Freelance-Force-Suite Project – Full Explanation

Phase 1: Problem Understanding & Industry Analysis

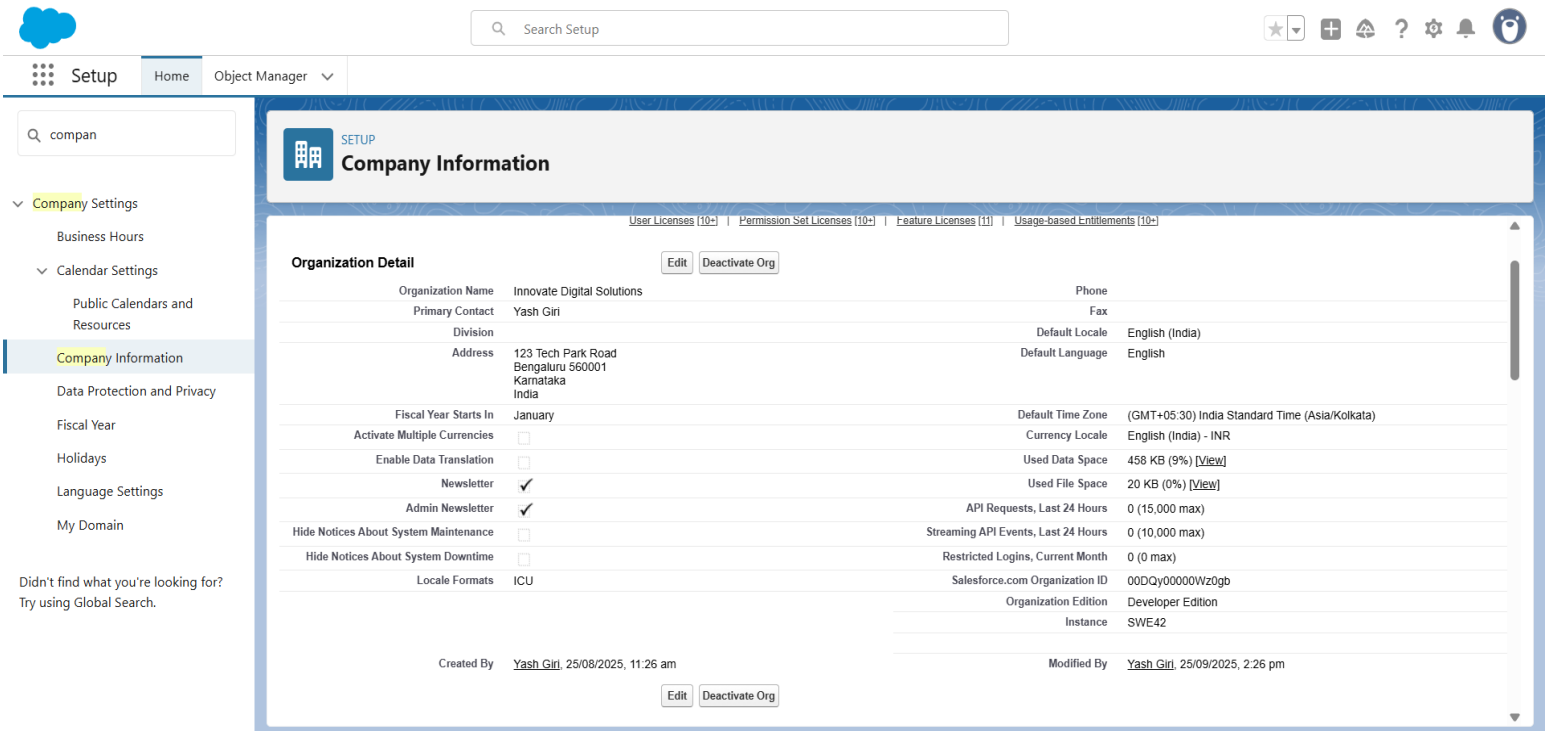
👉 Goal: Understand what we're building and why.

1. **Requirement Gathering**
    - Manage all client information and project details in one central location.
    - Track billable hours accurately against specific projects and tasks.
    - Automate the creation of professional invoices based on logged time.
    - Generate reports to analyze revenue and project profitability.
2. **Stakeholder Analysis**
    - **The Freelancer** (Needs a simple, all-in-one system to manage their entire business lifecycle, from client acquisition to getting paid. This stakeholder acts as the Admin, End User, and Manager.).
3. **Business Process Mapping**
    - **Current Process:** A fragmented workflow using multiple tools: a spreadsheet for client tracking, a separate app for task management (like Trello), another for time tracking, and a word processor for manually creating invoices. This process is time-consuming and prone to data entry errors.
    - **Future Process:** A client (Account) has a Project__c record created. The freelancer logs time against the project using Time_Log__c records. At the end of a billing cycle, a button click automatically generates an Invoice__c record, summing up all unbilled hours. The freelancer then manually updates the invoice status to "Paid" upon receiving payment.
4. **Industry-specific Use Case Analysis**
    - Freelancers often juggle multiple clients and projects simultaneously. The system needs to provide a clear, consolidated view of all ongoing work and outstanding payments. It must also handle both fixed-price projects and hourly billing models.
5. **AppExchange Exploration**
    - **Decision for Payment Tracking:** To ensure the project remains **completely cost-free**, no paid payment processing integrations (like Stripe or PayPal connectors) will be used. The core business need will be met by creating a custom Invoice Status field, which the freelancer will manually update from "Sent" to "Paid" after receiving funds through their existing, external methods (e.g., bank transfer).

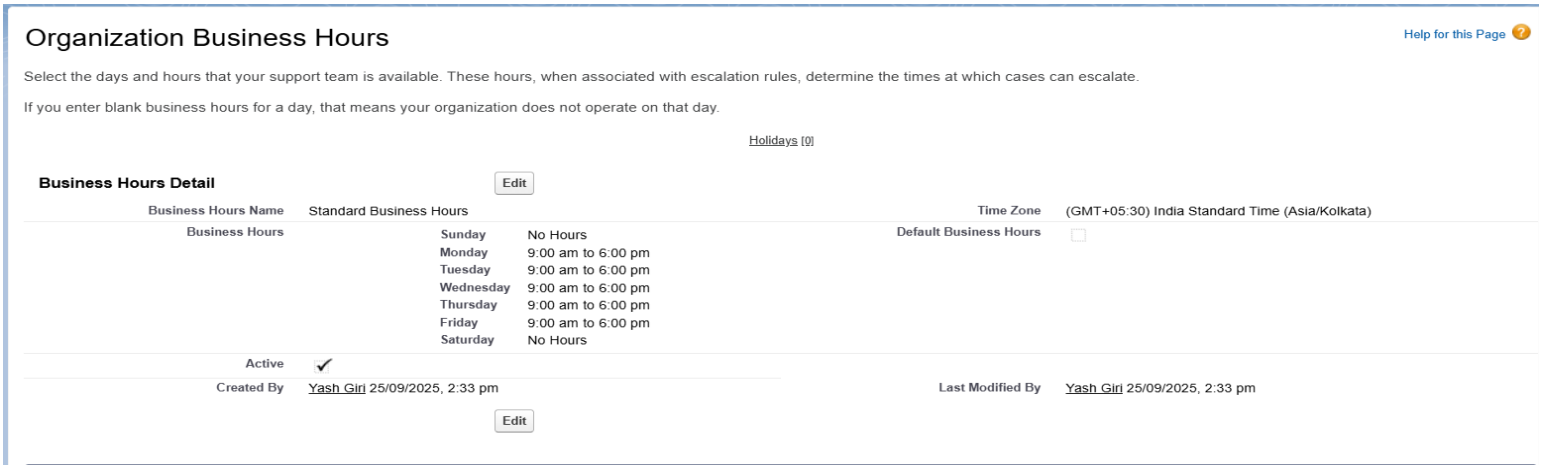
Phase 2: Org Setup & Configuration

👉 Goal: Prepare Salesforce environment.

1. **Salesforce Editions**
   - A free **Developer Edition Org** has been created and is in use for all development and configuration.
2. **Company Profile Setup**
   - The freelancer's business profile, including a business name, address, default time zone, and primary currency, has been configured in Setup > Company Information.



3. **Business Hours & Holidays**
   - Standard business hours (e.g., 9:00 AM - 6:00 PM, Monday-Friday) have been defined to ensure any time-based automations run correctly.



4. **Fiscal Year Settings**

   - A **Standard Fiscal Year** (January-December) has been configured to simplify revenue reporting for the calendar year.

5. **User Setup & Licenses**

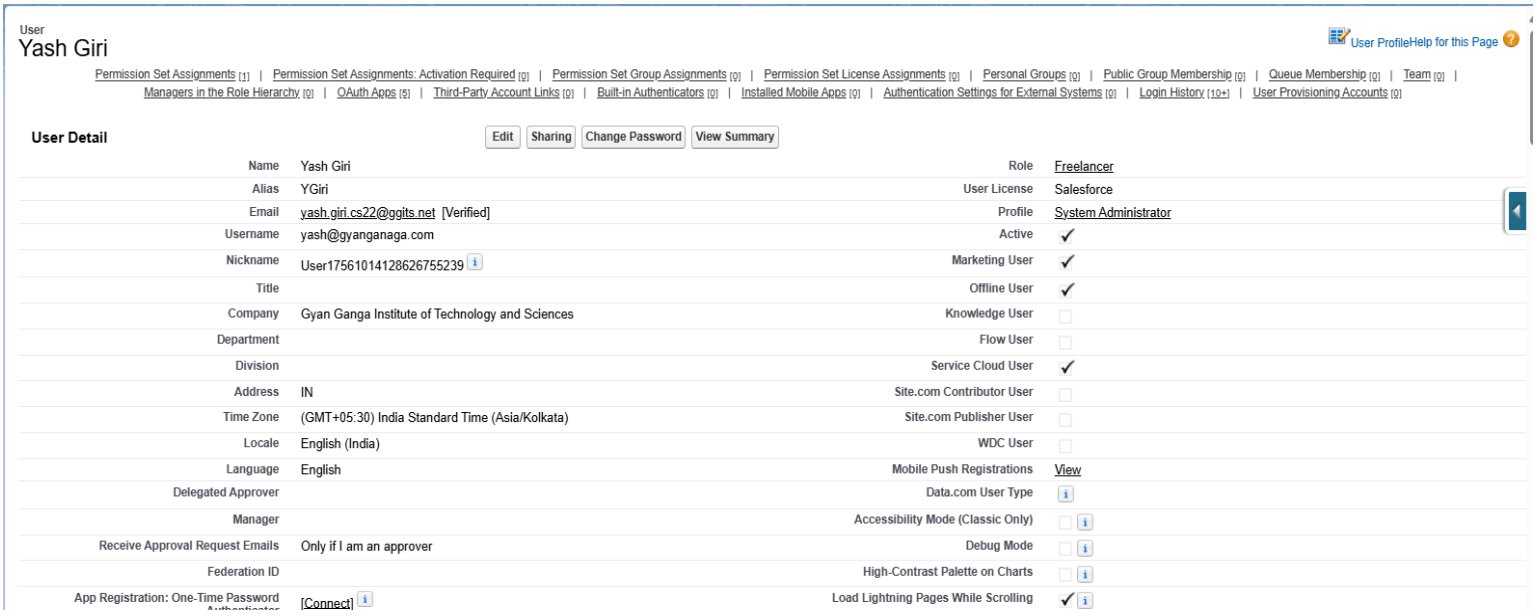
   - A single user record for the freelancer has been configured with a **Salesforce** license.

6. **Profiles**

   - The user is assigned the **System Administrator** profile, providing full access to build and customize the application.

7. **Roles**

   - A top-level role (e.g., "Freelancer") has been created and assigned to the user as a best practice for reporting and future scalability.



8. **OWD (Org-Wide Defaults)**

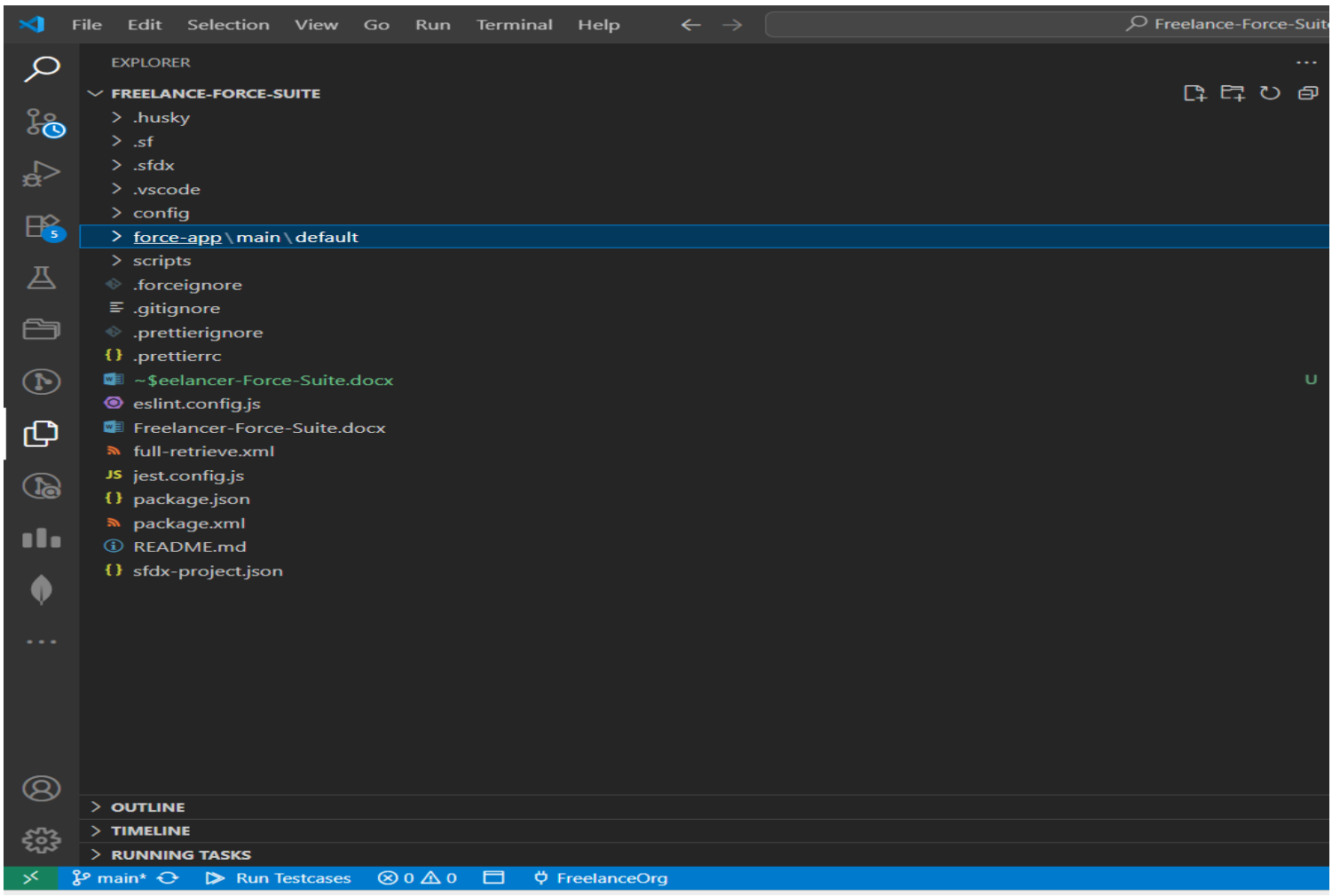
   - The Organization-Wide Defaults for all key standard objects (Account, Opportunity) and custom objects (Project__c) have been set to **Private** to ensure maximum data security.

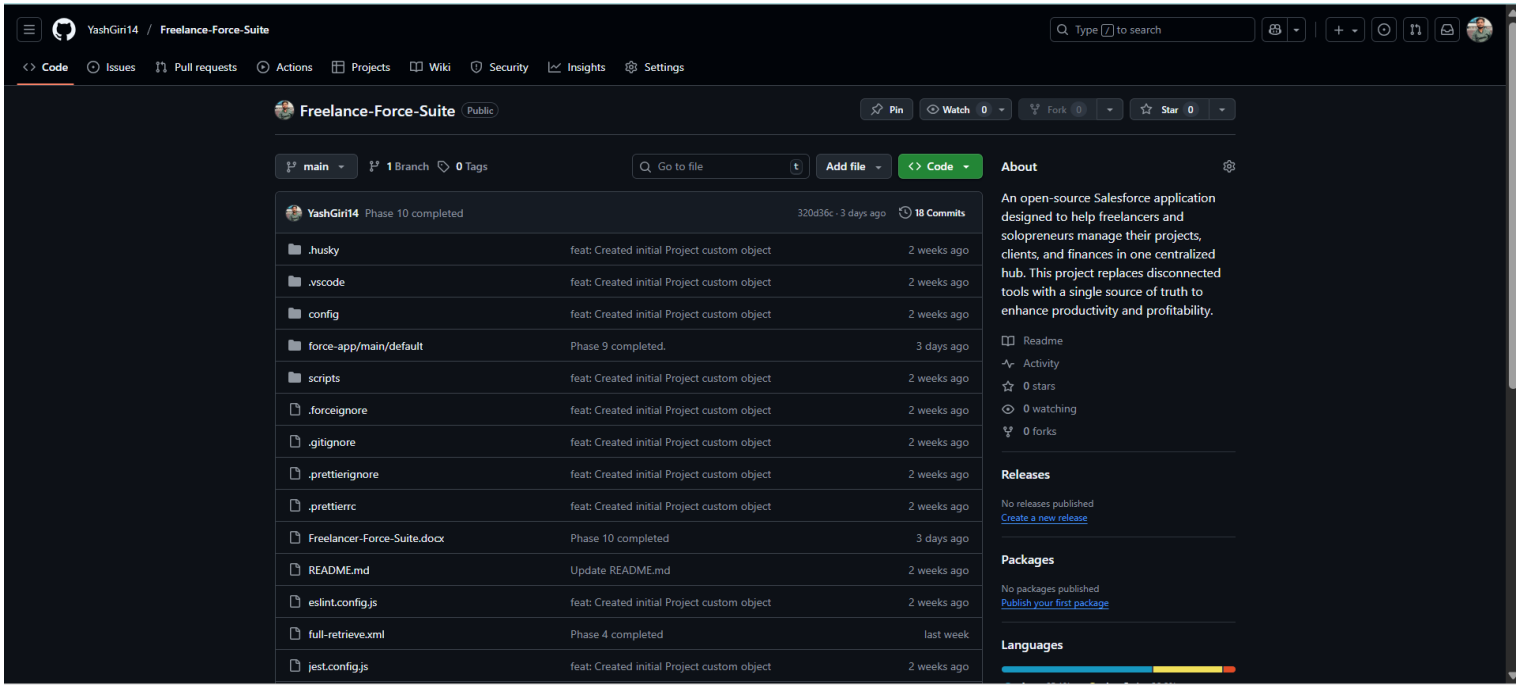
9. **Login Access Policies**

   - The default login and password policies for the org have been reviewed and confirmed.

10. **Deployment Basics**

- The Developer Org has been successfully connected to a local VS Code project. All setup and configuration from this phase have been retrieved and pushed to a public GitHub repository for version control.





Phase 3 : Data Modeling & Relationships

👉 Goal: Build the core data structure to manage clients, projects, time logs, and invoices.

1. **Standard & Custom Objects**

   - **Standard Objects:**

     - Account: To store client company information.
     - Contact: To store individual client contact details.

   - **Custom Objects:**

     - Project__c: The central object to track all details of a single project for a client.
     - Time_Log__c: To record individual blocks of billable time worked on a project.
     - Invoice__c: To generate and track the status of invoices sent to clients.

.

2. **Fields**

   - **Project Object Fields :**

     - Status (Picklist): Tracks the application's current stage (e.g., Planning, In Progress, Completed, On Hold).
     - Hourly Rate (Currency): The agreed-upon billing rate for the project.
     - Start Date (Date): When the project officially begins.
     - End Date (Date): The projected or actual completion date

   - **Time Log Object Fields :**

     - Date (Date): The date the work was performed.
     - Hours (Number): The number of hours worked (e.g., 2.5).
     - Description (Long Text Area): A summary of the work completed.
     - Status (Picklist): "Unbilled" or "Billed". Defaults to "Unbilled".

   - **Invoice Object Fields :**

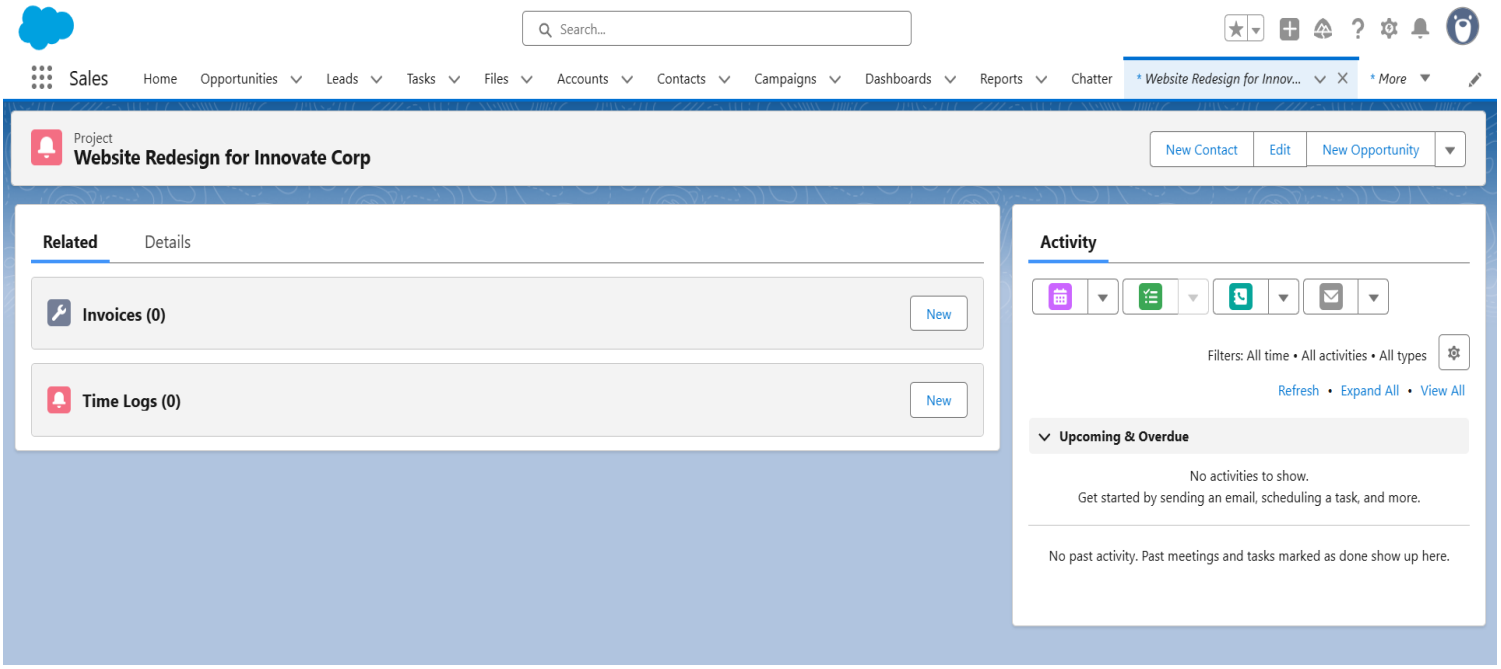
     - Status (Picklist): Tracks the invoice's lifecycle (e.g., Draft, Sent, Paid, Overdue).
     - Due Date (Date): The date payment is due.
     - Total Amount (Currency): The total calculated amount of the invoice.

3. **Record Types**

   - Record types are not required for the initial build of this project, as the process for all projects is the same. They could be added later if the freelancer decides to offer different types of services (e.g., "Fixed Price Project" vs. "Monthly Retainer").

4. Page Layouts

- The Project page layout is the main workspace. It has been configured to show the Time Logs and Invoices as related lists, so the freelancer can see all financial details for a project in one place.

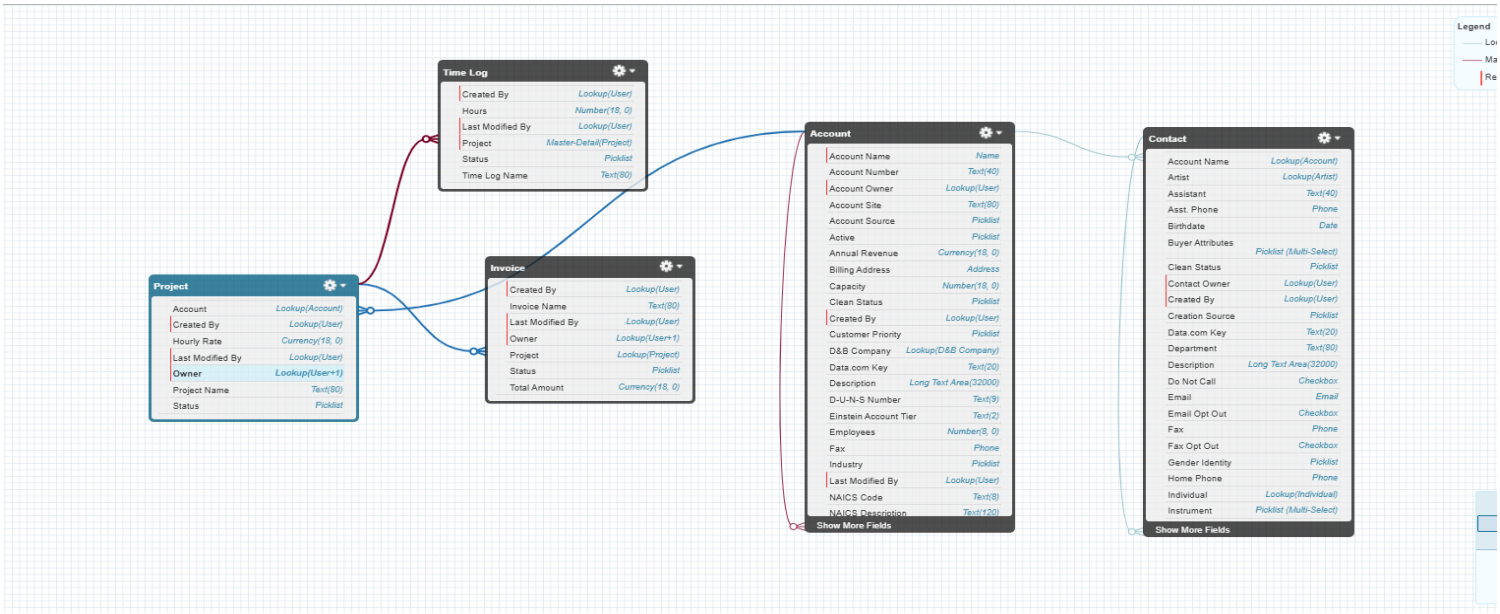


5. Compact Layouts

- The Project compact layout is configured to show the Project Name, Client Name, and Status at a glance on mobile and in list views.

6. Schema Builder

- The Schema Builder was used to visually design and confirm the relationships between Accounts, Contacts, Projects, Time Logs, and Invoices, ensuring the data model is logical and scalable.



7. **Lookup vs. Master-Detail vs. Hierarchical Relationships**

    - **Project to Account:** A required Lookup Relationship. A project must be linked to a client account, but they are independent records.
    - **Invoice to Project:** A required Lookup Relationship. An invoice must be related to a project.
    - **Time Log to Project:** A **Master-Detail Relationship**. A time log is a direct child of a project. If a project is deleted, all its associated time logs are automatically deleted. This ensures data integrity and allows for roll-up summary calculations

8. **External Objects**

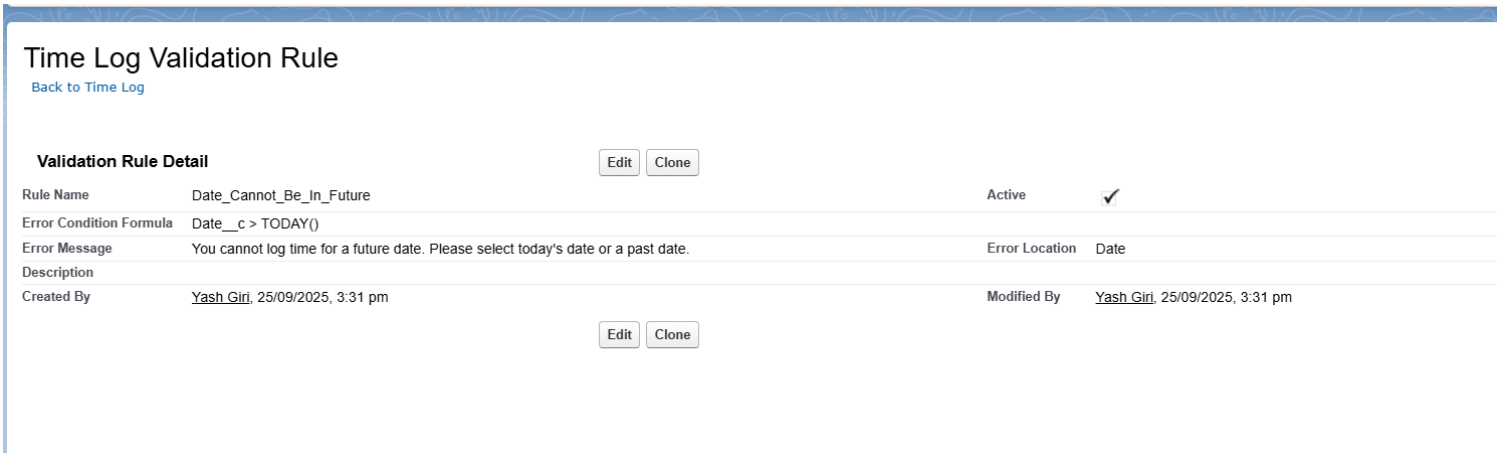
    - External objects are not in scope for this project. They could be used in a future enhancement to connect Salesforce with an external accounting system without migrating data.

Phase 4: Process Automation (Admin)

👉 **Goal:** Automate tasks to ensure data quality, calculate billing accurately, and reduce manual data entry.

1. **Validation Rules**

    - **Example:** A validation rule has been created on the Time Log object. It prevents a user from saving a time log if the Date is in the future, ensuring all logged work is for past or present dates.



2. **Workflow Rules (legacy)**

    - This is a legacy automation tool. All new automations for this project are being built in Flow Builder for better performance and capabilities.

3. **Process Builder (legacy)**

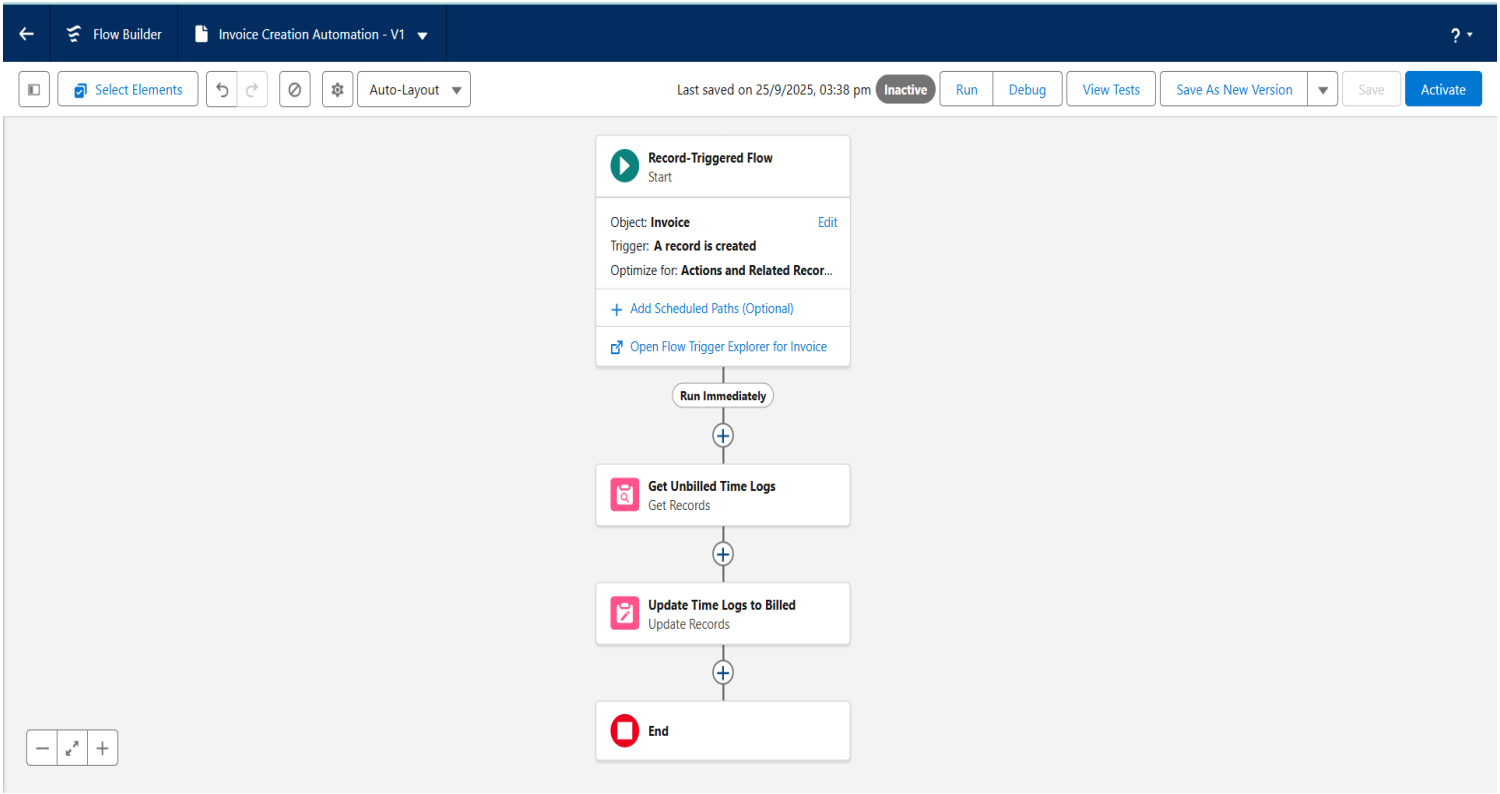
    - This is another legacy tool that has been superseded by Flow Builder, which is used for all record-triggered automations in this project.

4. **Approval Process**

    - An approval process is not required for this project's initial build, as it's designed for a single freelancer. This could be added in a future phase if the freelancer expands and hires a team that requires manager approval for timesheets or invoices.

5. **Flow Builder**

    - **Record-triggered Flow:** A powerful flow named "Invoice Creation Automation" runs automatically when a new Invoice record is created. This single flow is the core of the project's automation and performs several actions.



6. **Email Alerts**

    - No email alerts have been configured in this phase. A future enhancement could be to add an email alert within the flow to automatically send the newly created invoice to the client.

7. **Field Updates**

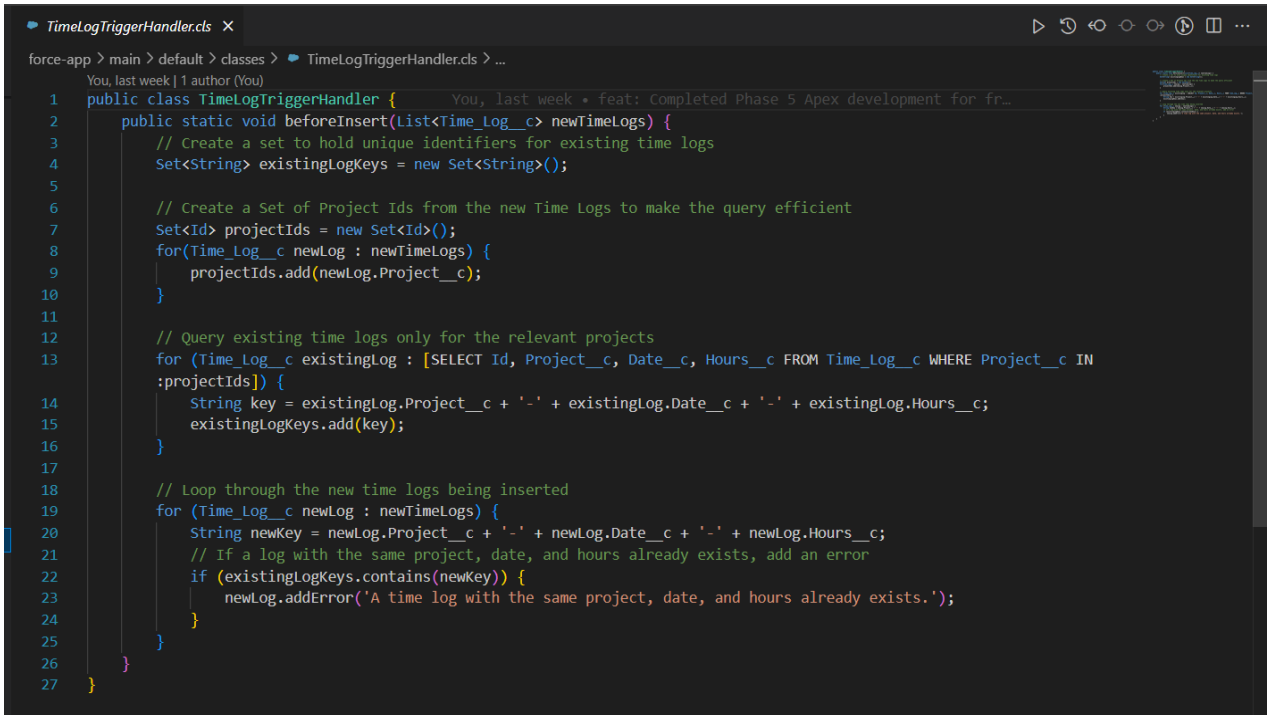
    - The "Invoice Creation Automation" flow performs multiple critical field updates:
        - It finds all related Time Log records with a status of "Unbilled" and updates their Status to "Billed."
        - It also populates the Invoice lookup field on those Time Log records, linking them to the new invoice.
        - It updates the Total Amount on the Invoice record by calculating the project's Total Logged Hours multiplied by the Hourly Rate.

8. **Tasks**
    - No automated task creation is included in this phase. A future enhancement could be a scheduled flow that creates a follow-up Task for the freelancer when an invoice becomes "Overdue."
9. **Custom Notifications**
    - Custom notifications are not required for this single-user system but could be added in the future.

Phase 5: Apex Programming (Developer)

👉 **Goal:** Add advanced custom logic that cannot be achieved with declarative tools.

1. **Classes & Objects**
    - Created a TimeLogTriggerHandler Apex class to contain the business logic for preventing duplicate time entries.
    - Created a MarkInvoicesOverdueSchedulable class to handle the logic for a nightly batch job.
2. **Apex Triggers**
    - A TimeLogTrigger has been implemented on the Time_Log__c object. It fires before insert to validate new records before they are saved to the database.
3. **Trigger Design Pattern**
    - A handler pattern was used to keep the trigger itself simple. All logic is delegated to the TimeLogTriggerHandler class, making the code organized and easier to test and maintain.



4. **SOQL & SOSL**

   - **SOQL:** The trigger handler uses a SOQL query to find existing time logs for comparison. The scheduled job uses a SOQL query to find all Invoice__c records that are past their due date and have not yet been paid.

5. **Collections: List, Set, Map**

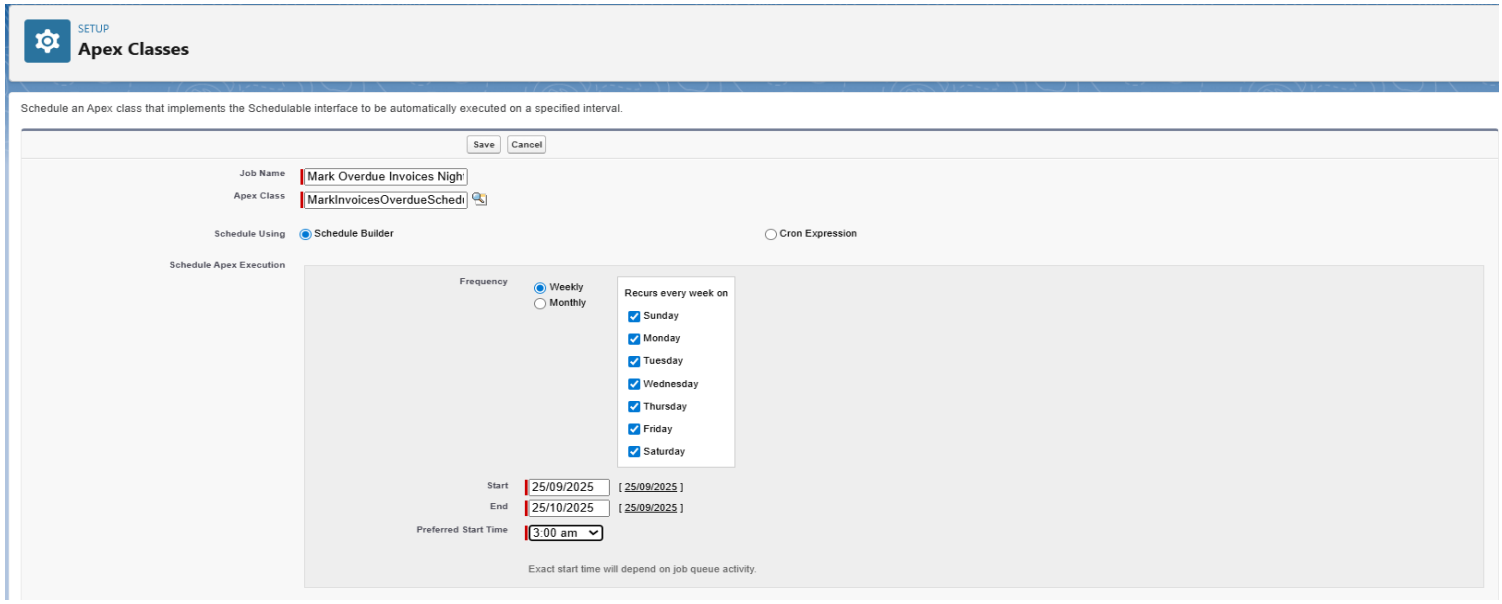
   - Used a Set<String> in the trigger handler to efficiently store unique keys representing existing time logs, allowing for a very fast check for duplicates.
   - Used a List<Invoice__c> in the scheduled job to hold the records that need to be updated.

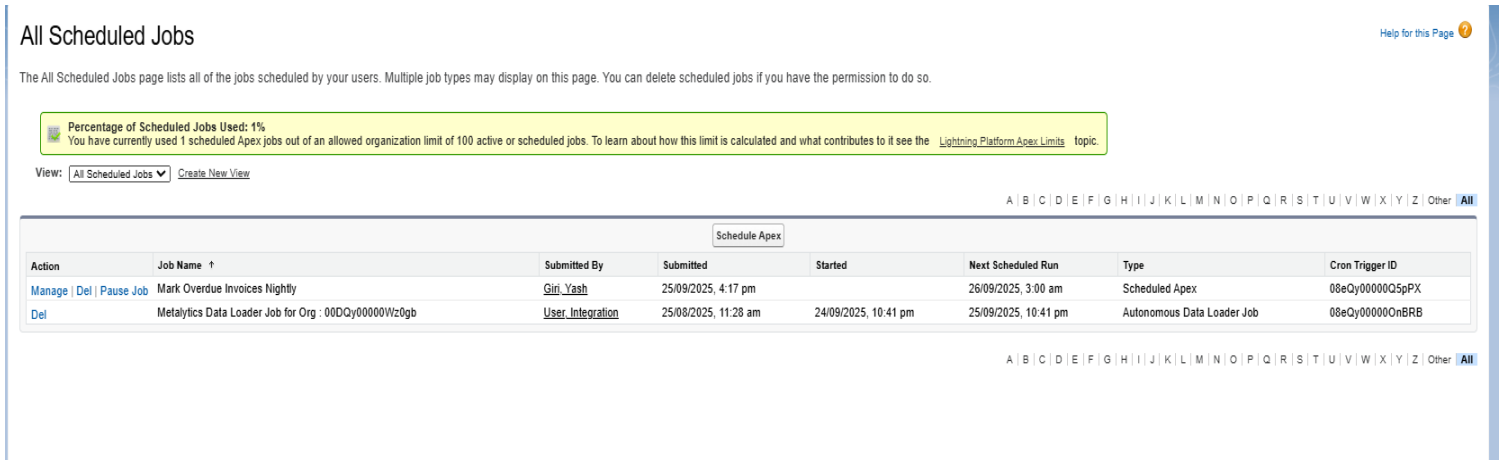
6. **Control Statements**

   - The trigger handler uses if statements and for loops to iterate through new Time_Log__c records and check if a duplicate already exists in the system.

7. **Scheduled Apex**

   - The MarkInvoicesOverdueSchedulable class runs on a nightly schedule. It queries for all invoices that are past their Due_Date__c and updates their Status__c to "Overdue."



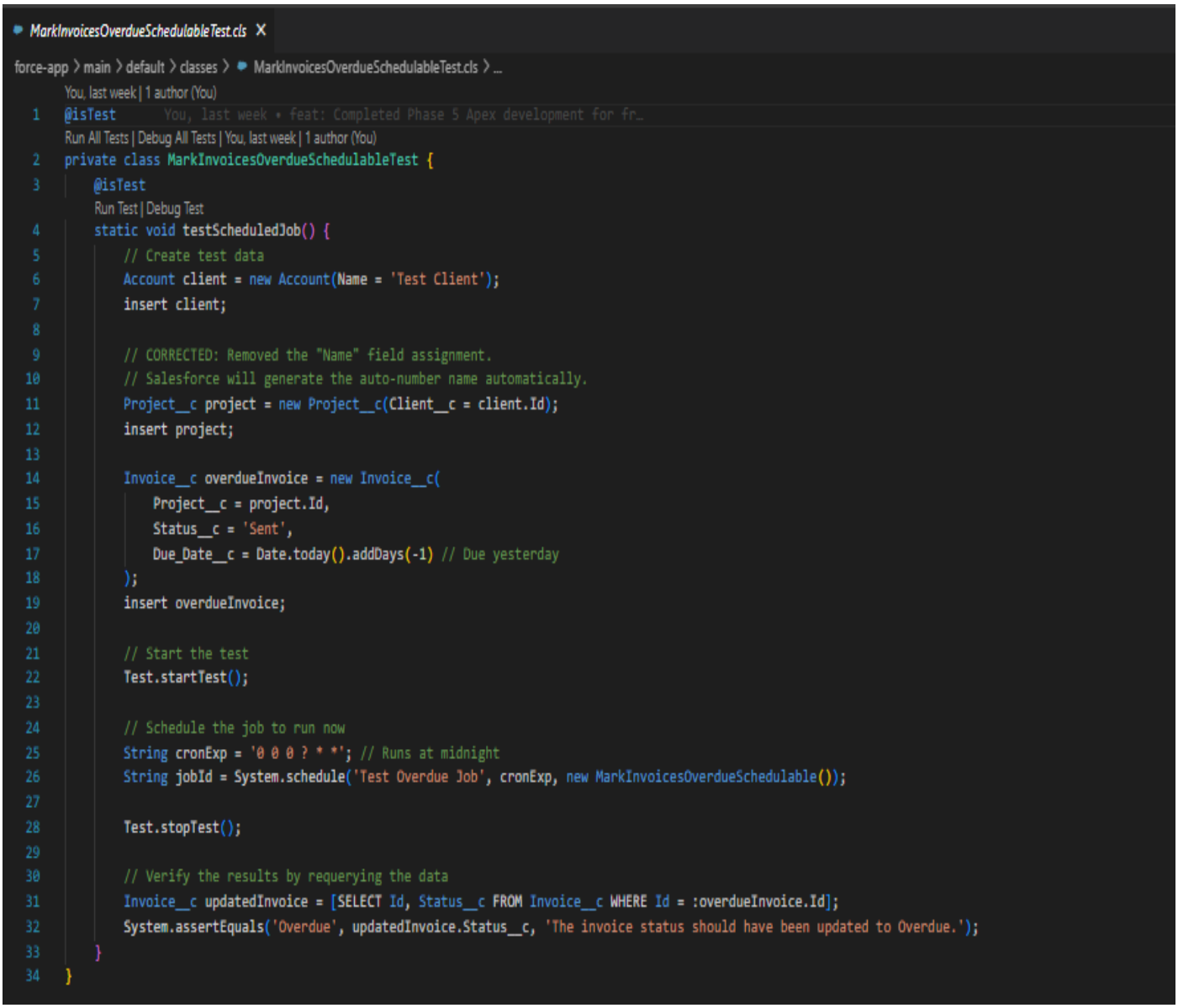


8. **Exception Handling**

   - The trigger uses the .addError() method to gracefully prevent a duplicate record from being saved and displays a clear error message to the user.
   - Test classes use try-catch blocks to verify that these errors are thrown correctly.

9. **Test Classes**

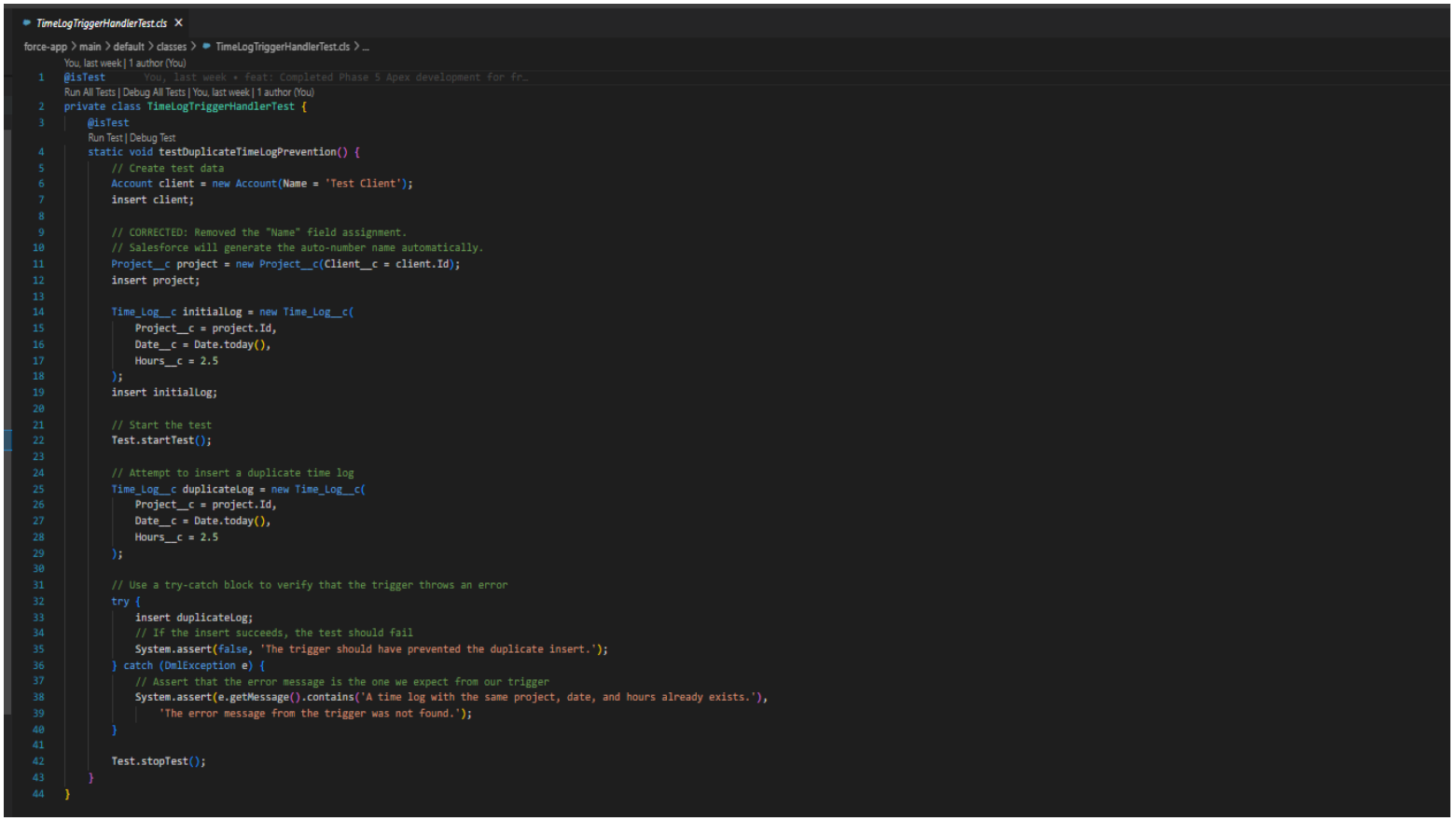
   - Created dedicated test classes (TimeLogTriggerHandlerTest, MarkInvoicesOverdueSchedulableTest) for all Apex code to ensure it functions as expected and meets Salesforce's code coverage requirements for deployment.



MarkInvoicesOverdueSchedulableTest.cls

force-app > main > default > classes > MarkInvoicesOverdueSchedulableTest.cls > ...

```
You, last week | 1 author (You)
1  @isTest      You, last week • feat: Completed Phase 5 Apex development for fr…
   Run All Tests | Debug All Tests | You, last week | 1 author (You)
2  private class MarkInvoicesOverdueSchedulableTest {
3      @isTest
       Run Test | Debug Test
4      static void testScheduledJob() {
5          // Create test data
6          Account client = new Account(Name = 'Test Client');
7          insert client;
8
9          // CORRECTED: Removed the "Name" field assignment.
10         // Salesforce will generate the auto-number name automatically.
11         Project__c project = new Project__c(Client__c = client.Id);
12         insert project;
13
14         Invoice__c overdueInvoice = new Invoice__c(
15             Project__c = project.Id,
16             Status__c = 'Sent',
17             Due_Date__c = Date.today().addDays(-1) // Due yesterday
18         );
19         insert overdueInvoice;
20
21         // Start the test
22         Test.startTest();
23
24         // Schedule the job to run now
25         String cronExp = '0 0 0 ? * *'; // Runs at midnight
26         String jobId = System.schedule('Test Overdue Job', cronExp, new MarkInvoicesOverdueSchedulable());
27
28         Test.stopTest();
29
30         // Verify the results by requerying the data
31         Invoice__c updatedInvoice = [SELECT Id, Status__c FROM Invoice__c WHERE Id = :overdueInvoice.Id];
32         System.assertEquals('Overdue', updatedInvoice.Status__c, 'The invoice status should have been updated to Overdue.');
33     }
34 }
```



```
TimeLogTriggerHandlerTest.cls
force-app > main > default > classes > TimeLogTriggerHandlerTest.cls > ...
You, last week | 1 author (You)
@isTest                You, last week • feat: Completed Phase 5 Apex development for fr...
Run All Tests | Debug All Tests | You, last week | 1 author (You)
private class TimeLogTriggerHandlerTest {
    @isTest
    Run Test | Debug Test
    static void testDuplicateTimeLogPrevention() {
        // Create test data
        Account client = new Account(Name = 'Test Client');
        insert client;

        // CORRECTED: Removed the "Name" field assignment.
        // Salesforce will generate the auto-number name automatically.
        Project__c project = new Project__c(Client__c = client.Id);
        insert project;

        Time_Log__c initialLog = new Time_Log__c(
            Project__c = project.Id,
            Date__c = Date.today(),
            Hours__c = 2.5
        );
        insert initialLog;

        // Start the test
        Test.startTest();

        // Attempt to insert a duplicate time log
        Time_Log__c duplicateLog = new Time_Log__c(
            Project__c = project.Id,
            Date__c = Date.today(),
            Hours__c = 2.5
        );

        // Use a try-catch block to verify that the trigger throws an error
        try {
            insert duplicateLog;
            // If the insert succeeds, the test should fail
            System.assert(false, 'The trigger should have prevented the duplicate insert.');
        } catch (DmlException e) {
            // Assert that the error message is the one we expect from our trigger
            System.assert(e.getMessage().contains('A time log with the same project, date, and hours already exists.'),
                'The error message from the trigger was not found.');
        }

        Test.stopTest();
    }
}
```

Phase 6: User Interface Development

👉 **Goal:** Make the application user-friendly and efficient for the freelancer.

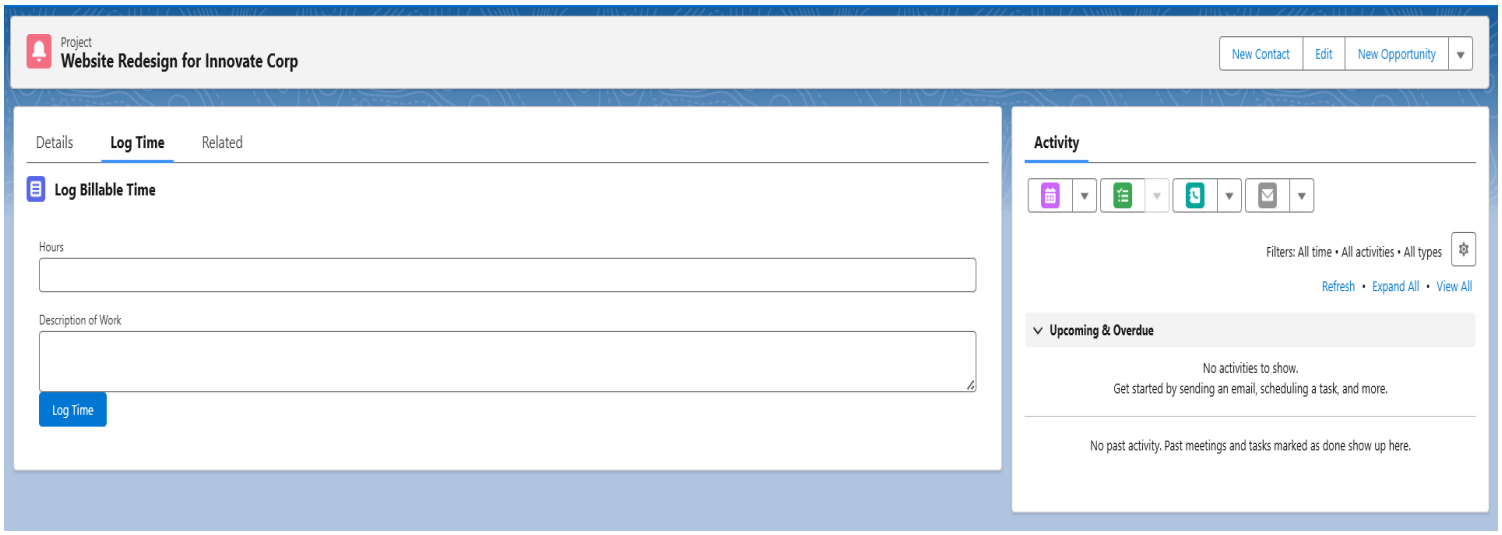
1. **Lightning App Builder**
    - Used the Lightning App Builder to create a dedicated, branded app named **"Freelance Suite"**.
2. **Record Pages**
    - Edited the default Project record page, adding a new custom tab to create a dedicated workspace for time logging.
3. **Tabs**
    - Added the standard **Projects** and **Invoices** tabs to the navigation menu of the "Freelance Suite" app.
    - Created a custom **"Log Time"** tab on the Project record page to house the new LWC.
4. **Home Page Layouts**
    - The home page has not been customized yet but could be enhanced in a later phase with dashboards showing revenue and project status.

5. **Utility Bar**

    - Added the standard **Recent Items** component to the app's utility bar for quick access to recently viewed records.

6. **LWC (Lightning Web Components)**

    - Built a custom Lightning Web Component named **timeLogger**. This component provides a clean and simple form for the freelancer to quickly enter their billable hours and description of work.



7. **Apex with LWC**

    - The timeLogger component's JavaScript imports the createTimeLog method from the TimeLogController Apex class, allowing the frontend form to securely create records in the backend database.

8. **Events in LWC**

    - The component uses the ShowToastEvent to display success or error notifications to the user after they attempt to log time.
    - It uses standard onchange and onclick events to handle user input from the form fields and button.

9. **Imperative Apex Calls**

    - The handleLogTime function in the component's JavaScript makes an **imperative call** to the createTimeLog Apex method when the user clicks the "Log Time" button.

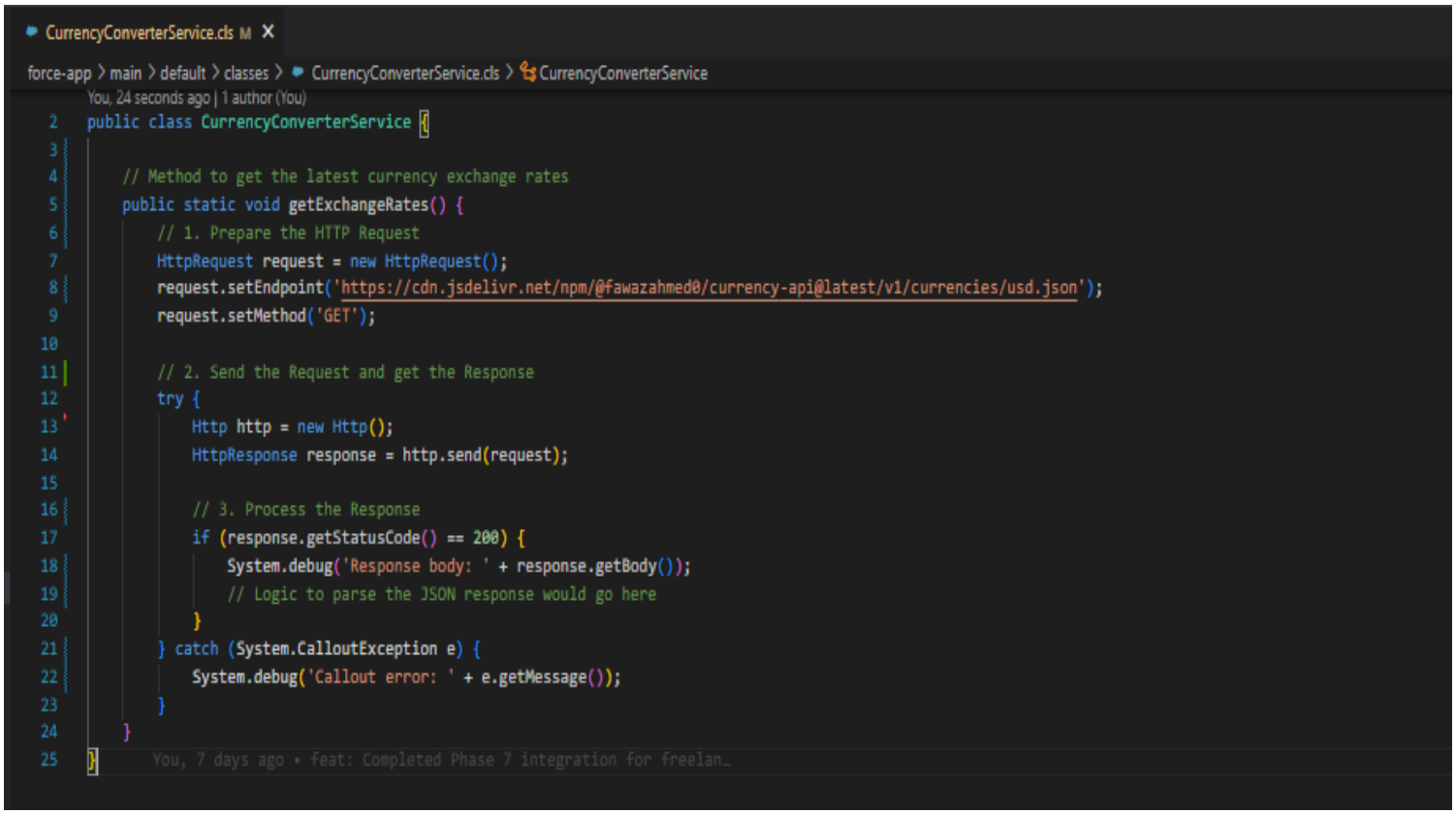
10. **Navigation Service**

    - The navigation service is not used in this component, but a future enhancement could be to add a button that navigates the user to a list of all time logs for the project.

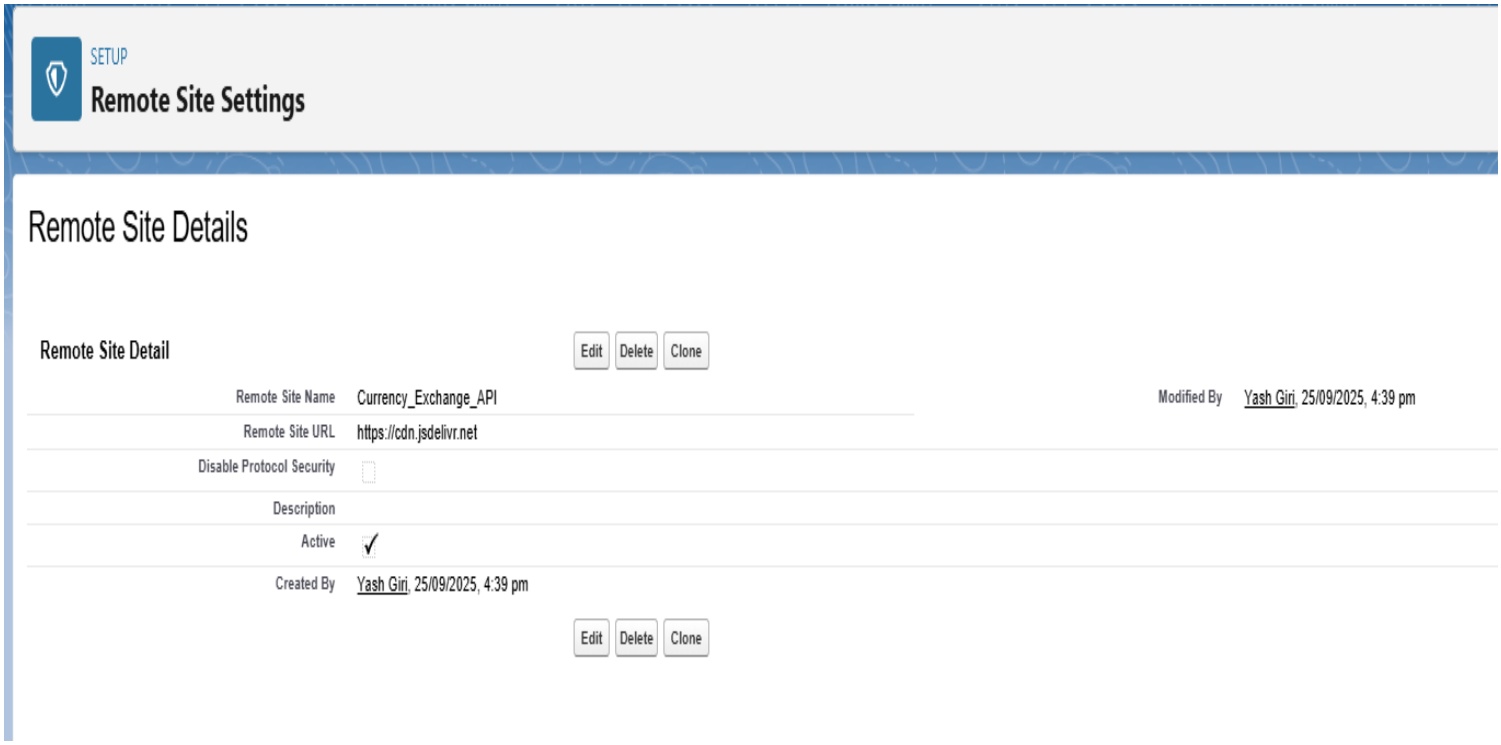
Phase 7: Integration & External Access

👉 **Goal:** Connect the application to outside systems to bring in external data.

1. **Named Credentials**
    - Named Credentials are not used in this phase but are a best practice for securely storing the URL and authentication details for an API endpoint.
2. **Web Services (REST/SOAP)**
    - A **REST callout** is used to connect to a free, public web service that provides live currency exchange rates. This allows the freelancer to get up-to-date conversion rates for international billing.
3. **Callouts**
    - An Apex class, CurrencyConverterService, was created to perform the HTTP callout. This class sends a request to the external currency API, parses the JSON response, and updates a field on the project record.



4. **Remote Site Settings**
    - To allow Salesforce to make the callout, the API's base URL (https://cdn.jsdelivr.net) was added to the **Remote Site Settings**. This is a mandatory security step to authorize the external domain.



5. **Other Integration Tools (Conceptual Knowledge)**
    - **Platform Events, Change Data Capture, Salesforce Connect:** These are more advanced integration tools not required for this project. They are used for real-time event-driven integrations and connecting to external databases.
6. **API Limits**
    - All integrations are subject to Salesforce's API call limits. The current integration is designed to be run on-demand (e.g., via a button click) to stay well within the free developer org limits.

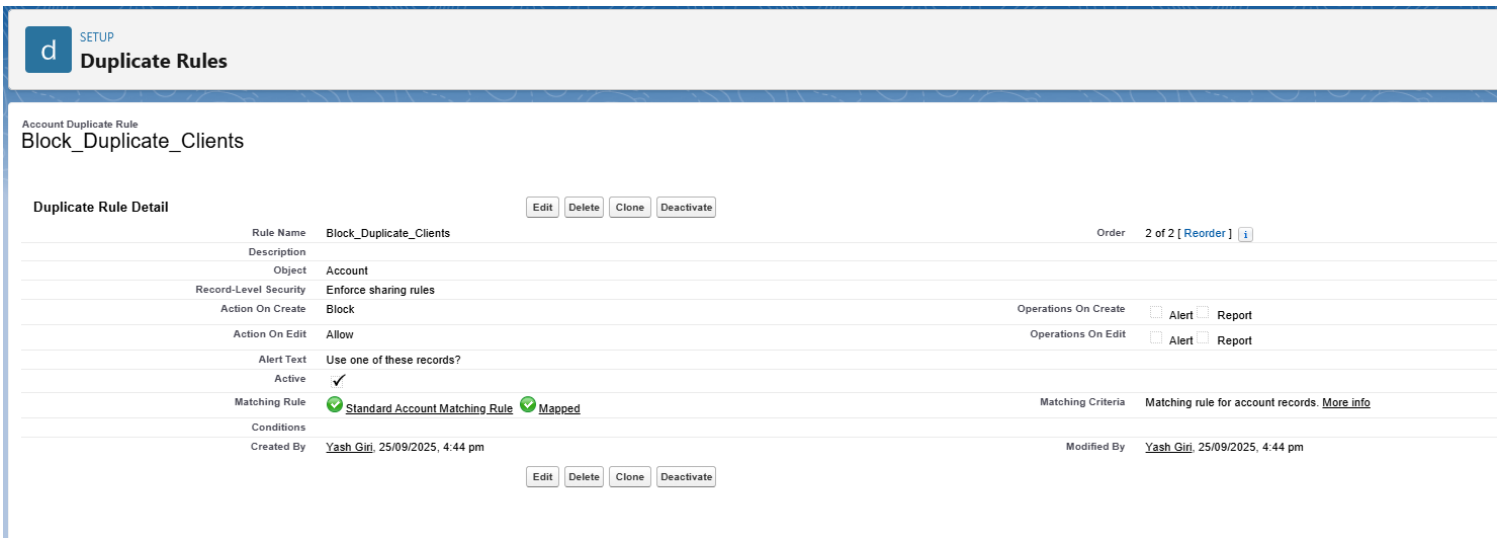
Phase 8: Data Management & Deployment

☞ **Goal:** Manage the application's data and understand deployment methodologies.

1. **Data Import Wizard**
    - The Data Import Wizard was used to perform a simple import of 3-4 sample **Account** records to act as the freelancer's clients. This was done by preparing a clients.csv file.
2. **Data Loader**
    - The desktop Data Loader application was used for a more complex data load. An projects.csv file was prepared, which included the unique Salesforce IDs of the Account records. This allowed us to insert new **Project** records and automatically link them to the correct clients.

3. **Duplicate Rules**

   - A **Duplicate Rule** was created and activated on the Account object. It uses a matching rule on the Account Name field to block the creation of a new client record if one with the exact same name already exists.

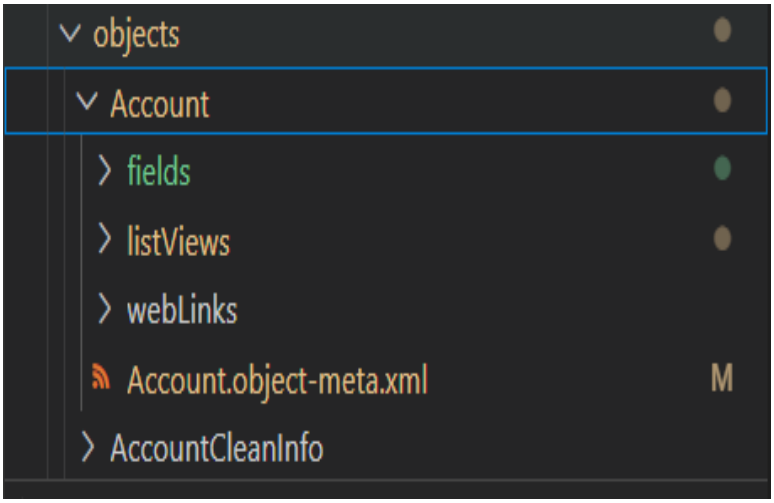


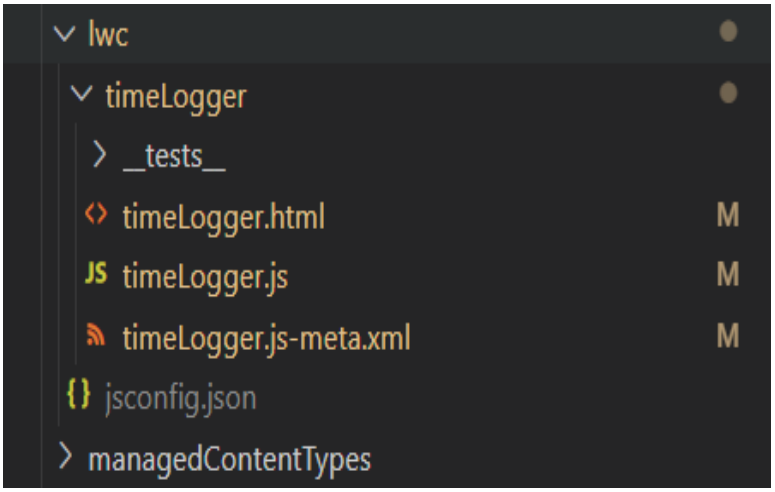
4. **Data Export & Backup**

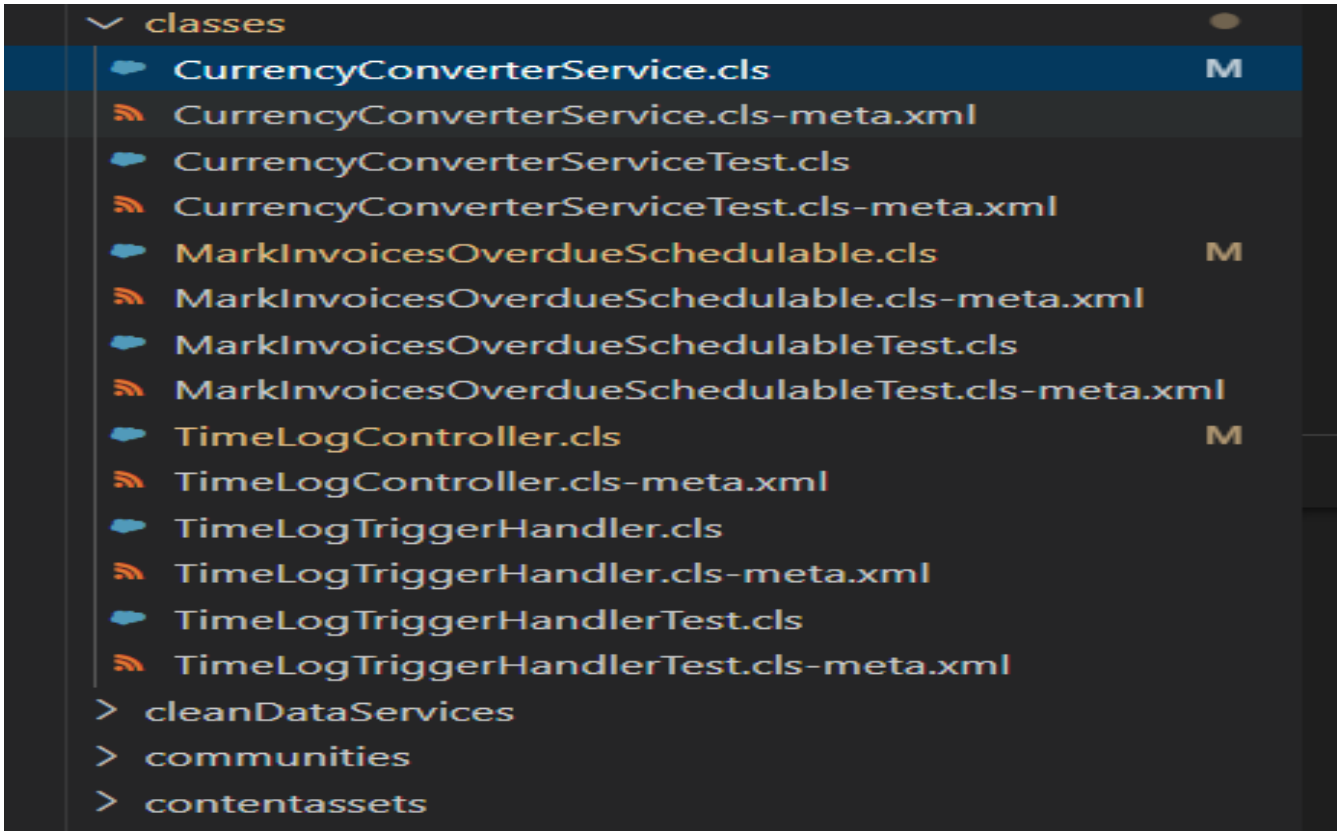
   - The native **Data Export** service was configured to perform an automated, weekly backup of all the project's key data, including Accounts, Projects, Time Logs, and Invoices.

5. **VS Code & SFDX**

   - This is the primary method used for all development and deployment in this project. All metadata components (Apex classes, custom fields, etc.) created in these phases have been successfully retrieved from the org to a local VS Code project and pushed to a GitHub repository for version control.





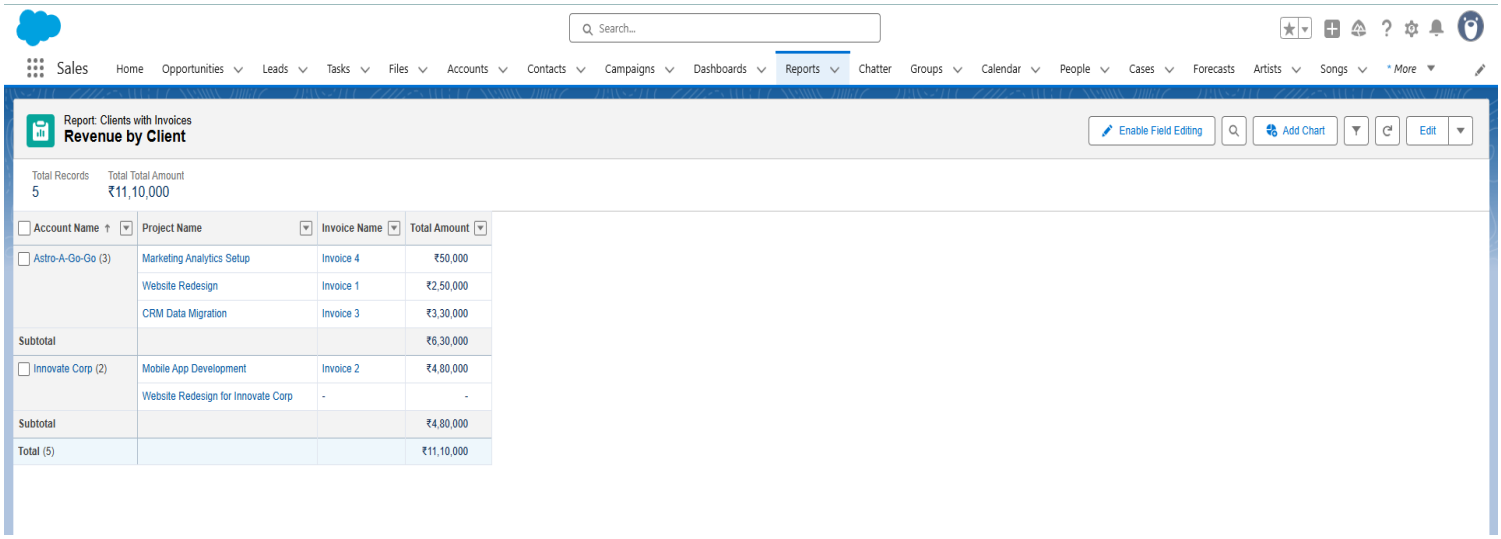


6. **Other Deployment Tools (Conceptual Knowledge)**
    - **Change Sets:** A UI-based tool for moving metadata between connected orgs (e.g., from a Sandbox to Production).
    - **ANT Migration Tool:** An older command-line deployment tool that uses XML files. It has largely been replaced by SFDX for modern development.

Phase 9: Reporting, Dashboards & Security Review

👉 **Goal:** Monitor business & secure data.

1. **Reports**
    - **Revenue by Client:** A summary report that groups all invoices by the client's Account Name and sums the Total Amount to show which clients are most valuable.
    - **Project Status Report:** A simple tabular report that lists all projects and shows their current status (e.g., In Progress, Completed) for an at-a-glance view of the current workload.

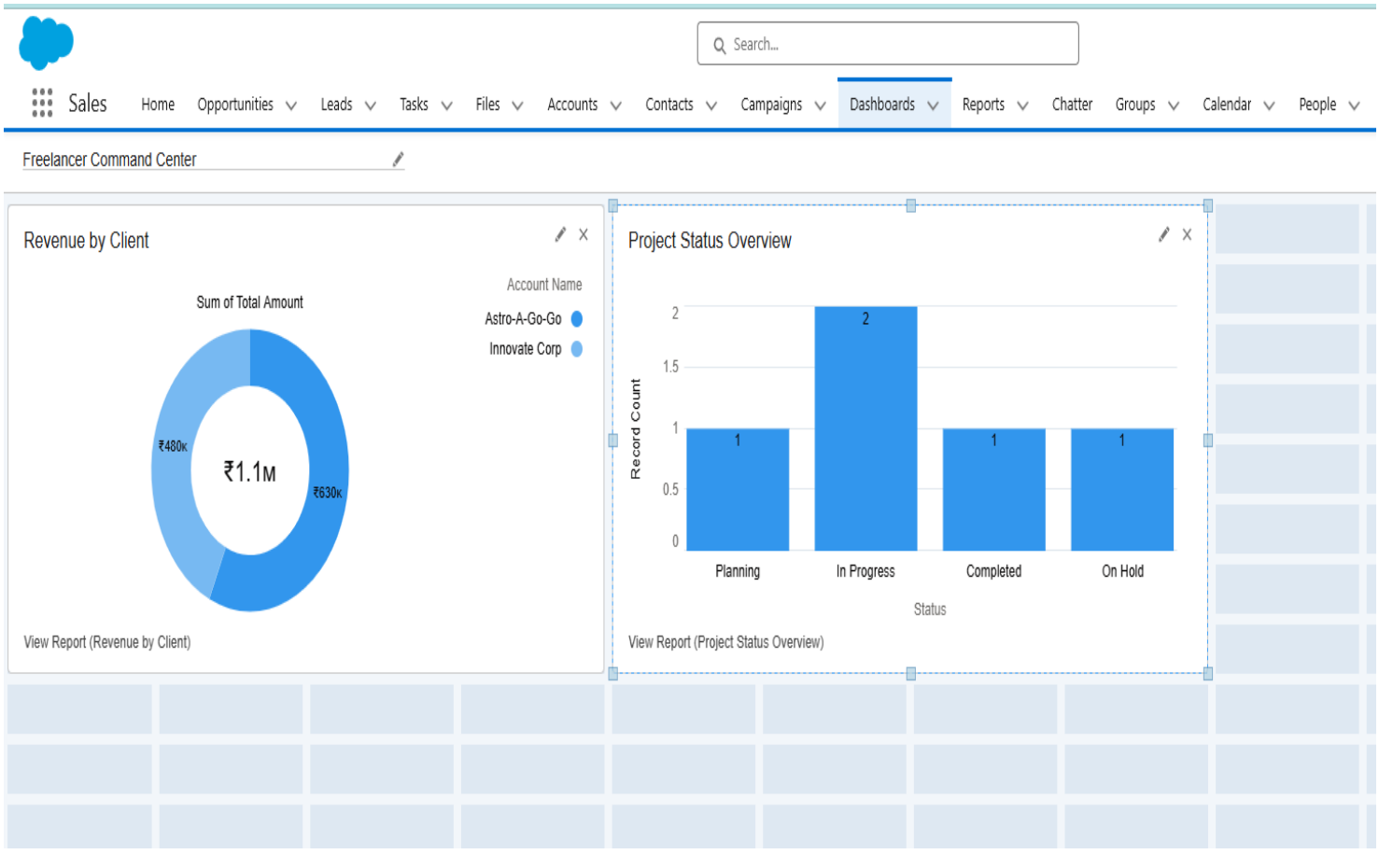


2. **Report Types**

   - **Custom Report Type:** A custom report type named "Clients with Invoices" was created. It links the Account object to its child Invoice records, making the "Revenue by Client" report possible.

3. **Dashboards**

   - **Freelancer Command Center:** A central dashboard was created to provide a high-level overview of the business. It features two components: a donut chart visualizing revenue by client and a Vertical Bar chart showing the breakdown of all projects by their status.



4. **Dynamic Dashboards**
    - This feature is not required for a single-user system. If the freelancer were to hire a team, a dynamic dashboard could be configured so that each team member sees only the data related to their own projects.
5. **Sharing Settings**
    - The Organization-Wide Defaults for Account and Project__c have been confirmed as **Private**, ensuring all client and project data is secure and not visible to anyone else by default.
6. **Field Level Security**
    - Field Level Security (FLS) was reviewed. For example, if a virtual assistant were hired, FLS could be used to hide sensitive financial fields like Hourly Rate from their profile while still allowing them to see other project details.
7. **Session Settings**
    - The default session settings were reviewed. The standard 2-hour timeout is appropriate for a single-user org but could be shortened for enhanced security.
8. **Login IP Ranges**
    - The ability to set Login IP Ranges on a profile was reviewed. As a security best practice, the freelancer could restrict logins to their specific home or office IP address, blocking any access attempts from other locations.
9. **Audit Trail**
    - The Setup Audit Trail was reviewed. It provides a complete log of all administrative changes made to the org, which is critical for security and for troubleshooting any configuration issues.

Phase 10: Final Presentation & Demo Day

👉 **Goal:** Wrap it up like a real project delivery.

1. **Pitch Presentation**
    - A slide deck was prepared to outline the project's story:
        - **Problem:** The inefficiency and risk of errors from using disconnected spreadsheets and documents to manage a freelance business.

- **Solution:** The Freelance-Force-Suite, a single, centralized application for all business operations.
- **Benefits:** Increased efficiency, accurate revenue tracking, professional invoicing, and data-driven business insights.

2. **Demo Walkthrough**

    - A live demo script was prepared to showcase a "day in the life" of the freelancer:
        1. Start on the **Dashboard** to get a high-level overview of revenue and project status.
        2. Open an in-progress **Project** record.
        3. Use the custom **timeLogger LWC** on the "Log Time" tab to quickly add new billable hours.
        4. Show the result of the **"Invoice Creation Automation" Flow** by creating a new invoice and showing how the total amount is calculated automatically.
        5. Demonstrate the **"Revenue by Client" Report** to show how the data provides valuable business insights.

3. **Handoff Documentation**

    - A simple user guide was created, explaining how to perform key tasks like creating a new client, logging time, and generating an invoice. An admin guide was also drafted, outlining the key custom objects and automations.

4. **Portfolio Project Showcase**

    - A professional summary was written for portfolio use:

        - **Title:** Salesforce Freelancer CRM (Freelance-Force-Suite)
        - **Description:** "Designed, developed, and deployed a custom CRM on the Salesforce platform to solve the core business challenges of a freelancer. The application centralizes client and project management, automates time tracking and invoicing, and provides key business insights through custom reports and dashboards."
        - **Skills:** Salesforce Administration, Apex, Lightning Web Components (LWC), SOQL, Process Automation (Flow), Data Modeling, Git, SFDX.
        - **Link:** https://github.com/YashGiri14/Freelance-Force-Suite

---