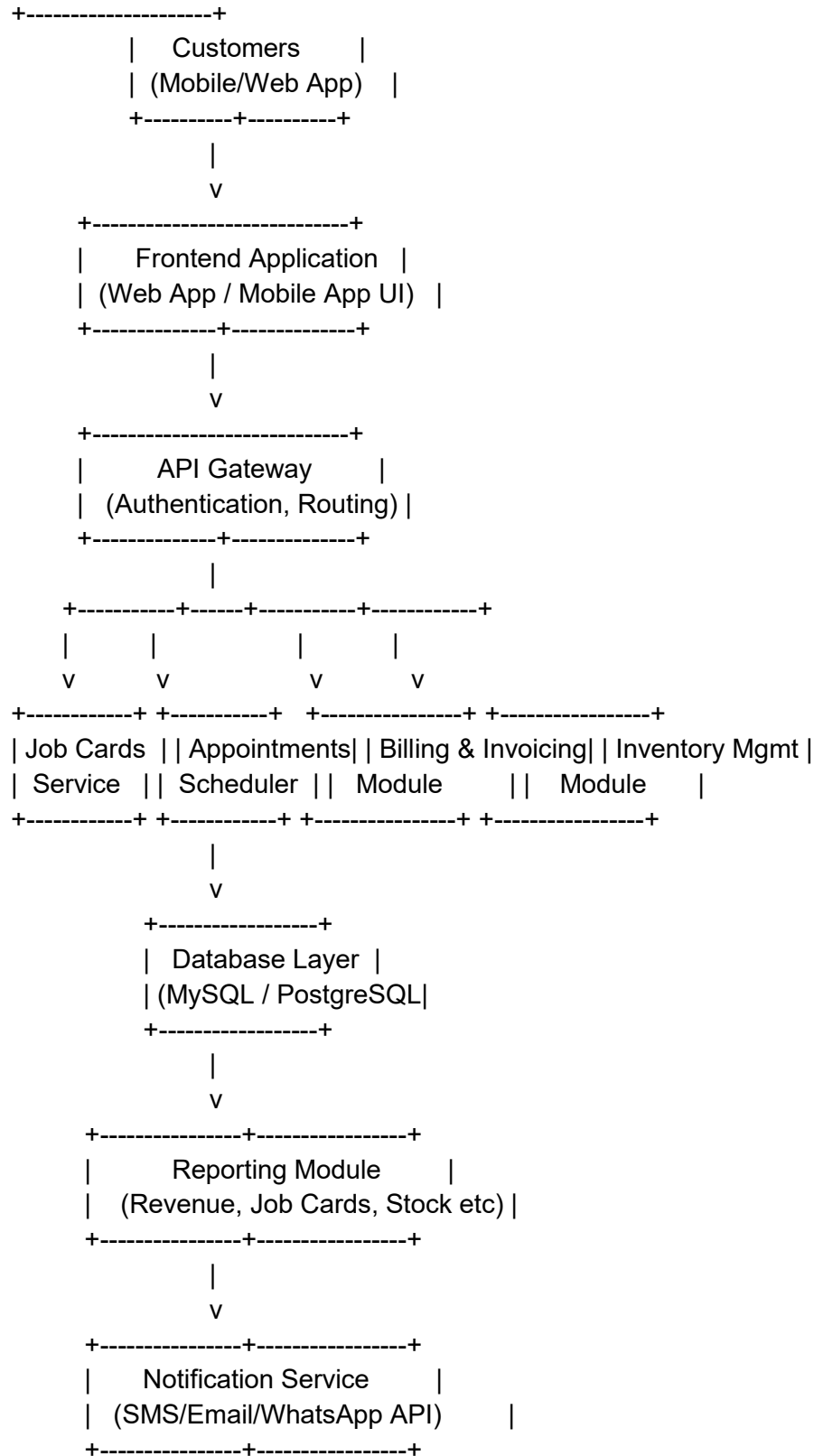
Notification Service (SMS/WhatsApp/Email)

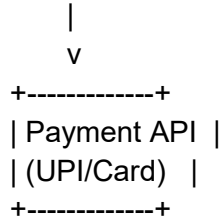
Payment Gateway

Report & Analytics Module

(Optional) AI Prediction Module for cost estimation, stock forecasting

Requirement Analysis:





Technology Stack (Architecture & Stack):

Date	28 June 2025
Team ID	LTVIP2025TMID28899
Project Name	TO GARAGE MANAGEMENT SYSTEM
College Name	Ideal institute of Technology
Team members	*Amaladasu Sony Amaladasusony322005@gmail.com *Kola.Prashanthi kolaprashanthi1@gmail.com

Technical Architecture

Table-1: Components & Technologies:

Component	Description	Suggested Technologies / Tools
Frontend (User Interface)	Interface for interacting with system features	HTML5, CSS3, JavaScript, React.js / Angular / Vue.js
Backend (Server Logic)	Handles business logic, API endpoints, and database interaction	Node.js / Express.js, Django (Python), Laravel (PHP), .NET
Database	Stores customer, vehicle, job, and inventory data	MySQL, PostgreSQL, MongoDB
Authentication Module	Handles user login, roles, and access control	Firebase Auth, JWT, OAuth2, Passport.js
Inventory	Manages parts, stock levels,	Backend + Database

Component	Description	Suggested Technologies / Tools
Management	and alerts	Logic, RESTful APIs
Job Management System	Manages job cards, assignments, and progress	Custom Backend Logic, Status Workflow Modules
Invoice & Billing Engine	Generates invoices and processes payments	Stripe / PayPal APIs, PDFKit (for printable invoices)
Notification System	Sends SMS/email for reminders and updates	Twilio (SMS), SendGrid / Mailgun (Email)
Reporting & Analytics	Provides reports and dashboards	Chart.js / D3.js (Frontend), SQL Queries, Power BI (optional)
Cloud Hosting / Deployment	Hosts and deploys the application	AWS, Azure, Heroku, DigitalOcean, Netlify (Frontend)
Version Control	Code collaboration and tracking	Git, GitHub / GitLab / Bitbucket
Mobile App (Optional)	Access system via mobile devices	React Native, Flutter, Kotlin / Swift
Backup & Recovery	Ensures data is backed up and restorable	AWS S3, Scheduled DB Backups, Cloud Storage Solutions
1. Frontend (Web UI)	Interfaces for staff (admin, mechanics, reception) to interact with the system	React.js, Angular, Vue.js, Bootstrap, Tailwind CSS
2. Frontend (Customer Portal)	Allows customers to view service history, schedule appointments, view invoices	React.js / Next.js, HTML/CSS, Responsive Design
3. Backend Server	Handles all business logic, REST APIs, and system operations	Node.js + Express, Django (Python), Laravel (PHP), ASP.NET Core

Component	Description	Suggested Technologies / Tools
4. Database	Stores customer, vehicle, service, billing, and inventory data	PostgreSQL, MySQL, MongoDB, Microsoft SQL Server
5. Authentication & Authorization	Manages login, session, role-based access control (RBAC)	JWT, OAuth 2.0, Firebase Auth, Passport.js
6. API Gateway	Manages external/internal API traffic, rate limiting, and routing	NGINX, Express Gateway, Kong API Gateway
7. Job Management System	Create and manage job cards, assign mechanics, track job status	Custom Backend Logic, Kanban-like modules, Drag-and-drop UI
8. Inventory Management	Manages parts inventory, reorder alerts, suppliers, and purchase orders	RESTful API + DB Schema + Admin Interface
9. Invoice & Billing System	Generates and processes invoices, taxes, payment tracking	PDFKit (PDF generation), Stripe/PayPal APIs, Razorpay (India)
10. Notification System	Sends SMS, email notifications for reminders and updates	Twilio (SMS), SendGrid, Mailgun, Firebase Cloud Messaging (FCM)
11. Reporting & Analytics	Graphs and reports on jobs, revenue, mechanic performance, inventory, etc.	Chart.js, D3.js, Google Charts, Power BI, Metabase
12. Scheduler / Calendar	Visual scheduling interface for appointments and job allocation	FullCalendar.js, Google Calendar API
13. File Management	Uploads & stores documents (RC copies, invoices, job cards)	AWS S3, Firebase Storage, Cloudinary
14. Mobile App	Mobile version for customers	React Native, Flutter,

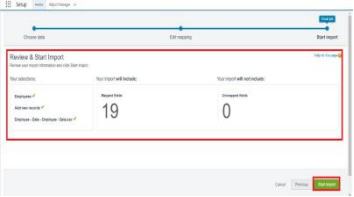

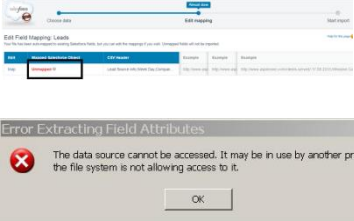
Component	Description	Suggested Technologies / Tools
(Optional)	or mechanics	Kotlin (Android), Swift (iOS)
15. Cloud Hosting / Deployment	Hosting and continuous deployment infrastructure	AWS EC2/S3, Heroku, DigitalOcean, Vercel, Netlify
16. CI/CD Pipeline	Automates testing and deployment of updates	GitHub Actions, GitLab CI/CD, Jenkins, CircleCI
17. Logs & Monitoring	System logs, performance monitoring, error tracking	LogRocket, Sentry, ELK Stack, Prometheus + Grafana
18. Backup & Disaster Recovery	Scheduled DB backups, restore mechanisms	AWS RDS Snapshots, Cron Jobs, Google Cloud Backup
19. QR Code/Barcode Scanner	For identifying parts or vehicles via barcode/QR code	jsQR, QuaggaJS, or native scanner APIs in mobile apps
20. Payment Integration	Secure online payment via gateway	Stripe, Razorpay, PayPal, Square

Project Development Phase Model Performance Test

Date	28 June 2025
Team ID	LTVIP2025TMID28899
Project Name	TO GARAGE MANAGEMENT SYSTEM
College Name	Ideal institute of Technology

Model Performance Testing:

Project team shall fill the following information in model performance testing template.

S.No.	Parameter	Values	Screenshot
1.	Model Summary	<p>Salesforce automation setup for Data management using Object, Fields and Reports.</p> <p>Note : Import Records if data Match Correctly then Records will Created or Else it will Show Error</p>	
2.	Accuracy	<p>Training Accuracy - 98%</p> <p>Validation Accuracy - 98%</p>	<p>Congratulations, your import has started!</p> <p>Click OK to view your import status on the Bulk Data Load Job page.</p> 
3.	Confidence Score (Only Yolo Projects)	<p>Class Detected - If detecting Object and fields name if wrong and other activity</p> <p>Confidence Score - If the model is 92% sure the object is correctly detected</p>	

#validation rule:

A **validation rule** is a condition that data input must meet before it's accepted by the system. It prevents invalid, incomplete, or illogical data from being stored in your database.

For example:

A vehicle registration number must not be empty

A phone number must contain only numbers and be 10 digits long.

The job start date cannot be later than the end date.

To create a validation rule to an Appointment Object:



Example: Validation in Code

HTML + JavaScript Example (Front-End Validation)

```
<input type="text" id="phone_number" name="phone_number" required>
<script>
function validatePhoneNumber() {
  let phone = document.getElementById("phone_number").value;
  if (!/^d{10}$/.test(phone)) {
    alert("Phone number must be exactly 10 digits.");
    return false;
  }
  return true;
}
```

```
}  
</script>
```

SQL Constraint Example

```
ALTER TABLE vehicles  
ADD CONSTRAINT chk_year CHECK (year >= 1980 AND year <= YEAR(CURDATE()));
```

#duplicate rule:

To create a matching rule to an Customer details Object:

Rule Details

Object: Customer Details

Rule Name: matching_Customer deta

Unique Name: matching_Customer_det

Description:

Matching Criteria

Tell the rule which fields to compare and how.

Field	Matching Method	Match Blank Fields
Email	Exact	<input type="checkbox"/>
Phone Number	Exact	<input type="checkbox"/>
--None--	Exact	<input type="checkbox"/>
--None--	Exact	<input type="checkbox"/>
--None--	Exact	<input type="checkbox"/>

AND

AND

AND

AND

Add Filter Logic...

Save Cancel

A **duplicate rule** is a set of conditions defined in a system to **prevent the creation of duplicate records** in your database.

It ensures that critical data — like customer records, vehicle registrations, or invoices — remains **unique and consistent**.

Inventory mismanagement

Service job tracking errors

Data inconsistencies when generating reports

Example:

If two customer records with the same phone number are created, the system might misattribute a job to the wrong customer, or overbook a vehicle.

Rule Example:

No two customer records can have the same phone number or email.

How to Implement Duplicate Rules

At Different Levels:

1. Front-End (Form Validation)

Check for existing records before allowing new ones.

```
function checkDuplicatePhone(phoneNumber) {  
  // Assuming you have a list of phone numbers from the database  
  if(existingPhoneNumbers.includes(phoneNumber)) {  
    alert("This phone number already exists!");  
    return false;  
  }  
  return true;  
}
```

2. Back-End (Server-Side Validation)

Before inserting new data, check for existing records in the database.

```
SELECT * FROM customers WHERE phone_number = '9876543210';
```

If a record exists — reject the insert/update request.

3. Database Level (Unique Constraints)

SQL Example

```
ALTER TABLE vehicles  
ADD CONSTRAINT unique_registration UNIQUE (registration_number);
```

This will automatically block any attempt to insert a duplicate vehicle registration number.

Best practice:

Use a combination of **front-end checks**, **back-end validation**, and **database constraints** for strong, reliable duplicate prevention.

#Profile:

A **Profile** in a GMS refers to a set of personal or operational details stored for a system user or entity, typically used to **manage identity, access control, and operational settings**.

There can be different types of profiles:

Manager Profile:

Custom Object Permissions						
	Basic Access				Data Administration	
	Read	Create	Edit	Delete	View All	Modify All
Appointments	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Billing details and feedback	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Customer Details	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Environments	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

	Basic Access				Data Administration	
	Read	Create	Edit	Delete	View All	Modify All
Laptops	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Service records	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SessionData	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Customer Profiles

Vehicle Profiles

Staff/Employee Profiles

User Accounts (System Admin, Technician, Manager, etc.) Profiles

Each profile stores relevant information and preferences related to the entity it represents.

Types of Profiles and Their Fields

1 Customer Profile

Stores personal details of the customer.

Fields:

Email

Address

Vehicle(s) Associated

Service History Summary

Total Amount Spent

Purpose:

Track individual customer service history, contact details, and loyalty metrics.

2 Vehicle Profile

Stores all details of a vehicle registered in the system.

Fields:

-

Purpose:

Control who can access which parts of the system and manage account security.

Example: Customer Profile Layout (UI Concept)

```
-----  
| Customer Name: John Doe           |  
| Phone Number: +91-9876543210      |  
| Email: johndoe@gmail.com          |  
| Address: 123, Main Road, Hyderabad |  
| Vehicles Owned: MH12AB1234, MH12CD5678 |  
| Total Services Availed: 5          |  
| Total Amount Spent: ₹15,000        |  
-----
```

How to Implement Profiles Technically

Database Example (Customer Table):

```
CREATE TABLE customers (  
  customer_id INT PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(100),  
  phone_number VARCHAR(20) UNIQUE,  
  email VARCHAR(100) UNIQUE,  
  address TEXT  
);
```

User Profile Example (User Accounts Table):

```
CREATE TABLE users (  
    user_id INT PRIMARY KEY AUTO_INCREMENT,  
    username VARCHAR(50) UNIQUE,  
    password_hash VARCHAR(255),  
    role VARCHAR(20),  
    last_login DATETIME,  
    status VARCHAR(10)  
);
```

#role and role heirarchy:

Creating another Manager Roles:

[Collapse All](#) [Expand All](#)

 **Thesmartbridge**

 **Add Role**
 **CEO** [Edit](#) | [Del](#) | [Assign](#)

 **Add Role**
 **CFO** [Edit](#) | [Del](#) | [Assign](#)

 **Add Role**
 **COO** [Edit](#) | [Del](#) | [Assign](#)

 **Add Role**
 **Manger** [Edit](#) | [Del](#) | [Assign](#)

 **Add Role**
 **SVP, Customer Service & Support** [Edit](#) | [Del](#) | [Assign](#)

 **Add Role**
 **SVP, Human Resources** [Edit](#) | [Del](#) | [Assign](#)

 **Add Role**
 **SVP, Sales & Marketing** [Edit](#) | [Del](#) | [Assign](#)

 **Add Role**

A **Role** defines a set of permissions and responsibilities assigned to a user or employee within the system. It controls:

What data a user can access

- What actions they can perform (view, edit, delete, assign jobs, generate bills, etc.)
- What system modules they can interact with

Common Roles in a Garage Management System

Role Name	Description
Admin	Full access to all system modules, settings, and reports.
Manager	Can manage staff, jobs, invoices, and approve major actions.
Receptionist	Can register customers, schedule jobs, and generate bills.
Technician	Can view and update assigned jobs and parts used.
Customer (if system has online portal)	Can view their service history and invoices.

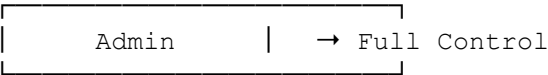
Role Hierarchy

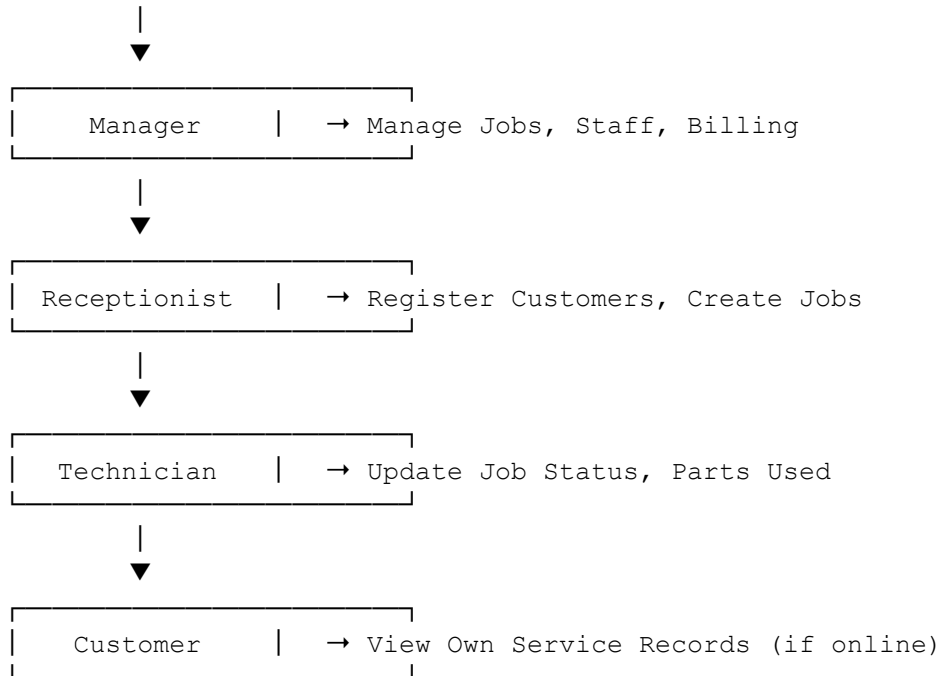
A **Role Hierarchy** is a structured arrangement where roles are organized in levels based on their authority and system access.
Higher roles inherit permissions from lower roles and have additional privileges.

Example:

- Admin** can do everything a **Manager** does, plus more.
 - Manager** can do everything a **Receptionist** does, plus more.
-

Example: Role Hierarchy Diagram for a Garage Management System





How to Implement Roles & Hierarchy (Technical Concept)

Database Table: roles

```
CREATE TABLE roles (  
  role_id INT PRIMARY KEY AUTO_INCREMENT,  
  role_name VARCHAR(50),  
  parent_role_id INT  
);
```

Sample Data:

role_id	role_name	parent_role_id
1	Admin	NULL
2	Manager	1
3	Receptionist	2
4	Technician	3
5	Customer	4

Assigning Role to User

```
CREATE TABLE users (  
  user_id INT PRIMARY KEY AUTO_INCREMENT,  
  username VARCHAR(50) UNIQUE,  
  password_hash VARCHAR(255),  
  role_id INT,  
  FOREIGN KEY (role_id) REFERENCES roles(role_id)  
);
```

#users and creating users:

User

A **User** in a GMS is anyone who can log in to and interact with the system. This typically includes:

Internal users: Admin, Managers, Receptionists, Technicians

External users (optional): Customers (if you have an online customer portal or app)

Each user has:

A unique identity (username)

A secure password (usually hashed)

An assigned **role** (which defines their system permissions)

Types of Users in a Garage Management System

User Type	Description
Admin	Full control — can manage system settings, users, and reports
Manager	Manages jobs, staff, invoices, and approvals
Receptionist	Handles customer registrations, job bookings, invoicing
Technician	Can view and update assigned jobs and parts used
Customer (if online portal)	Can view their own service records, bills, and book appointments

How to Create Users in a Garage Management System

1 In the Database

User Table SQL Example

```
CREATE TABLE users (  
  user_id INT PRIMARY KEY AUTO_INCREMENT,  
  username VARCHAR(50) UNIQUE NOT NULL,  
  password_hash VARCHAR(255) NOT NULL,  
  email VARCHAR(100),  
  phone_number VARCHAR(20),  
  role_id INT NOT NULL,  
  status VARCHAR(10) DEFAULT 'Active',  
  last_login_time DATETIME,  
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ONUPDATE CURRENT_TIMESTAMP,  
  FOREIGN KEY (role_id) REFERENCES roles(role_id)  
);
```

Example Form Layout:

Field Name	Input Type
Username	Text
Password	Password
Confirm Password	Password
Email	Email
Phone Number	Text
Role	Dropdown (Admin, Manager, etc.)
Status	Toggle (Active/Inactive)

Password Storage Note:

- ✓ Never store passwords in plain text.
- ✓ Use a secure hashing algorithm like **bcrypt**, **argon2**, or **SHA-256 (with salt)**.

Example (PHP using bcrypt):


```
$password = password_hash($inputPassword, PASSWORD_BCRYPT);
```

Concept	Description
User	A person with login access to the system
Fields	Username, password (hashed), email, phone, role, status, etc.
Roles	Defines permissions (Admin, Manager, Technician, etc.)
Creation Methods	Through database inserts, admin dashboard forms
Security	Password hashing, account status control, role-based access

#Public groups:

Creating New Public Groups:

The screenshot shows the 'New Public Group' configuration page in Salesforce. At the top, there are 'Save' and 'Cancel' buttons. The page is titled 'New Public Group' with a red exclamation mark icon indicating required information. The 'Label' field contains 'Sales Team' and the 'Group Name' field also contains 'Sales Team'. Below these fields, the 'Grant Access Using Hierarchies' checkbox is checked. A search bar is present with 'Roles' selected in the dropdown and a 'Find' button. The 'Available Members' list on the left includes roles such as 'Customer Support, North America', 'Director, Channel Sales', 'Director, Direct Sales', 'Eastern Sales Team', 'Installation & Repair Services', 'Manager', 'Marketing Team', 'SVP, Customer Service & Support', 'SVP, Human Resources', 'SVP, Sales & Marketing', 'VP, International Sales', 'VP, Marketing', 'VP, North American Sales', and 'Western Sales Team'. The 'Selected Members' list on the right contains 'Role: Sales person'. Between the two lists are 'Add' and 'Remove' buttons. At the bottom of the page, there is a link 'Add to Delegated Administration Groups'.

A **Public Group** in Salesforce is a **collection of users, roles, and other public groups** that you can use to simplify user management, data access, sharing rules, and security permissions within your Salesforce org.

They are designed to:

- Grant shared access to records

- Simplify sharing rules configuration

- Support workflow email alerts

Manage folder, report, and dashboard access

Example Use Case:

Create a public group called Technician Team consisting of all technicians in your Garage Management System Salesforce app, then use a **sharing rule** to give them read/write access to service job records.

Example Public Group Structure

Public Group Name	Members
Admin Group	Admin User 1, Admin User 2
Service Managers	Role: Service Manager
Technician Team	User: Tech1, User: Tech2, Role: Technician

#Sharing setting:

Creating Sharing settings:

The screenshot displays the Salesforce Sharing Settings interface. The top section, titled 'Sharing Settings', contains a list of objects and their corresponding sharing settings. The 'Service records' row is highlighted with a red box, indicating the selected sharing setting is 'Private'. Below this, the 'User Visibility Settings' section shows 'Portal User Visibility' and 'Site User Visibility' both set to 'Off'. The 'Other Settings' section includes 'Standard Report Visibility' (checked), 'Manual User Record Sharing' (unchecked), and 'Manager Groups' (unchecked). At the bottom, the 'Save' button is highlighted with a red box.

Object	Sharing Setting	Visible
Work Plan Template	Private	<input type="checkbox"/>
Work Step Template	Private	<input type="checkbox"/>
Work Type	Private	<input type="checkbox"/>
Work Type Group	Public Read/Write	<input type="checkbox"/>
Appointment	Public Read/Write	<input checked="" type="checkbox"/>
Billing details and feedback	Public Read/Write	<input checked="" type="checkbox"/>
Customer Details	Public Read/Write	<input checked="" type="checkbox"/>
Environment	Public Read/Write	<input checked="" type="checkbox"/>
Laptop	Public Read/Write	<input checked="" type="checkbox"/>
Service records	Private	<input checked="" type="checkbox"/>
SessionData	Public Read/Write	<input checked="" type="checkbox"/>

User Visibility Settings

Portal User Visibility ☐ Site User Visibility ☐

Other Settings

Standard Report Visibility ☒ Manual User Record Sharing ☐ Manager Groups ☐

Minimize the number of roles created, which improves performance by cutting down processing loads ☒ Grant site users access to related cases ☒ Secure guest user record access ☐ Require permission to view record names in lookup fields ☐

Save **Cancel**

Sharing Settings in Salesforce control **how records are accessed, shared, and secured within your organization.**

They work in combination with roles, profiles, permission sets, and rules to manage both **object-level** and **record-level access**.

Core Components of Salesforce Sharing Architecture

Let's review each component with a concise explanation:

1 Organization-Wide Default (OWD)

What it is:

The **baseline level of access** that all users have to records they **don't own** for each object in Salesforce.

OWD Access Levels:

Access Level	Description
Private	Only the record owner and people above them in the role hierarchy can access the record
Public Read-Only	Everyone can view, but only owners and admins can edit
Public Read/Write	Everyone can view and edit the records
Controlled by Parent	Access depends on the parent record's sharing setting

Why it's important:

It defines the **default security posture** for every object in the org.

2 Role Hierarchy

What it is:

A **tree-like structure** that defines who can view, edit, and report on records owned by users in lower roles.

Key Principle:

Users higher in the hierarchy automatically inherit access to records owned by users below them (if OWD allows).

Why it's important:

Enables **delegated access and management oversight**.

Example:

A Service Manager can view and edit all records owned by Technicians reporting to them.

Types of Sharing Rules:

-

Type	Purpose
Owner-based	Share records owned by certain users/roles/groups
Criteria-based	Share records that meet specific field values

Summary Table: Salesforce Sharing Tools

Tool/Feature	Controls Access at	Purpose
OWD	Record-level	Define baseline default access for each object
Role Hierarchy	Record-level	Grant access upward through organizational reporting lines
Profiles	Object/Field-level	Define general access and permissions at object and field levels
Permission Sets	Object/Field-level	Extend permissions to users on top of their profile
Sharing Rules	Record-level	Automatically share records based on ownership or criteria
Manual Sharing	Record-level	Manually share specific records as exceptions

Together, they create a **robust, scalable, and flexible security model** tailored to organizational needs.

#Flows:

Create a Flow:

Set Filter Conditions

Condition Requirements to Update Record

All Conditions Are Met (AND) ▼

Field	Operator	Value
Payment_Status__c	Equals ▼	Completed

+ Add Condition

Set Field Values for the Billing details and feedback Record

Field	Value
Payment_Paid__c	← \$Record > Service records > Appointment > Service A... X

+ Add Field

Cancel Done

Record-Triggered Flow

Start

Save as

A New Version A New Flow

*Flow Label

Billing Amount Flow

*Flow API Name

Billing_Amount_Flow

Description

Show Advanced

Cancel Save

- ✓ Automate complex, multi-step business processes
- ✓ Eliminate manual data entry or approval chains
- ✓ Display custom screens to users for guided interactions
- ✓ Update related records across multiple objects
- ✓ Replace older Process Builders and Workflow Rules (Salesforce is migrating to Flow-first strategy)

Types of Flows in Salesforce

Salesforce Flows come in several types based on how and when they are triggered:

Type	Description
Screen Flow	Interactive flows with screens for users to input or view data
Record-Triggered Flow	Runs automatically when a record is created, updated, or deleted
Schedule-Triggered Flow	Runs automatically at a scheduled time
Auto-launched Flow	Runs in the background without user interaction; invoked by Apex, process builder, or another flow
Platform Event-Triggered Flow	Runs when a platform event message is received

A **Flow** is made up of **elements (actions)** connected by **paths** that determine how the process moves from one step to another.

Key Flow Elements:

Screen: To collect or display information to users

Create Records: To create new records

Update Records: To modify existing records

Get Records: To retrieve records based on criteria

Delete Records: To remove record

Decision: To branch the flow logic based on conditions

Assignment: To set or update variable values

Loop: To iterate over collections of data

Subflow: To call another flow inside the current flow

Example Use Cases for Flows

Scenario	Flow Type
Guided customer appointment booking screen	Screen Flow
Auto-send email when a job status changes to "Completed"	Record-Triggered Flow
Schedule monthly report data cleanup	Schedule-Triggered Flow
Automatically create a service job record when a new vehicle is added	Auto-launched Flow
Trigger actions when a platform event is published	Platform Event-Triggered Flow

Pro Tip:

Salesforce recommends migrating older **Workflow Rules** and **Process Builders** to **Flows** as part of their **Flow-First strategy**. Flows are now the primary tool for declarative automation in Salesforce.

#Apex trigger and apex handler:

An **Apex Trigger** is a piece of code that **executes automatically before or after a record is created, updated, deleted, or undeleted** in Salesforce.

It's part of Salesforce's server-side automation and is written in the **Apex programming language**.

Example Use Cases in a Garage Management System

Scenario	Trigger Event
Auto-create a Service Job record when a new Vehicle is added	after insert
Update Vehicle ' s Last Service Date when a Job Status changes to "Completed"	after update
Prevent deletion of a Job record if its status is "In Progress"	before delete

Apex Handler code:

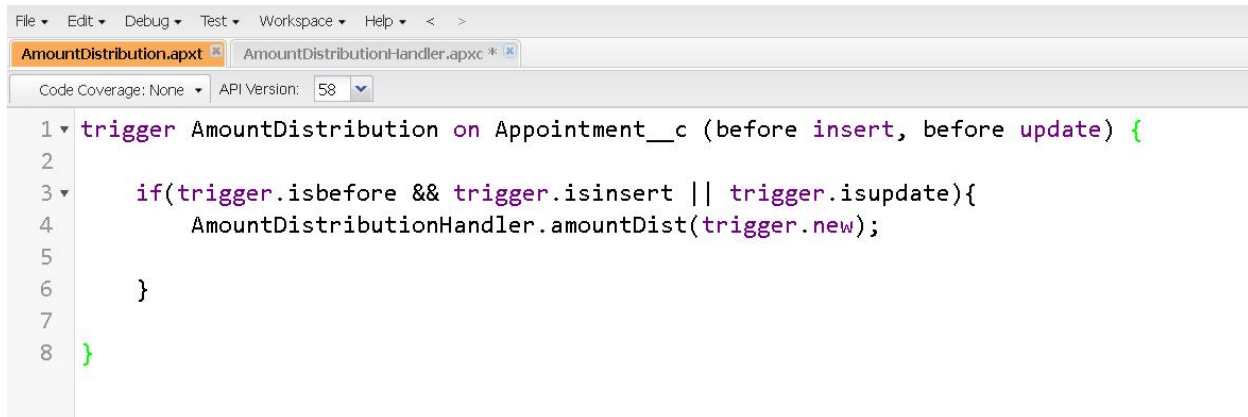
```
AmountDistribution.apxt | AmountDistributionHandler.apxc *
Code Coverage: None | API Version: 59 | Go To

1 public class AmountDistributionHandler {
2
3     public static void amountDist(list<Appointment__c> listApp){
4         list<Service_records__c> serList = new list<Service_records__c>();
5
6         for(Appointment__c app : listApp){
7             if(app.Maintenance_service__c == true && app.Repairs__c == true && app.Replacement_Parts__c == true){
8                 app.Service_Amount__c = 10000;
9             }
10            else if(app.Maintenance_service__c == true && app.Repairs__c == true){
11                app.Service_Amount__c = 5000;
12            }
13            else if(app.Maintenance_service__c == true && app.Replacement_Parts__c == true){
14                app.Service_Amount__c = 8000;
15            }
16            else if(app.Repairs__c == true && app.Replacement_Parts__c == true){
17                app.Service_Amount__c = 7000;
18            }
19            else if(app.Maintenance_service__c == true){
20                app.Service_Amount__c = 2000;
```

```
AmountDistribution.apxt | AmountDistributionHandler.apxc *
Code Coverage: None | API Version: 58 |

12     }
13     else if(app.Maintenance_service__c == true && app.Replacement_Parts__c == true){
14         app.Service_Amount__c = 8000;
15     }
16     else if(app.Repairs__c == true && app.Replacement_Parts__c == true){
17         app.Service_Amount__c = 7000;
18     }
19     else if(app.Maintenance_service__c == true){
20         app.Service_Amount__c = 2000;
21     }
22     else if(app.Repairs__c == true){
23         app.Service_Amount__c = 3000;
24     }
25     else if(app.Replacement_Parts__c == true){
26         app.Service_Amount__c = 5000;
27     }
28
29 }
30 }
31 }
```

Handler for the Appointment Object:



```
File Edit Debug Test Workspace Help < >
AmountDistribution.apxt AmountDistributionHandler.apxc *
Code Coverage: None API Version: 58
1 trigger AmountDistribution on Appointment__c (before insert, before update) {
2
3     if(trigger.isbefore && trigger.isinsert || trigger.isupdate){
4         AmountDistributionHandler.amountDist(trigger.new);
5     }
6 }
7
8 }
```

Example: Simple Apex Trigger

When a new Vehicle record is inserted, create a Service Job

```
trigger CreateJobOnVehicleInsert on Vehicle__c (after insert) {
    List<Service_Job__c> jobsToCreate = new List<Service_Job__c>();

    for (Vehicle__c v : Trigger.new) {
        Service_Job__c job = new Service_Job__c();
        job.Vehicle__c = v.Id;
        job.Status__c = 'Scheduled';
        jobsToCreate.add(job);
    }

    insert jobsToCreate;
}
```

A **Trigger Handler** is an Apex class used to hold the logic that would normally be inside a Trigger.

It helps follow the **Trigger Framework Pattern** — keeping Triggers clean, modular, and maintainable by moving logic to a separate Apex class.

Example: Trigger with a Handler Class

Trigger

```
trigger VehicleTrigger on Vehicle__c (after insert) {
    if (Trigger.isAfter && Trigger.isInsert) {
        VehicleTriggerHandler.createJobs(Trigger.new);
    }
}
```

Handler Class

```

public class VehicleTriggerHandler {
    public static void createJobs(List<Vehicle__c> newVehicles) {
        List<Service_Job__c> jobsToCreate = new List<Service_Job__c>();

        for (Vehicle__c v : newVehicles) {
            Service_Job__c job = new Service_Job__c();
            job.Vehicle__c = v.Id;
            job.Status__c = 'Scheduled';
            jobsToCreate.add(job);
        }

        insert jobsToCreate;
    }
}

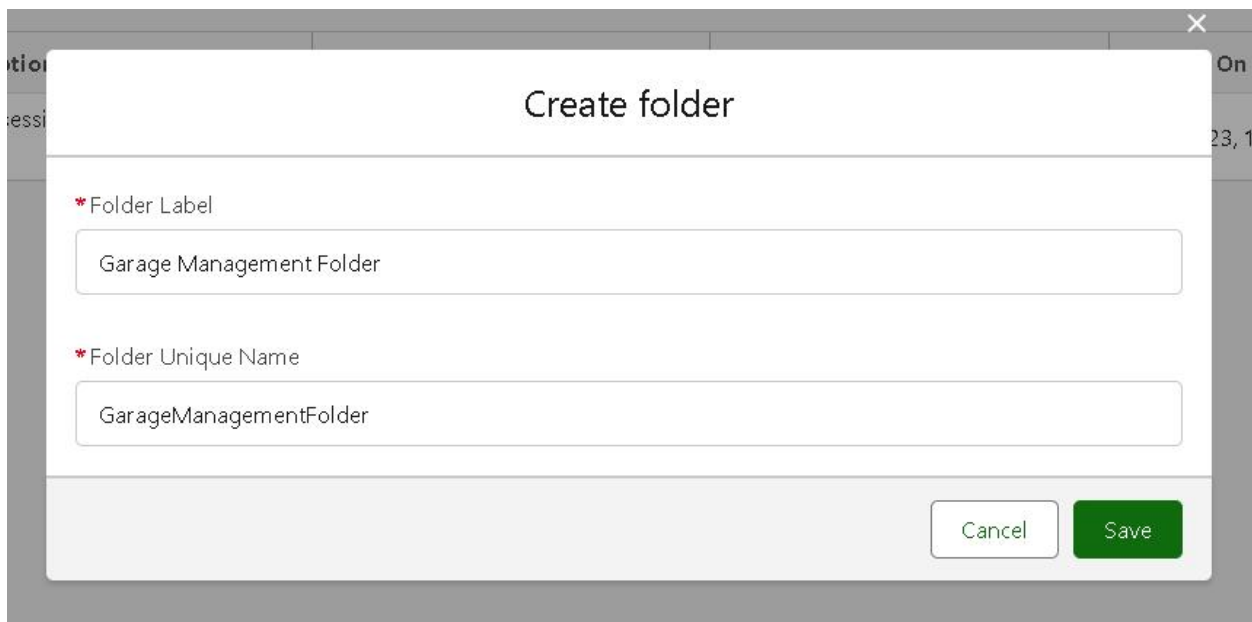
```

#Reports:

A **Report** in Salesforce is a **list of records that meet a set of defined criteria**. It lets you analyze and view your Salesforce data in different, meaningful formats and share insights across your organization.

Reports pull data from Salesforce objects (like **Accounts, Jobs, Vehicles, Invoices**) and present it in tables, summaries, or charts.

Creating a Report Folders:



The screenshot shows a 'Create folder' dialog box. It contains two required fields, both marked with a red asterisk. The first field, 'Folder Label', contains the text 'Garage Management Folder'. The second field, 'Folder Unique Name', contains the text 'GarageManagementFolder'. At the bottom right of the dialog are two buttons: 'Cancel' and 'Save'.

Types of Reports in Salesforce

Salesforce provides four main report formats. Let's break them down clearly:

1 Tabular Reports

What it is:

A simple list of records, similar to a spreadsheet.

Use case:

Lists of customers, invoices, vehicles due for service, or open jobs.

Example:

Show all Service Jobs created this month.

Limitations:

Can't create groupings or charts.

2 Summary Reports

What it is:

Like a tabular report but with the ability to **group records by rows** and create subtotals.

Use case:

Summarizing jobs by status, service revenue by month, or vehicles by make.

Example:

List of Service Jobs grouped by Job Status with total Service Amount.

3 Matrix Reports

What it is:

Allows grouping records **by rows and columns** — like a pivot table.

Use case:

Comparing totals and counts across multiple categories.

Example:

Number of service jobs by Job Status (rows) and Technician (columns).

4 Joined Reports

What it is:

Combines data from **multiple report types into a single report** with different views.

Use case:

Comparing related data sets, like Service Jobs vs. Invoices or Jobs vs. Parts Used.

Example:

One block for all completed Service Jobs, another for pending invoices, displayed side by side.

Example: Garage Management System Reports

Report Name	Type	Purpose
Open Service Jobs List	Tabular	View all open jobs
Service Jobs by Status	Summary	Group jobs by status to track workflow
Jobs by Technician and Status	Matrix	Track technician workloads and job completion
Service Jobs vs. Invoices Report	Joined	Compare completed jobs and associated invoices

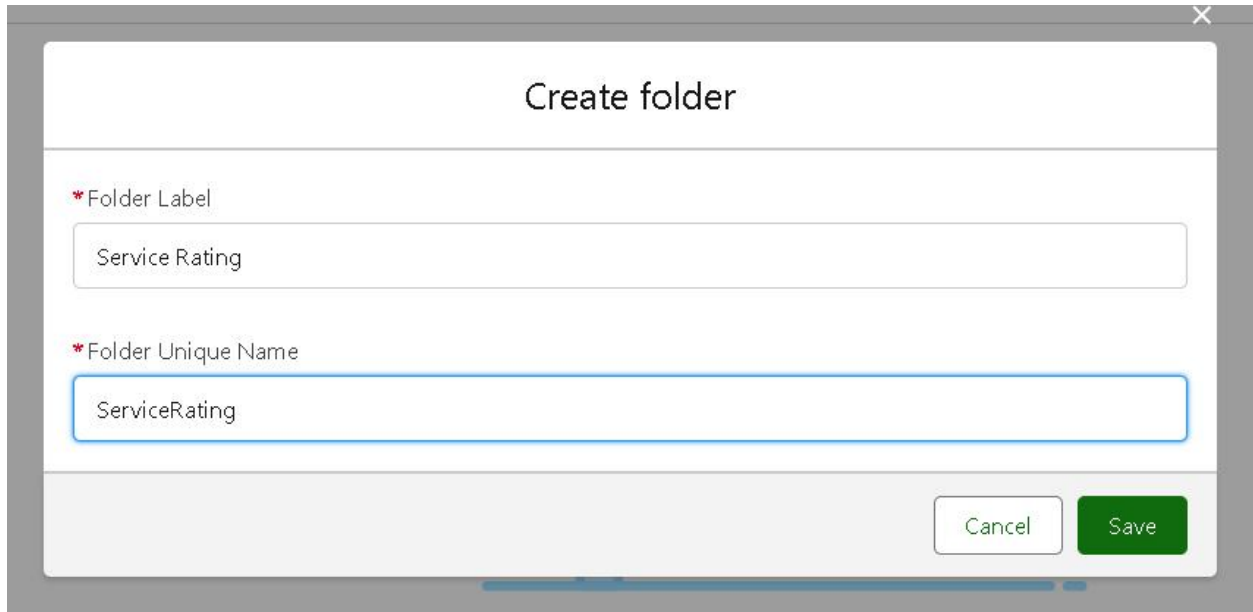
#Dashboards:

A **Dashboard** is a **visual display of key metrics and trends** for your business, presented using data from your Salesforce **reports**.

It helps you quickly understand real-time business conditions and make informed decisions based on current data.

Think of it as a control panel for your business data.

Creating Dashboard folder:



Create folder

* Folder Label

Service Rating

* Folder Unique Name

ServiceRating

Cancel Save

#User adoption and creating records in it:

User Adoption refers to how actively and effectively your users (technicians, receptionists, managers, etc.) engage with and use the **Garage Management System** in their daily work.

A system is only valuable if users actually use it consistently and correctly to:

- Log service jobs

- Update job statuses

- Create vehicle and customer records

- Issue invoices

- Track inventory and parts

- Generate reports

Without good user adoption, even a well-built system will fail to deliver its intended business value.

Creating Records in a Garage Management System

Creating records is the core of daily garage operations within the system. Key records typically include:

Key Record Types to Be Created

Record Type	Purpose
Customer	Stores customer contact and vehicle information
Vehicle	Captures vehicle details linked to a customer
Service Job	Tracks each vehicle service job and its status
Invoice	Records job-related charges and payments
Parts Used	Logs parts consumed in each job for inventory tracking

Example: Creating a Service Job Record

Required Fields:

Job Name / Number

Customer Name (lookup)Vehicle (lookup)

Job Description

Job Status (Open, In Progress, Completed)

Assigned Technician (lookup)

Service Date

Estimated Cost

Notes

Why it matters for adoption:

If this form is too complicated or slow to fill, technicians may avoid using it

Simplifying and training users to fill this record properly ensures better operational data and workflow trackin

Creating Records:

Garage Manageme...

Customer Details

Appointments

Service records

Billing details and feedback

Reports

Dashboards

Appointment

app-016

Appointment Name

app-016

Owner

Annapurna SmartBridge

Customer Details

Mac

* Appointment Date

13/11/2024

Maintenance service

☒

Repairs

☒

Replacement Parts

☐

Service Amount

* Vehicle number plate

TS30EU0443

Created By

Annapurna SmartBridge, 18/11/2024, 3:28 pm

Last Modified By

Annapurna SmartBridge, 18/11/2024, 3:28 pm

Cancel

Save

Related

Details

Service Record Name

ser-010

Owner

Annapurna SmartBridge

Appointment

[app-016](#)

Quality Check Status

☒

Service Status

Completed

service date

18/11/2024

Created By

Annapurna SmartBridge, 18/11/2024, 4:32 pm

Last Modified By

Annapurna SmartBridge, 18/11/2024, 4:34 pm

Record Creation Methods in a GMS

Method

Use Case

Method	Use Case
Manual via UI Form	Receptionist logs a new customer or job manually
Auto-Creation via Process/Flow	Auto-create a service job when a new vehicle is added
Bulk Import	Upload multiple customer or vehicle records at once
Mobile Interface	Technician updates job status or adds notes via mobile app

Linking User Adoption to Record Creation Success

Make record creation quick and intuitive
 Limit required fields to essentials only
 Train users on what data matters and why
 Use validation rules to enforce data quality without overcomplicating it
 Offer easy dashboards to show them the impact of their own records (e.g., completed job counts)

#Future Scope:

The future of garage management systems is revving up fast with the growth of the automobile industry and the rising demand for smarter, more efficient services. Here's how the road ahead is shaping up:

1. ****Integration with IoT and Smart Devices****
 - Real-time diagnostics from connected cars
 - Automatic service reminders using telematics
 - Smart inventory management with RFID and sensors
2. ****Mobile Accessibility & Customer Engagement****
 - User-friendly apps for booking, tracking repairs, and payments
 - Push notifications for maintenance schedules or discounts
 - Loyalty programs and digital service history for personalized experiences
3. ****AI-Powered Predictive Maintenance****
 - Analyzing past repair data to predict component failures
 - Intelligent scheduling and resource allocation

- Chatbots and virtual assistants for customer queries

4. ****Cloud-Based Infrastructure****

- Accessible from anywhere with centralized data
- Seamless integration with accounting, CRM, and invoicing tools
- Secure backup and scalability for multi-location garages

5. ****Automation and Workflow Optimization****

- Automated job cards, invoicing, and billing
- Workflow tracking for technicians and service bays
- Reducing human errors and turnaround time

6. ****Data-Driven Business Insights****

- Sales and service analytics to guide growth strategies
- Performance metrics of employees and services
- Forecasting demand trends and inventory needs

7. ****EV and Hybrid Vehicle Support****

- Specialized modules for handling EV diagnostics and parts
- Training management systems for EV-certified technicians

this is a high-torque sector with plenty of momentum. Want help brainstorming features or building a roadmap for a GMS project? Let's roll up our sleeves.

Conclusion:

A **Garage Management System** modernizes and simplifies the way workshops manage their day-to-day business, improves operational control, enhances customer service, and empowers decision-makers with real-time, data-backed insights.

It's an essential solution for automotive service businesses aiming for efficiency, transparency, and growth.

The evolution of the automobile industry demands solutions that are not just efficient, but intelligent, adaptive, and deeply customer-focused. A well-crafted Garage Management System stands at the forefront of this transformation—streamlining operations, enhancing customer satisfaction, and unlocking data-driven decision-making.

From integrating with IoT devices and supporting electric vehicles, to harnessing AI for predictive maintenance and scaling services through the cloud, a GMS is no longer just a tool—it's the digital backbone of a modern auto service center.

As garages gear up for the future, embracing a comprehensive GMS isn't just an upgrade; it's a strategic move toward excellence, sustainability, and long-term growth. The road ahead is digital, and with the right system in place, it's a smooth ride all the way.

A garage management system streamlines and automates the operations of vehicle service centers, improving both efficiency and customer satisfaction. In conclusion, such systems:

- **Centralize data** related to customers, vehicles, inventory, and service history.
- **Enhance workflow** by organizing job cards, assigning tasks to technicians, and tracking progress.
- **Improve customer relations** through timely updates, reminders, and transparent billing.
- **Boost profitability** by reducing manual errors, optimizing stock levels, and generating insightful reports.

Whether you're running a small workshop or a large service chain, a well-designed garage management system is a powerful tool to drive productivity and build trust with clients. Want help creating a report or presentation around this? I've got your back.

A garage management system, when effectively implemented, offers a comprehensive solution to the multifaceted challenges faced by automotive repair and maintenance businesses. The core value proposition of such a system lies in its ability to streamline operations, enhance customer satisfaction, and ultimately, drive profitability.

Key areas of impact and conclusion analysis:

* **Operational Efficiency:** The most immediate and tangible benefit is the significant improvement in operational efficiency. By centralizing data related to appointments, vehicle history, parts inventory, and technician schedules, the system eliminates manual processes, reduces errors, and minimizes downtime. This leads to faster service delivery, better resource allocation, and a more organized workflow. For example, automated scheduling prevents double-bookings, and real-time inventory tracking ensures parts availability, avoiding delays.

* **Customer Relationship Management (CRM):** A robust garage management system fosters stronger customer relationships. It provides a detailed history of each vehicle's service, allowing for personalized recommendations and proactive maintenance reminders. This not only improves customer loyalty but also presents opportunities for

upselling and cross-selling relevant services. Features like online booking and automated communication (SMS updates, email reminders) significantly enhance the customer experience.

- * **Financial Management and Profitability:** The system provides critical insights into the financial health of the garage. It automates invoicing, tracks payments, and generates reports on revenue, expenses, and profitability. By identifying high-performing services, frequently used parts, and technician productivity, managers can make data-driven decisions to optimize pricing, control costs, and increase overall profitability. Integration with accounting software further simplifies financial reconciliation.

- * **Inventory Control:** Effective inventory management is crucial for any garage. A dedicated system helps in tracking parts from procurement to installation, minimizing stockouts and overstocking. This reduces capital tied up in inventory, prevents losses due to obsolete parts, and ensures that technicians have the necessary components when needed, directly impacting service speed and customer satisfaction.

- * **Technician Productivity:** By providing technicians with easy access to repair manuals, vehicle history, and job assignments, the system empowers them to work more efficiently. Digital job cards and real-time progress tracking allow managers to monitor workload and identify training needs or areas for improvement, ultimately boosting overall technician productivity and quality of work.

- * **Data-Driven Decision Making:** Perhaps one of the most powerful aspects of a modern garage management system is its ability to generate actionable insights from the collected data. Through various reports and analytics, owners and managers can gain a clear understanding of business performance, identify trends, anticipate future needs, and make informed strategic decisions regarding marketing, staffing, and service offerings.

In conclusion, a garage management system is no longer a luxury but a strategic imperative for modern automotive service businesses. Its comprehensive capabilities in streamlining operations, enhancing customer satisfaction, optimizing financial performance, and providing valuable data insights contribute to a more efficient, profitable, and customer-centric business model. The investment in such a system typically yields a significant return by improving productivity, reducing operational costs, and ultimately securing a competitive edge in the dynamic automotive service industry.