Salesforce Project Documentation: EV Charging CRM

Project Documentation Overview

Project documentation in a Salesforce CRM implementation serves as a comprehensive record of the application's purpose, design, development, and deployment. It ensures that business requirements are clearly defined and aligned with the system's functionality. This document acts as a blueprint for developers and admins, guiding consistent development and future scalability. It supports user training, facilitates troubleshooting, and ensures smooth maintenance by detailing every component and its logic.

Project Overview

This project is an EV Charging CRM built on the Salesforce platform, designed to efficiently manage a growing network of Electric Vehicle (EV) charging stations. The primary challenge this CRM solves is the lack of a centralized system for tracking station status, managing real-time slot availability, handling customer bookings, and processing payments. The CRM provides a comprehensive solution for customers to find and reserve charging times, for station managers to monitor their assets, and for admins to oversee the entire operation and generate key reports.

Objectives

The main goal of building this CRM is to streamline all business processes related to the EV charging network and enhance the end-user experience. The project links these goals to business value by:

- **Improving customer management** by tracking customer vehicles, booking history, and support cases.
- **Streamlining the booking process** through an intuitive interface and real-time availability tracking.
- **Providing analytics** on station utilization, revenue, and customer behavior to drive business decisions.

Phase 1: Problem Understanding & Industry Analysis

Business Problem

The primary challenges addressed by this project are:

- **Station Management:** Tracking the location, operational status (operational, maintenance), and specifications of all charging stations.
- **Slot Availability:** Real-time monitoring and display of individual charging slot availability within each station.
- **Bookings:** Allowing customers to find available slots, reserve charging times, and manage their bookings.
- **Payments:** Securely processing payments for charging sessions.
- **User Management:** Handling different user types (Customers, Station Managers, Admins).
- **Reporting:** Generating insights on station utilization and revenue.

Stakeholders

- **Admin:** Oversees the entire system, manages configurations, and needs full access and reporting.
- **Customer:** The end-user who books slots, manages vehicles, and makes payments.
- **Station Manager:** Responsible for one or more stations; monitors slot status and local bookings.
- **Technician:** Handles maintenance and repairs; needs access to station/slot details and cases.

Business Process Flow

1. **Discovery:** Customer finds nearby stations and checks slot availability.
2. **Booking:** Customer selects a station, slot, and time, then confirms.
3. **Allocation:** The system marks the slot as 'Reserved'.
4. **Session:** Customer arrives and starts the charging session.
5. **Calculation:** Session ends. System calculates the cost.
6. **Payment:** Customer pays via the integrated gateway; system records the transaction.
7. **Update:** System marks the slot as 'Available' again.

Industry Analysis & AppExchange

The EV charging market is rapidly expanding. Key competitors in the Indian market include Tata Power EZ Charge, ChargeGrid, Ather Grid, and Statiq. This CRM can differentiate through superior reliability tracking, a seamless booking experience, and advanced analytics. Functionality was inspired by AppExchange categories like Field Service Management (for technician dispatch), Booking/Scheduling Apps, and Subscription Billing/Payment Apps (like Stripe Connectors).

Phase 2: Org Setup & Configuration

Salesforce Edition

- **Edition Used:** Developer Edition.

Company Profile & Business Hours

- **Company Information:** The company profile was set up with the name "EV Charging CRM Solutions Pvt Ltd" and the appropriate address, time zone, and locale.
- **Business Hours:** Default business hours for support cases were set to 9:00 AM to 6:00 PM, Monday-Friday, IST.

User Setup & Roles

- **User Creation:** Standard users were created to represent key stakeholders, including System Administrator, Station Manager, and Customer Support.
- **Role Hierarchy:** A vertical role hierarchy was defined to enable managers to view records owned by their subordinates. The structure includes CEO, Operations Manager, Station Manager, and Station Staff.

Permission Sets

- **Manage Payments:** A Permission Set was created to grant additional "Manage Payments" access (e.g., Edit/Delete on the Payment object) to specific finance users without changing their profile.

Phase 3: Data Modeling & Relationships

Custom Objects

The core data model for the CRM consists of the following custom objects:

- **Charging Station (Charging_Station__c):** Represents a physical charging station location.
- **Charging Slot (Charging_Slot__c):** Represents an individual charging point within a Station. This has a Master-Detail relationship to the Charging Station.
- **Booking (Booking__c):** Records a customer's reservation of a specific Slot. It has lookups to Account, Contact, Slot, and Vehicle.
- **Payment (Payment__c):** Records financial transactions related to Bookings. It has a lookup to Booking.
- **Vehicle (Vehicle__c):** Represents a customer's electric vehicle, linked via lookup to Account/Contact.
- **Case (Case__c):** Represents customer issues after charging.

Key Fields & Relationships

- **Charging Station (Charging_Station__c):**
    - Location__c (Geolocation)
    - Status__c (Picklist: Operational, Maintenance, Offline)
- **Slot (Charging_Slot__c):**
    - Station__c (Master-Detail to Station)
    - Availability__c (Picklist: Available, Unavailable, Reserved) - This is updated by automation.
    - Hourly_Rate__c (Currency) - Used for payment calculation.
- **Booking (Booking__c):**
    - Slot__c (Lookup to Slot)
    - Contact__c (Lookup to Contact)
    - Start_Time__c (Date/Time)
    - Status__c (Picklist: Pending, Confirmed, Completed, Cancelled) - This field triggers automations.

    - Total_Price__c (Currency)
    - Payment_Status__c (Picklist: Unpaid, Paid, Failed)
    - Reminder_Sent__c (Checkbox) - Used by the reminder flow.
- **Payment (Payment__c):**
    - Booking__c (Lookup to Booking)
    - Amount__c (Currency)
    - Status__c (Picklist: Pending, Succeeded, Failed)
    - Transaction_ID__c (Text, External ID) - For linking to the payment gateway.
- **Vehicle (Vehicle__c):**
    - Vehicle_Identifier__c (Text, Name Field - e.g., License Plate)
    - Contact__c (Lookup to Contact)

Phase 4: Process Automation (Admin)

Validation Rules

- **Use Case:** Prevent Booking End Time Before Start Time.
- **Object:** Booking__c.
- **Error Condition Formula:** End_Time__c < Start_Time__c.
- **Error Message:** "Booking End Time cannot be earlier than the Start Time.".

Scheduled Flow: Send 24-Hour Appointment Reminders

- **Flow Type:** Schedule-Triggered Flow.
- **Trigger:** Runs on a defined schedule (e.g., Hourly).
- **Logic:**
    1. **Get Records:** Finds Booking__c records where Start_Time__c is 24 hours from now AND Reminder_Sent__c is false.
    2. **Loop:** Iterates through the found Bookings.

3. **Send Email Alert:** Uses a standard Email Alert action (BookingReminder_24h_EmailAlert) to notify the customer.
4. **Update Records:** After the loop, performs a single update to set the Reminder_Sent__c checkbox to true for all processed records.

Record-Triggered Flow: Auto Update Slot Occupancy

- **Flow Type:** Record-Triggered Flow (After Save).
- **Object:** Booking__c.
- **Entry Conditions:** Runs when Status__c is changed to 'Confirmed' or 'Cancelled'.
- **Logic:**
    1. **Get Records:** Retrieves the related Slot__c record.
    2. **Decision:** Checks if the Booking status is 'Confirmed' or 'Cancelled'.
    3. **Update Records (Confirmed):** If 'Confirmed', updates the Slot's Availability__c to 'Unavailable'.
    4. **Update Records (Cancelled):** If 'Cancelled', updates the Slot's Availability__c to 'Available'.

Screen Flow: Guided Station Creation

- **Flow Type:** Screen Flow.
- **Logic:**
    1. **Screen 1:** Collects Station__c details (Name, Address).
    2. **Create Records:** Creates the new Station__c record.
    3. **Screen 2:** Asks "How many Slots to create for this Station?".
    4. **Loop:** Loops from 1 to the number entered.
    5. **Assignment (Inside Loop):** Adds new Slot record variables to a collection.
    6. **Create Records:** After the loop, performs a single create action to insert all Slot__c records in the collection.
    7. **Screen 3:** Displays a confirmation message.

Phase 5: Apex Programming (Developer)

Apex Trigger: BookingTrigger

- **Object:** Booking__c.
- **Events:** after insert, after update.
- **Purpose:** This trigger is "logic-less." It delegates all processing to the BookingTriggerHandler class, which is a best practice.
- **Logic:**
    - On after insert, it calls BookingTriggerHandler.handleAfterInsert(Trigger.new).
    - On after update, it calls BookingTriggerHandler.handleAfterUpdate(Trigger.new, Trigger.oldMap).

Apex Class: BookingTriggerHandler

This class contains the core logic for booking processing:

- **handleAfterInsert(List<Booking__c> newBookings):**
    - **Purpose:** Updates the related Slot__c to 'Unavailable' when a new Booking is 'Confirmed'.
    - **Logic:** Collects Slot__c IDs from confirmed Bookings into a Set<Id>, queries for those slots, updates their Availability__c field, and performs a single DML update. It includes a try-catch block for exception handling.
- **handleAfterUpdate(List<Booking__c> newBookings, Map<Id, Booking__c> oldMap):**
    - **Purpose:** Handles status changes on existing Bookings.
    - **Logic:**
        1. Iterates through newBookings and compares the new Status__c with the oldMap version.
        2. If Status changes to 'Confirmed', it updates the related Slot to 'Unavailable'.

3. If Status changes to 'Cancelled', it updates the related Slot to 'Available'.
4. **Crucially, if Status changes to 'Confirmed'**, it calls the asynchronous future method PaymentGatewayIntegration.processPayment(...) to handle the payment callout.

Test Classes: BookingTrigger_Test

- **Purpose:** To verify the trigger and handler functionality and achieve >75% code coverage for deployment.
- **Key Components:**
  - @isTest Annotation: Marks the class as test code.
  - @testSetup Method: Creates common test data (Accounts, Stations, Slots) for all test methods.
  - **Test Methods:** Each method tests a specific scenario (e.g., testAfterInsertTrigger_UpdatesSlot, testAfterUpdate_CancelFreesSlot).
  - **Arrange, Act, Assert:** Test methods set up data, perform an action (like DML), and use System.assertEquals() to verify the outcome is correct.
  - Test.startTest() and Test.stopTest() are used to reset governor limits around the action being tested.

Phase 6: User Interface Development

Lightning App Builder: Custom App & Home Page

- **Custom App: "EV Charging CRM"**
  - A dedicated Lightning App was created to provide users with a focused experience.
  - **Tabs Added:** Includes Home, Accounts, Contacts, Vehicles, Charging Stations, Charging Slots, Bookings, and Payments, ordered logically.

    - **Profiles:** Assigned to System Administrator and other relevant profiles like Station Manager.

- **Custom Home Page: "EV Charging Home Page"**
    - A custom Home Page was designed using the Lightning App Builder.
    - **Components:**
        - **Dashboard Component:** Added to the main region to display the "EV Charging Overview" dashboard.
        - **Rich Text Component:** Used in the sidebar to provide "Quick Links" to common actions.
    - **Activation:** This page was activated and assigned as the App Default for the "EV Charging CRM" app.

Lightning Web Component (LWC): "Book a Slot" Form

- **Component Name:** bookASlot.
- **Purpose:** Provides a dynamic form for users to create new Booking__c records, as standard record creation cannot dynamically filter slots based on a station.
- **Key Features:**
    - **Dynamic Slot Fetching:**
        1. A lightning-combobox for "Select a Station" is populated on load using a @wire call to an Apex method (bookingController.getStations()).
        2. When a station is selected, the handleStationChange JavaScript function **imperatively** calls another Apex method (bookingController.getAvailableSlots(stationId)).
        3. The second combobox, "Select an Available Slot," is then populated with only the available slots for the chosen station.
    - **Record Creation:** On submission, the handleBookNow function uses the createRecord wire adapter to create the new Booking__c record with the selected slot and start time.

    - **User Feedback:** Uses lightning-spinner during loading and ShowToastEvent to display success or error messages.
- **Placement:** The LWC was added to the "EV Charging Home Page" using the Lightning App Builder.

Phase 7: Integration & External Access

This phase simulates a callout to a third-party payment gateway when a booking is confirmed.

Configuration for Callouts

- **Remote Site Settings:** A Remote Site named Payment_API_Mock was created to whitelist the mock API domain (https://httpbin.org) for Apex callouts.
- **Named Credentials (Preferred Method):**
    - A Named Credential named Payment_API was created, pointing to the same URL (https://httpbin.org).
    - This is the preferred method as it separates the endpoint URL and authentication from the code.
    - **Authentication:** Set to Anonymous and No Authentication as the mock API is public.

Apex Callout Example (Mock Payment Processing)

- **Apex Class: PaymentGatewayIntegration**
    - **Purpose:** Simulates sending booking details to an external payment processor.
    - **Method:** processPayment(Id bookingId, Decimal amount).
    - **Annotation:** @future(callout=true) - This is required for two reasons:
        1. It allows the method to make external callouts.
        2. It runs the logic asynchronously, which is required when a callout is initiated from a trigger.
    - **Logic:**

1. Constructs an HttpRequest.
2. Sets the endpoint using the Named Credential: callout:Payment_API/post.
3. Sets the method to POST and Content-Type to application/json.
4. Builds a JSON body with the booking ID and amount.
5. Uses Http.send(req) to send the request.
6. Checks the response status code (expecting 200).
7. If successful, performs a DML update on the original Booking__c record to set Payment_Status__c to 'Paid'.
8. Uses a try-catch block to handle System.CalloutException errors.

- **Trigger Handler Modification:**
  - The BookingTriggerHandler.handleAfterUpdate method was modified.
  - It checks if the Booking__c status changed to 'Confirmed'. If it did, it calls PaymentGatewayIntegration.processPayment(...) to initiate the asynchronous callout.

Phase 8: Data Management & Deployment

Data Import Wizard (Initial Data Upload)

- **Use Case:** The Data Import Wizard was used for the initial upload of foundational Account and Contact records.
- **Process:**
  1. A CSV file (e.g., sample_accounts.csv) was prepared.
  2. The wizard was launched via Setup.
  3. The 'Accounts' object and 'Add new records' action were selected.
  4. Fields were mapped, and the import was initiated.

Duplicate Rules

- **Purpose:** To maintain data quality by preventing users from creating duplicate Contact records based on email.
- **Setup:**
    1. **Matching Rule:** A rule named Contact_Email_Matching_Rule was created on the Contact object to find records where the Email field is an exact match.
    2. **Duplicate Rule:** A rule named Block_Contacts_with_Duplicate_Email was created.
    3. **Action:** The Action on Create was set to **Block**.
    4. **Alert:** An alert text "A contact with this email already exists" was configured.
    5. Both rules were activated.

Deployment Methods

- **SFDX (Salesforce Developer Experience via VS Code):**
    - This was the method used for deploying metadata (objects, fields, code, flows) from the developer org to a target org.
    - **Process:**
        1. **Authorize Target Org:** Connected VS Code to the destination org using SFDX: Authorize an Org.
        2. **Retrieve (if needed):** Ensured the manifest/package.xml file included all components to be deployed.
        3. **Deploy:** Right-clicked the project folder and selected SFDX: Deploy Source to Org to push the metadata.

Phase 9: Reporting, Dashboards & Security Review

Reports

- **Daily Bookings:**
    - A report on the Booking object, grouped by Created Date (by Calendar Day).

    - **Purpose:** To track the volume of new bookings each day.
- **Revenue by Station:**
    - A report on the Booking object, filtered for Status = 'Completed'.
    - It is grouped by Station and includes a SUM on the Total_Price__c field.
    - **Purpose:** To analyze revenue generation per station.
- **Slot Utilization:**
    - A report on the Slot object, grouped first by Station and then by Availability__c.
    - **Purpose:** To visualize the current status (e.g., Available, Unavailable) of slots across all stations.

Dashboard

- **Dashboard Name:** EV Charging Overview.
- **Purpose:** To provide a high-level, at-a-glance view of key performance indicators.
- **Components:**
    1. **Total Bookings Today:** A Metric component using the Daily Bookings report (filtered for Today).
    2. **Revenue by Station:** A Bar Chart component using the Revenue by Station report.
    3. **Overall Slot Utilization:** A Gauge or Chart component using the Slot Utilization report to show the ratio of available vs. unavailable slots.

Security Review

A layered security model was configured:

- **Organization-Wide Defaults (OWD):**
    - **Booking:** Private
    - **Payment:** Private
    - **Station:** Public Read Only

    - **Slot:** Controlled by Parent (inherits read-only access from Station via the Master-Detail relationship).

- **Role Hierarchy:**
    - A vertical hierarchy (CEO -> Operations Manager -> Station Manager) was built to allow managers to view their subordinates' records.
- **Sharing Rules:**
    - **Use Case:** A criteria-based sharing rule named "Share Bookings within Station" was created.
    - **Purpose:** To allow all users in the "Station Staff" role to see all other bookings for their *same* station, even though the OWD for Booking is Private.
    - **Access Level:** Read Only.
- **Profiles & Permission Sets:**
    - **Profiles:** A custom profile Station Staff was cloned from Standard User to define base-level object permissions (e.g., Read/Create/Edit on Bookings).
    - **Permission Sets:** Used to grant *additional* access. For example, the Manage Payments permission set grants Edit/Delete on the Payment object to specific users without changing their profile.
- **Field-Level Security (FLS):**
    - Used to restrict access to specific fields. For example, the Hourly_Rate__c field on the Slot object was made Read-Only for the Station Staff profile.

Quality Assurance Testing:

Test cases were prepared for each feature to ensure functionality aligns with requirements.

Test Case 1: Validation Rule

- **Use Case / Scenario:** Prevent Booking End Time Before Start Time.
- **Test Steps:**

1. Navigate to the Booking tab and click "New".
2. Select a Contact and a Slot.
3. Enter Start_Time__c = 10/20/2025, 10:00 AM.
4. Enter End_Time__c = 10/20/2025, 9:00 AM (i.e., before the start time).
5. Click "Save".

- **Expected Result:** A validation error message appears: "Booking End Time cannot be earlier than the Start Time.". The record is not saved.
- **Actual Result:** (Screenshot) As expected, the error message appeared at the top of the page, and the save was prevented.

Test Case 2: Record-Triggered Flow

- **Use Case / Scenario:** Slot availability updates to 'Unavailable' when a Booking is 'Confirmed'.
- **Test Steps:**
    1. Create a new Charging_Slot__c record. Verify its Availability__c is 'Available'.
    2. Create a new Booking__c record associated with the Slot from Step 1.
    3. Set the Status__c field to 'Confirmed'.
    4. Click "Save".
- **Expected Result:** The flow triggers. The related Charging_Slot__c record's Availability__c field is automatically updated from 'Available' to 'Unavailable'.
- **Actual Result:** (Screenshot) As expected, the flow ran, and the Slot's availability was correctly set to 'Unavailable'.

Test Case 3: Apex Callout (Asynchronous)

- **Use Case / Scenario:** Payment Status updates to 'Paid' after a Booking is confirmed.
- **Test Steps:**

1. Create a new Booking__c record with Status__c = 'Pending' and Payment_Status__c = 'Unpaid'.
2. Update the Booking__c record's Status__c from 'Pending' to 'Confirmed'.
3. Click "Save".
4. Navigate to Setup -> Apex Jobs to monitor the future method.

- **Expected Result:** The BookingTrigger fires, and the PaymentGatewayIntegration future method is queued and runs successfully. The Booking__c record's Payment_Status__c field is updated from 'Unpaid' to 'Paid'.
- **Actual Result:** (Screenshot) As expected, the Apex job completed, and the Payment_Status__c field on the Booking record was successfully updated to 'Paid'.

Conclusion

This project successfully delivered a comprehensive EV Charging CRM on the Salesforce platform. It addresses all core business requirements, from customer-facing booking and payment processing to internal station management and security. The application provides a scalable foundation for managing charging operations, automating key processes like payment callouts and reminders , and delivering critical business insights through reports and dashboards. The use of platform-native tools and a logic-less trigger framework ensures the system is both robust and maintainable for future enhancements.