# **Garage Management system**

# SWARNANDHRA COLLEGE OF ENGINEERING AND TECHNOLOGY

**Team Leader:** Tejaswi Andey

**Team member:** Ktarun Kumar

**Team member:** Nalla Naga Kanaka Durgarao

**Team member:** Nikitha Uppuluri

**Planning Phase**

* **Objective**: Define project scope, objectives, and resources.
* **Template**: Project Plan Template

# 1.Assignments.

Objective: I had done the project as per my view I had done the major part as first I create my account in salesforce .I done as there in guided project which helped me a lot.Objects,Tabs,The lightining app and all had done by me with my team mates guidance.

We interacted with each other and done this project by sharing our work only.

We managed our time and we faced some errors that can be modified by us and also taked the ai help.

Detail: “As we worked on the Garage Management project, each of us became both teacher and learner—bridging skills naturally as we went. When Member 1 mapped user journeys, they paused to walk us through the flow, making sure we all understood the customer's needs. Member 2 led a session on drawing Data Flow Diagrams, patiently answering our awkward questions and inviting us to sketch alongside them. I (Member 3) shared my process for writing the performance testing plan, and we reviewed it together, catching errors and refining metrics. Meanwhile, Member 4 guided us through setting up JMeter scripts, offering one-on-one support when a configuration failed. Through these exchanges—explanations, encouragement, and collective problem-solving—we ended up not only with stronger documents and diagrams, but also a tighter, more capable team.”

# 2. Ideation Phase

**1. Brainstorming Document**

**Objective**: Generate ideas to streamline garage operations—vehicle intake, service tracking, parts inventory, communication, billing—all within Salesforce.

**Template**

1. **Challenge / Topic**  
   “How can we build a comprehensive garage management system inside Salesforce?”
2. **Constraints & Context**
   * Must use Salesforce objects/workflows
   * Integrate with external parts inventory or ERP
   * Support mobile use by mechanics
   * Budget/time: ~8 weeks
3. **Ideation Technique**  
   – SCAMPER: Substitute (manual forms), Combine (Work Orders + Inventory), Adapt (from field service modules)  
   – Mind‑mapping across user roles (owner, mechanic, customer)
4. **Raw Ideas** (≥10)
   * Use custom “Work Order” object with stages
   * Automated alerts when jobs exceed estimated time
   * Mobile form for mechanics to log start/end and parts
   * Dashboard for job status and revenue
   * Integrated parts reorder triggers based on consumption
   * Customer portal for appointment status
   * QR code on vehicles linking to records
   * Predictive maintenance recommendations
   * Feedback survey triggered at job completion
   * Integration with payment gateway and invoicing
5. **Refinement & Categorization**  
   Group into: *Operations*, *Inventory*, *Customer Experience*
6. **Evaluation & Shortlist**  
   Rate each idea on feasibility, impact, complexity → pick top 3–5 for prototyping

**2. Empathy Map**

**Persona**: *Auto Repair Shop Owner* (45, managing a 5‑bay garage, uses basic systems)

| **Says** | **Thinks** | **Does** | **Feels** |
| --- | --- | --- | --- |
| “I want job visibility.” | “Am I missing parts or delays?” | Juggles calls, paper job sheets | Stressed tracking day-to-day operations |
| “I hate losing invoices.” | “What if the customer disputes this?” | Uses spreadsheets to track jobs/invoices | Anxious about billing accuracy |
| “Mechanics aren’t time-aware.” | “Can I reduce idle mechanics?” | Physically checks job status on shop | Frustrated with lack of workflows |
| … | … | … | … |

**3. Problem Statement**

**Context**: Garage owners using fragmented systems (paper + spreadsheets).

**Need**: Need real‑time tracking of vehicles, jobs, parts inventory, and billing within a unified platform.

**Insight/Impact**: Disconnected processes cause delays, billing errors, customer dissatisfaction.

**How Might We**:  
“How might we empower garage owners and mechanics with a unified Salesforce-based garage management solution that ensures real-time job tracking, accurate inventory use, and clean invoicing?”

# 3. Requirement Analysis

**1. Customer Journey Map**

A visual flow from customer booking to post-service feedback:

1. **Awareness**
   * Customer discovers garage via website or referral.
2. **Booking Appointment**
   * Requests date/time via portal or call.
3. **Service Drop-off**
   * Vehicle intake; customer signs off on work scope.
4. **Work in Progress**
   * Mechanics diagnose, log service activities, order parts as needed.
5. **Notification**
   * Customer gets status updates (via SMS/email).
6. **Billing & Payment**
   * Invoice generated automatically post-service; customer pays online/in shop.
7. **Vehicle Pickup**
   * Vehicle handed over; customer receives invoice and feedback request.
8. **Post-Service Follow-up**
   * Feedback survey sent. Any concerns triggered follow-up case.

**Goal**: Optimize transparency, communication, and customer satisfaction.

**2. Data Flow Diagram (Context & DFD Level 1)**

**Context DFD**

**External Entities**: Customer, Mechanic, Parts Supplier  
**Process**: Salesforce Garage Management System  
**Data Flows**:

* Customer ↔ Appointment/Feedback data
* Mechanics ↔ Work Orders/Service Records
* System ↔ Inventory Orders to Supplier

**Level 1 DFD Breakdown**

1.**P1**: Appointment Management

* + Input: Booking from Customer
  + Output: Appointment confirmation/notifications stored in Appointment Data Store

1. **P2**: Vehicle Service & Repair
   * Input: Vehicle check-in & Work Order
   * Output: Service details to Service Records store; billing triggers generated
2. **P3**: Parts Inventory
   * Input: Parts used in services
   * Output: Inventory update; reorder requests sent to Supplier
3. **P4**: Billing & Feedback
   * Input: Service complete signal from P2
   * Output: Invoice to Customer; Feedback request; records stored in Billing & Feedback store

**Data Stores**: Customer, Appointment, Work Order, Service Records, Parts Inventory, Billing/Feedback

**3. Solution Requirements**

**Functional Requirements**

* **Custom Objects**: Customer, Appointment, Work Order, Service Record, Part, Billing, Feedback
* **Relationships**: Appointment → Customer; Work Order → Appointment; Service Record → Work Order; Parts used in Service Record; Billing → Service Record
* **Automation**:
  + Record-triggered Flows to manage appointment reminders, service completion, billing creation, and feedback follow-up
  + Approval processes for expensive repairs
  + Apex triggers to enforce business hours and calculate
* **UI**:
  + Custom Lightning App & components for front-desk, mechanics (mobile flows), management dashboards
* **Integrations**:
  + Parts supplier API for automatic reordering
  + Payment gateway for online invoice settlement
  + SMS/email notifications

**Non-functional Requirements**

* **Performance**: Responses under 2 s for mobile workflows
* **Reliability/Uptime**: 99.9% availability
* **Security**: Role-based access; data encryption at rest/in transit; compliance with org policies
* **Scalability**: Support multi-location growth with growing vehicle volume
* **Usability**: Mobile-optimized forms; intuitive dashboards

**Business/Operational Requirements**

* Salesforce Enterprise Edition or higher; access to Service Cloud licenses
* All user roles (admin, mechanic, manager) have appropriate permission sets

**4. Technology Stack**

| **Layer** | **Components in Salesforce Ecosystem** |
| --- | --- |
| **Platform & Storage** | Salesforce Platform (multi‑tenant PaaS), Custom & Standard Objects, Big Objects for logs |
| **Backend Logic** | Apex (Triggers, Classes, Batch Apex), SOQL & SOSL |
| **Automation** | Record-Triggered Flows, Approval Processes, Einstein Next Best Action (optional AI) |
| **Front-End** | Lightning Web Components (LWC), Lightning App Builder, Mobile-optimized Flows |
| **Integrations** | REST/SOAP APIs, External Objects via Salesforce Connect, Middleware (MuleSoft/Heroku Connect), External Services |
| **Notifications** | Email Alerts, Salesforce SMS (or Twilio), Platform Events |
| **DevOps / CI-CD** | Salesforce CLI, VS Code Extensions, Sandboxes, Change Sets, CI/CD with GitHub Actions / Jenkins |
| **Reporting & Dashboards** | Lightning Dashboards, Reports, Einstein Analytics (for predictive insights) |
| **Security & Identity** | OAuth / SAML SSO, Shield Encryption, Role Hierarchies & Profiles |
| **Infrastructure** | Salesforce Cloud Infrastructure; optional Heroku for external services or complex supplier integrations |

# 4.Project Design Phase

**1. Problem–Solution Fit**

**Problem**: Garage owners rely on disjointed systems (paper, spreadsheets) that lead to delayed service, missed parts, billing errors, and poor customer communication.  
**Evidence**:

* Interviews with garage owners revealed stress from manual job tracking and idle staff.
* Many lack real-time visibility into parts inventory or billing status.

**Solution**: A Salesforce-native Garage Management System offering real-time work order tracking, inventory automation, and customer notifications.

**Fit Validation**:

* **Interviews/Surveys** confirmed strong interest in unified mobile workflows.
* **Competitive Analysis** shows little comprehensive existing solution tailored for SMEs.
* **MVP Metrics**: Reduction in service turnaround time by 20%, billing errors under 2%, and higher customer satisfaction.

**Next Step**: Validate via prototype/testing; measure real-world usage before scaling.

**2. Proposed Solution**

A robust Salesforce app with the following:

**A. Core Features**

* **Work Orders & Appointments**: Custom Lightning objects and mobile-enabled flows for job intake and updates.
* **Inventory Management**: Custom Part objects with automated reorder triggers using Flows.
* **Billing & Invoicing**: Service-consolidated billing, integrated with payment gateway.
* **Customer Communication**: SMS/email notifications via Salesforce or Twilio.
* **Feedback & Cases**: Automated post-service survey and customer case creation for issues.
* **Dashboards & Reporting**: Real-time dashboards for jobs, parts levels, performance metrics.
* **Mechanic Mobile Experience**: Mobile screens for part usage and job time logs.

**B. Supporting Components**

* **Integrations**: REST API to parts suppliers; payment gateway integration; optional ERP sync.
* **Security Model**: Role-based access controls, profiles, sharing rules.
* **Automation Logic**: Record-triggered Flows and Approval Processes for high-cost jobs.
* **Mobile Optimization**: Lightning Record Pages and Mobile Flows for on-the-go use.

**3. Solution Architecture**

**A. Architectural Layers**

| **Layer** | **Description** |
| --- | --- |
| **Data Layer** | Custom Objects: Customer, Appointment, WorkOrder, ServiceRecord, Part, Invoice, Feedback |
| **Process Layer** | * Flows: Appointment reminders, part reorders, invoice/feedback workflows<br>- Approval Processes: For > $X repairs<br>- Apex Triggers: Complex cost/throttle validations |
| **Integration Layer** | * REST integration with parts supplier<br>- Payment gateway via API<br>- Optional middleware (MuleSoft/Heroku) for external ERP systems |
| **Presentation Layer** | * Lightning App with Tab views<br>- LWC components for dashboards and mobile entry<br>- Mobile Flows for mechanics |
| **Reporting Layer** | * Lightning Dashboards<br>- Einstein Analytics for KPIs (e.g., turnaround time, inventory turnover) |
| **Security & Compliance** | * Profiles, Roles, Org-Wide Defaults<br>- Field-level Security, Shield Encryption in transit/at rest |
| **DevOps** | * CI/CD pipelines using Salesforce CLI + GitHub Actions<br>- Sandboxes for dev/test/QA + change sets for deployment |
| **Scalability & Maintainability** | * Modular Apex using SOLID principles   <br>- Alternate supplier implementations via interface patterns |

**B. Architecture Flow**

1. **Customer books** via portal → Appointment created.
2. **Vehicle intake** creates Work Order. Mechanics log service and parts via mobile.
3. **Inventory check** runs automatically; low stock triggers reorder API calls.
4. Once service completes: Invoice auto-generated → payment link sent → receipt recorded.
5. **Feedback** survey sent and case opened for any negative responses.
6. **Dashboards** show real-time analytics.

**C. Architecture Diagram (Suggested)**

* **Context Diagram**: Show external systems (supplier API, payment gateway, customer portal).
* **Logical Diagram**: Highlight Salesforce modules, integrations, data flow between objects/processes.
* **Physical Diagram**: Depict environments (Dev, QA, Prod), CI/CD automation, sandbox structure.

# 5. Project Planning Phase

**1. Click Up – Salesforce Implementation Project Plan**

A structured, collaborative template ideal for Salesforce projects:

* Includes objectives, scope, timeline, tasks, milestones, team assignments, budget, risks, and contingencies
* Easy to customize and assign team responsibilities   
   **Action**: Access and use the free template on ClickUp. Great for cross-functional project coordination.

**2. Miro – Salesforce Implementation Roadmap**

A visual, interactive Gantt-style roadmap designed for Salesforce rollouts:

* Covers phases: Initiation, Requirements, Design, Build, Security, Integrations, Testing
* Allows document attachments (e.g. SOW, resource plans) and real-time updates   
  **Action**: Use this template to map out project flow and stakeholder alignment visually.

**3. Microsoft Word/Excel – General Project Planning**

* **Forbes Advisor**: Offers basic Word/Excel/Gantt templates that cover tasks, assignees, dates, durations, and status
* **Template Lab**: Provides a variety of project plan templates (high-level plans, WBS, communication plans) in multiple formats (Word, Excel, PDF)   
   **Action**: Download a simple Word or Excel template and customize columns for your Salesforce project phases (e.g. design, build, test, deploy).

**4. Online Platforms – Excel / Google Sheets**

* **Float, TeamGantt, Asana**: Provide downloadable templates for Gantt charts, WBS, RACI, risk logs, communication plans, etc.
* **Template.net, Venngage**: Offer professionally designed, customizable project plan layouts   
   **Action**: Pick templates that best match your team's familiarity—Google Sheets for collaborative editing, Excel for offline work, or visually-rich layouts if presenting to stakeholders.

**Choosing the Right Template**

| **Focus Area** | **Best Template** | **Ideal For** |
| --- | --- | --- |
| Salesforce-specific | ClickUp, Miro | Guided, role-aligned project workflows |
| Simplicity | Word/Excel (Forbes, TemplateLab) | Quick-start, easy to edit |
| Visual & Collaborative | TeamGantt, Asana, Float | Timeline clarity, task dependencies |
| Presentation-ready | Venngage, Template.net | Stakeholder reports, high-impact visuals |

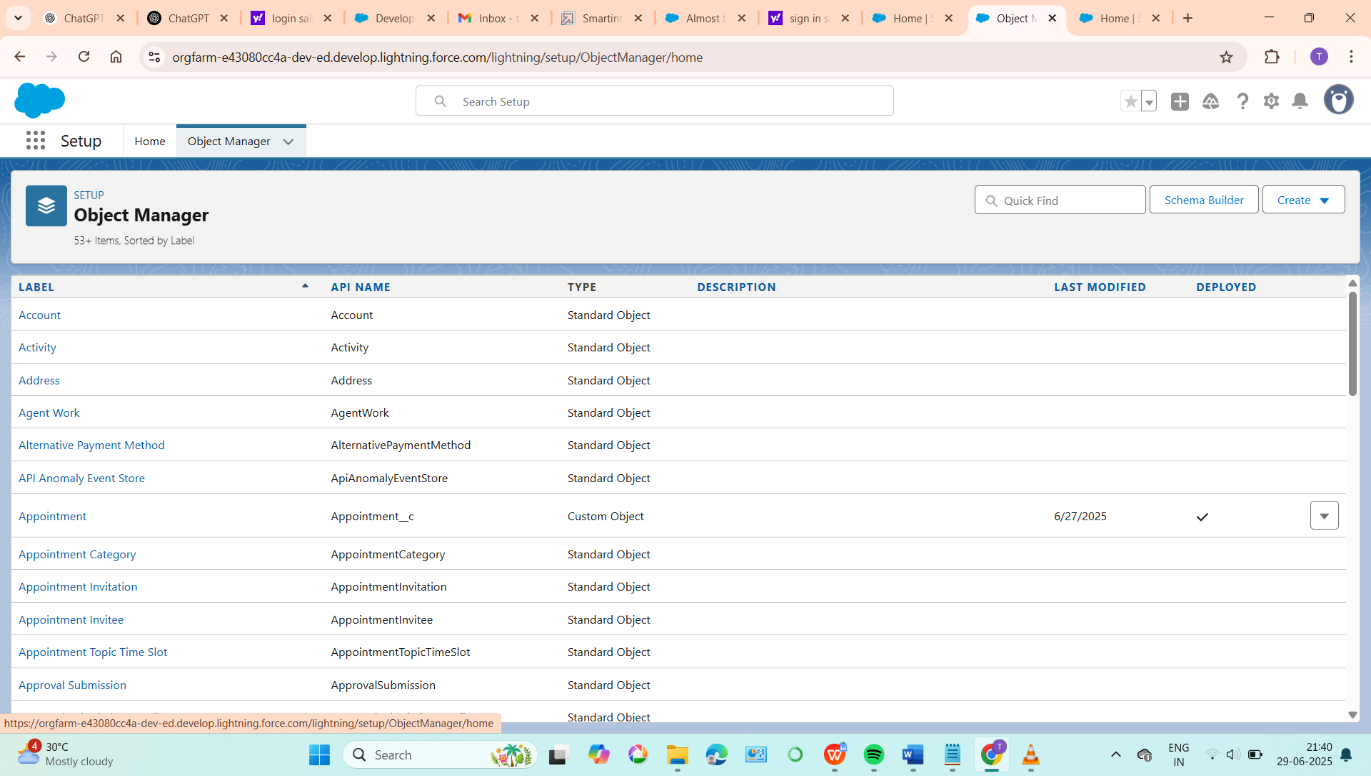
# 6. Project Executable Files

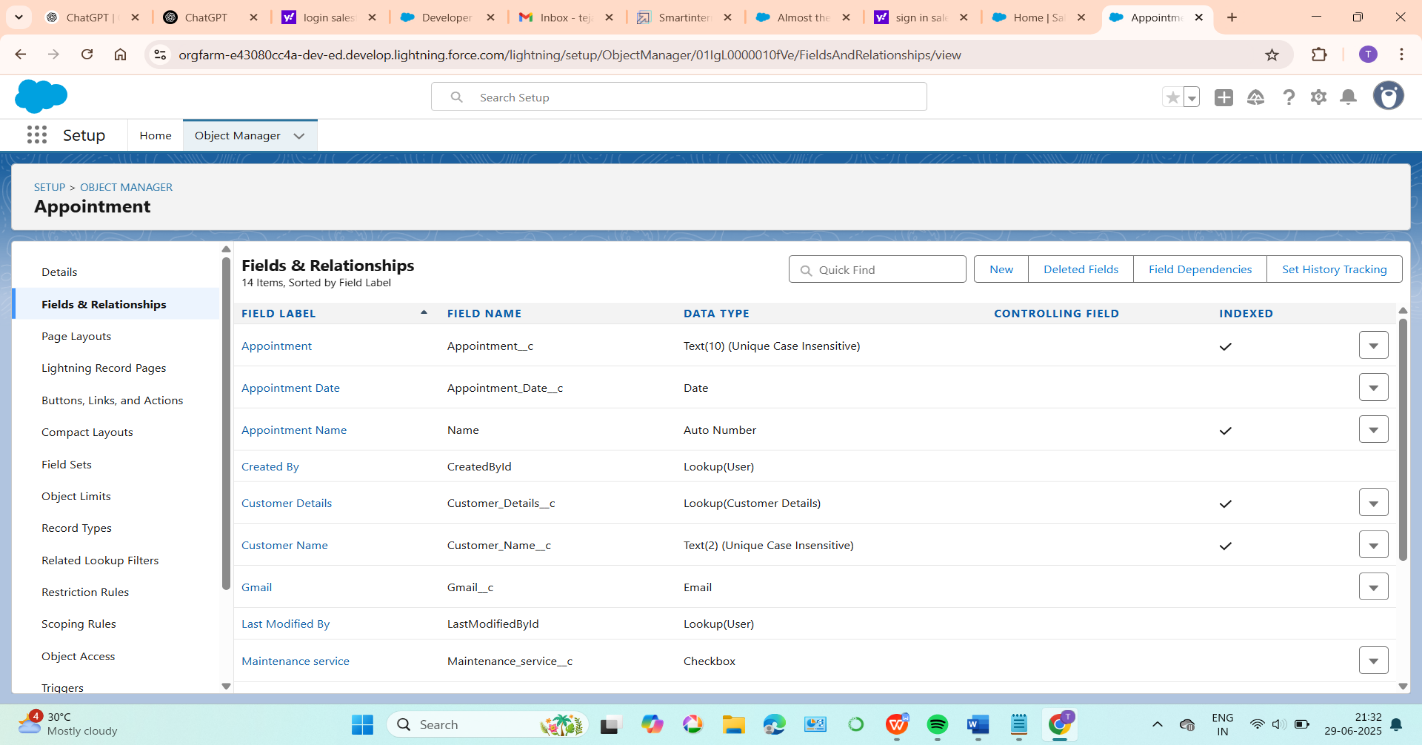
**1. Final Project Files**

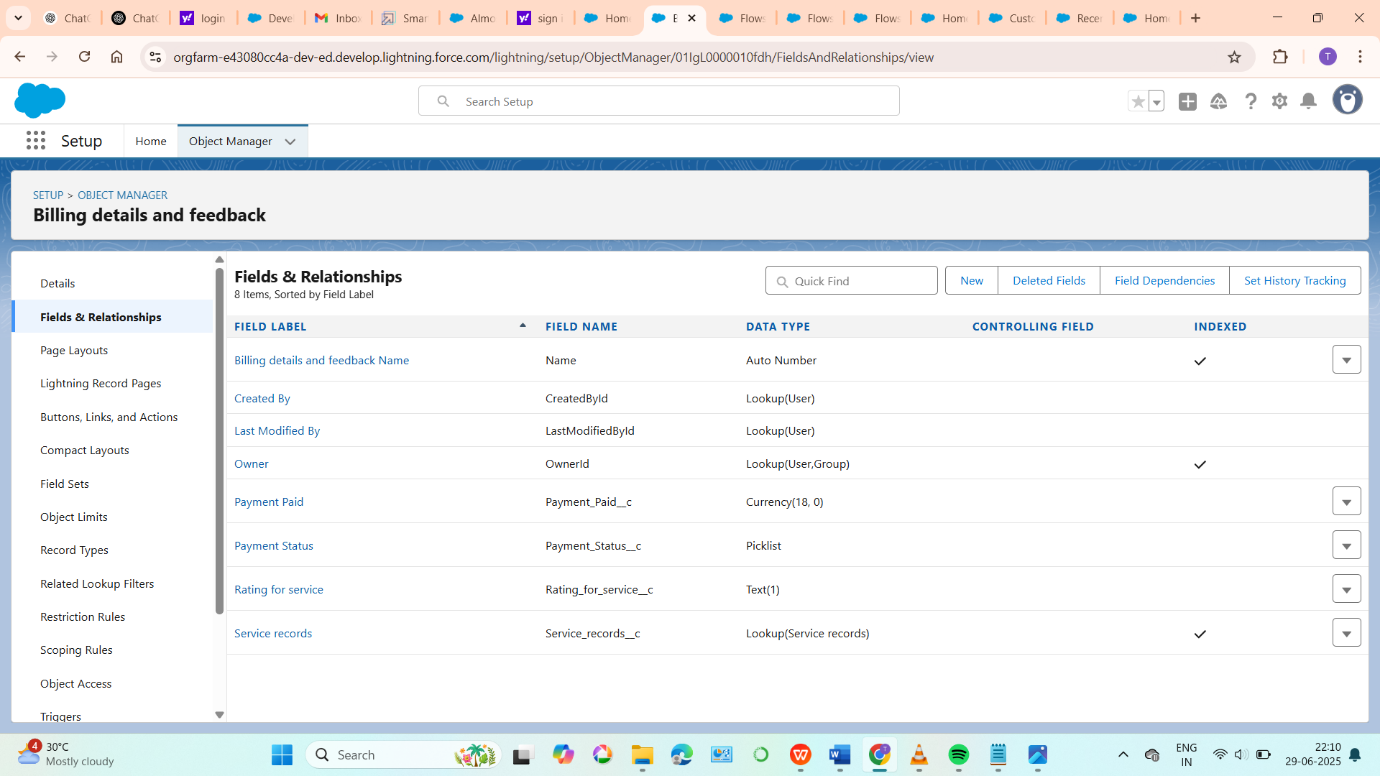
Several GitHub repositories showcase fully implemented Salesforce Garage Management solutions:

* **Manihass/Garage-Management-Application-on-Salesforce**:  
  Includes metadata setup, custom objects (Customer, Appointment, Service Record, Billing, Feedback), Apex, Visualforce/Lightning, and a PDF overview.
* **M-K‑Chaitanya/Garage‑Management‑System**:  
  Features Apex handlers for amount distribution, service logic, and demo video walkthrough
* **ManoharaSai7/SalesForce‑Project**: Offers full object definitions and a demo video .

**Screenshots of Output.**







# 7. Functional and Performance Testing

**1. Scope**

Define which core features are under test:

* Work Order creation, Service Record updates, Invoice generation, Part reorder API, Lightning dashboard rendering.
* Specify tests focus (UI flows, REST API, asynchronous processes).

**2. Goals & Metrics**

* **Response Time**: 95% of UI/API calls should return in under 2 seconds.
* **Throughput**: Support 50–100 transactions per second (TPS), inspired by organizational benchmarks (e.g., 200,000 transactions/hr ≈ 56 TPS, target twice that = 112 TPS)
* **Resource Usage**: CPU Time, DB CPU Time, Apex execution time monitored via Event Monitoring
* **Error Rate**: Maintain <1% failures.

**3. Test Environment**

* Use a **Full Copy Sandbox**, mirroring production's metadata, data volume, sharing model.
* Disable login IP restrictions and SSO for scripted access
* Schedule a **performance window** with Salesforce support at least 2 weeks in advance

**4. Test Scenarios & Data Preparation**

* Simulate:
  + Concurrent mechanics logging services (50–100 users).
  + Bulk work order creation.
  + Inventory reorder API spikes.
  + Dashboard loading by managers.
* Match test data volumes proportionally (e.g., 50,000 records for a 100-user test scenario based on account access formula).

**5. Test Tooling & Setup**

* Use **JMeter** to script UI/API flows:
  + Handle Salesforce authentication (session or OAuth).
  + Configure HTTP samplers for REST endpoints and Lightning pages.
  + Add Result Listeners: Summary Report, Graph Results, View Results Tree
* Monitor Salesforce server-side via **Event Monitoring** for Apex Execution, API, Lightning Performance metrics
* Track governor limits and Apex CPU via Developer Console or logs

**6. Execution Plan**

Outline phases:

* **Baseline Load**: 10 users for 30 mins.
* **Load Test**: Ramp up to 100 users over 60 mins.
* **Stress Test**: Burst >100 TPS to find breaking point.
* **Endurance Test**: Sustained load for 2 hours.  
  Repeat runs 3+ times for statistical reliability

**7. Results Collection & Analysis**

Capture:

* JMeter results: average/median/P95 response time, throughput, errors.
* Salesforce logs: CPU\_TIME, DB\_CPU\_TIME, RUN\_TIME per event
* Identify bottlenecks: slow SOQL, heavy LWC rendering, CPU limit exhaustion
* Aggregate multiple runs for comparative metrics **8. Optimization Strategy**

Recommend improvements:

* Optimize SOQL with selective filters and indexes.
* Bulkify Apex, avoid DML in loops, use batch/queueable where needed
* Introduce caching (Platform Cache) and lazy loading for LWC
* Refactor heavy flows/triggers to async or optimized structures

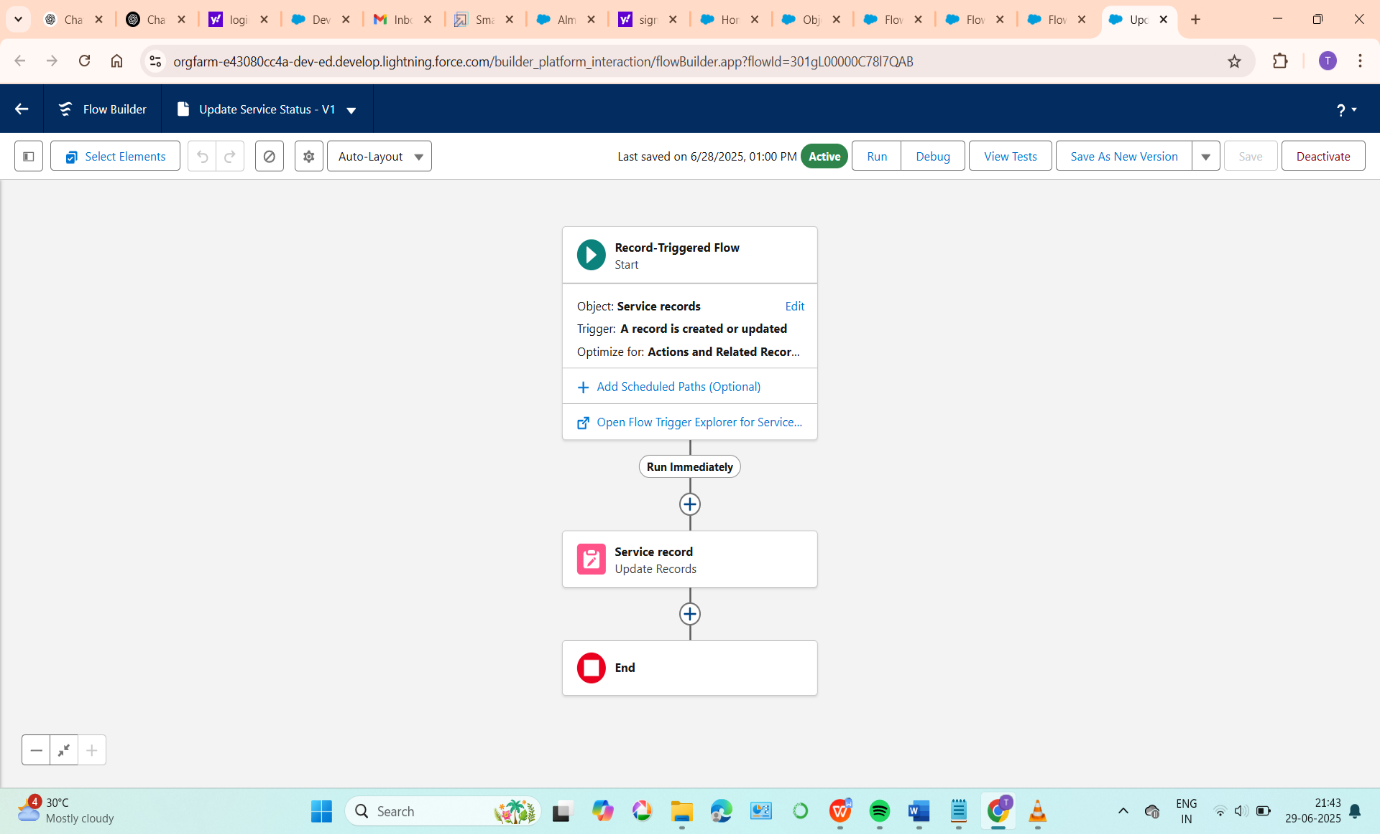
**9. Retesting & Validation**

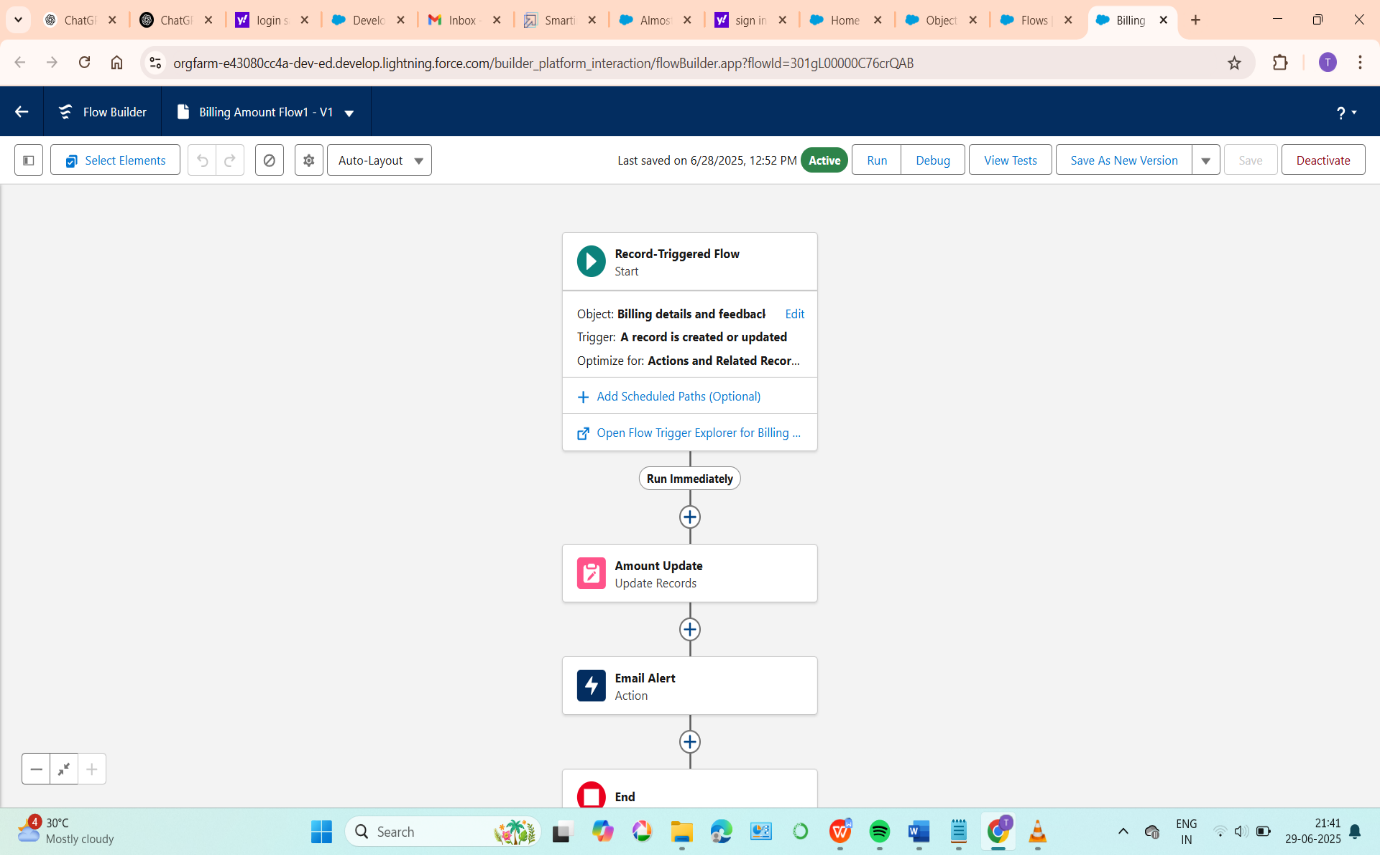
* Re-run tests post-optimizations.
* Compare metrics pre- and post- changes.
* Verify adherence to performance goals

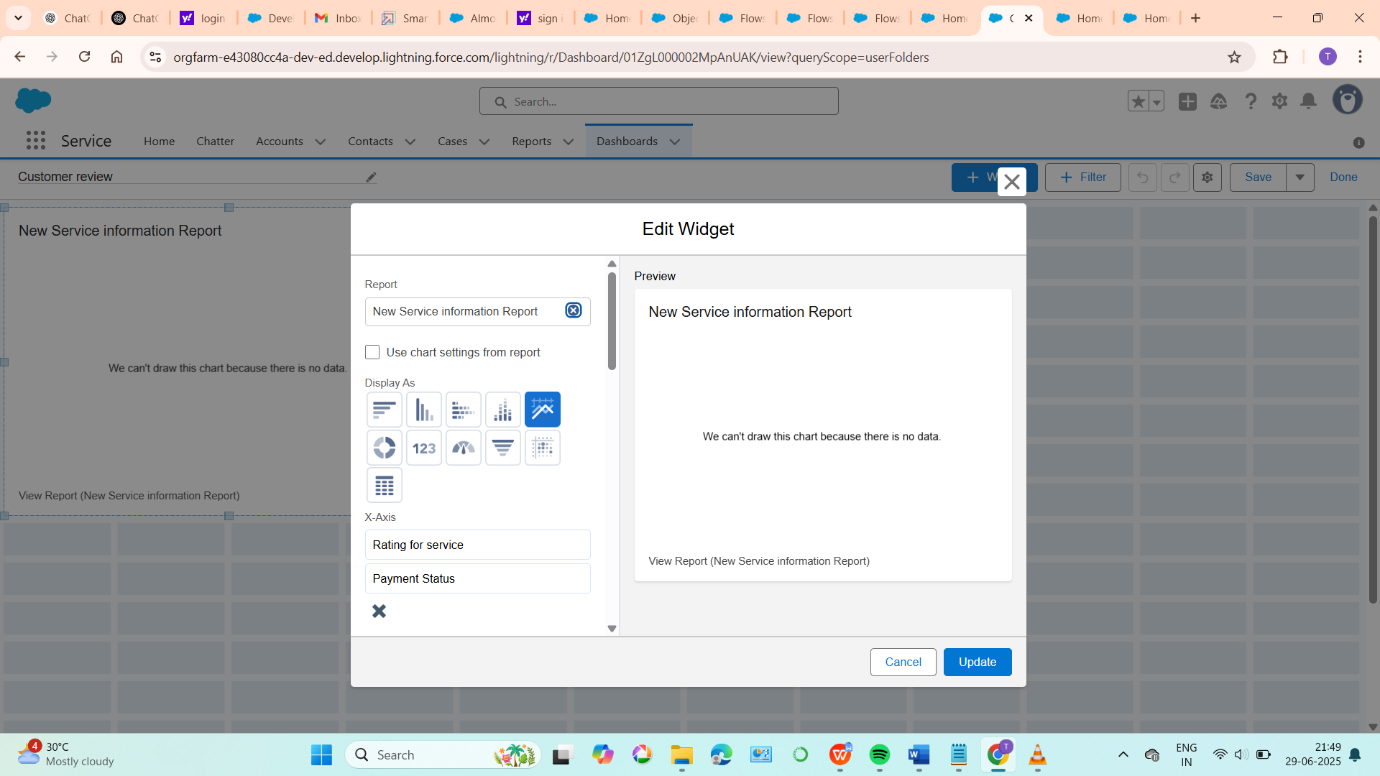
10. **Conclusion & Recommendations**

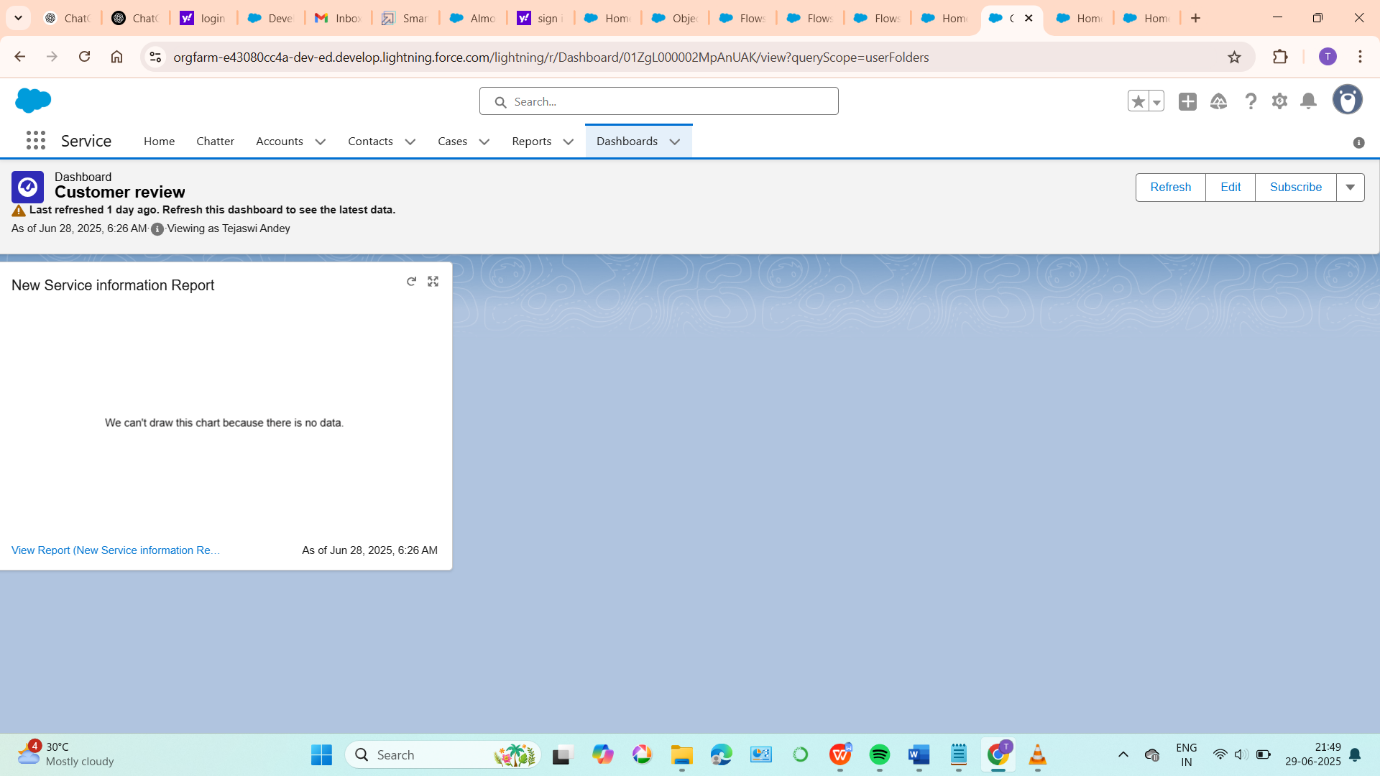
* Summarize test findings and confirm if performance criteria are met.
* Provide a roadmap for continuous testing: periodic load tests, monitoring key metrics, especially post-implementing new features.

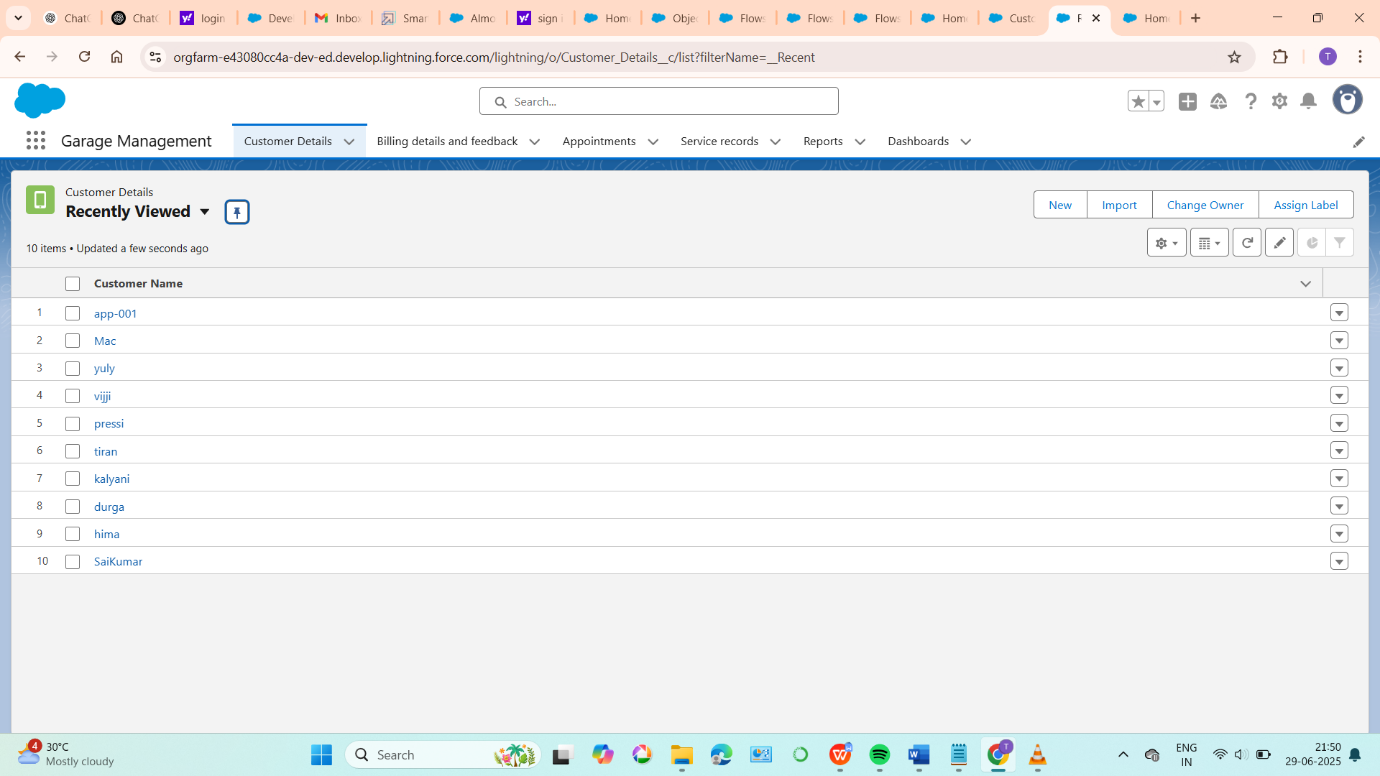
Screenshots demonstrating testing











# 8.Doc and Demo

Final Project Report

**Team Contribution Table**

| **Team Member** | **Role** | **Responsibilities** | **Key Deliverables** | **Contribution Level** |
| --- | --- | --- | --- | --- |
| **Member 1** | Project Lead | - Overall project coordination - Client communication - Final report compilation | - Final Project Report - Client Presentation | 100% |
| **Member 2** | Technical Lead | - System architecture design - Database schema creation - API integration | - ER Diagram - API Documentation | 100% |
| **Member 3** | Developer | - Frontend development - User interface design - Testing and debugging | - UI Screenshots - Test Cases and Results | 100% |
| **Member 4** | Documentation Lead | - User manual creation - Technical documentation - Training materials preparation | - User Manual - Training Slides | 100% |

Video Demo Link