

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.

The screenshot shows the 'Company Information' setup page. The left sidebar is collapsed, showing sections like Company Settings, Business Hours, Calendar Settings, and Company Information (which is selected). The main content area displays organization details for 'Innovate Digital Solutions'. Key fields include:

- Organization Name:** Innovate Digital Solutions
- Primary Contact:** Yash Giri
- Address:** 123 Tech Park Road, Bengaluru 560001, Karnataka, India
- Division:** None
- Phone:** None
- Fax:** None
- Default Locale:** English (India)
- Default Language:** English
- Fiscal Year Starts In:** January
- Activate Multiple Currencies:**
- Enable Data Translation:**
- Newsletter:**
- Admin Newsletter:**
- Hide Notices About System Maintenance:**
- Hide Notices About System Downtime:**
- Locale Formats:** ICU
- Default Time Zone:** (GMT+05:30) India Standard Time (Asia/Kolkata)
- Currency Locale:** English (India) - INR
- Used Data Space:** 458 KB (9%) [View]
- Used File Space:** 20 KB (0%) [View]
- API Requests, Last 24 Hours:** 0 (15,000 max)
- Streaming API Events, Last 24 Hours:** 0 (10,000 max)
- Restricted Logins, Current Month:** 0 (0 max)
- Salesforce.com Organization ID:** 00DQy00000Wz0gb
- Organization Edition:** Developer Edition
- Instance:** SWE42

At the bottom, it shows 'Created By' Yash Giri on 25/08/2025, 11:26 am and 'Modified By' Yash Giri on 25/09/2025, 2:26 pm. There are 'Edit' and 'Deactivate Org' buttons at the bottom.

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.

The screenshot shows the 'Business Hours Detail' setup page. It lists standard business hours from Monday to Friday from 9:00 am to 6:00 pm. The 'Time Zone' is set to (GMT+05:30) India Standard Time (Asia/Kolkata). The 'Default Business Hours' checkbox is checked. At the bottom, it shows 'Active' checked, 'Created By' Yash Giri on 25/09/2025, 2:33 pm, and 'Last Modified By' Yash Giri on 25/09/2025, 2:33 pm. There is also a 'Holidays (0)' link.

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.

The screenshot shows the User Detail page for a user named 'Yash Giri'. The page includes a navigation bar with links like 'Permission Set Assignments', 'Managers in the Role Hierarchy', 'OAuth Apps', 'Third-Party Account Links', 'Built-In Authenticators', 'Installed Mobile Apps', 'Authentication Settings for External Systems', 'Login History', and 'User Provisioning Accounts'. Below the navigation is a 'User Detail' section with tabs for 'Edit', 'Sharing', 'Change Password', and 'View Summary'. The 'Edit' tab is selected. The user's details are listed in two columns:

Name	Yash Giri	Role	Freelancer
Alias	YGiri	User License	Salesforce
Email	yash_giri.cs22@gkits.net [Verified]	Profile	System Administrator
Username	yash@gyanganaga.com	Active	<input checked="" type="checkbox"/>
Nickname	User17561014128626755239	Marketing User	<input checked="" type="checkbox"/>
Title		Offline User	<input checked="" type="checkbox"/>
Company	Gyan Ganga Institute of Technology and Sciences	Knowledge User	<input type="checkbox"/>
Department		Flow User	<input type="checkbox"/>
Division		Service Cloud User	<input checked="" type="checkbox"/>
Address	IN	Site.com Contributor User	<input type="checkbox"/>
Time Zone	(GMT+05:30) India Standard Time (Asia/Kolkata)	Site.com Publisher User	<input type="checkbox"/>
Locale	English (India)	WDC User	<input type="checkbox"/>
Language	English	Mobile Push Registrations	View
Delegated Approver		Data.com User Type	View
Manager		Accessibility Mode (Classic Only)	<input type="checkbox"/> i
Receive Approval Request Emails	Only if I am an approver	Debug Mode	<input type="checkbox"/> i
Federation ID		High-Contrast Palette on Charts	<input type="checkbox"/> i
App Registration: One-Time Password	[Connect] i	Load Lightning Pages While Scrolling	<input checked="" type="checkbox"/> i

8. OWD (Org-Wide Defaults)

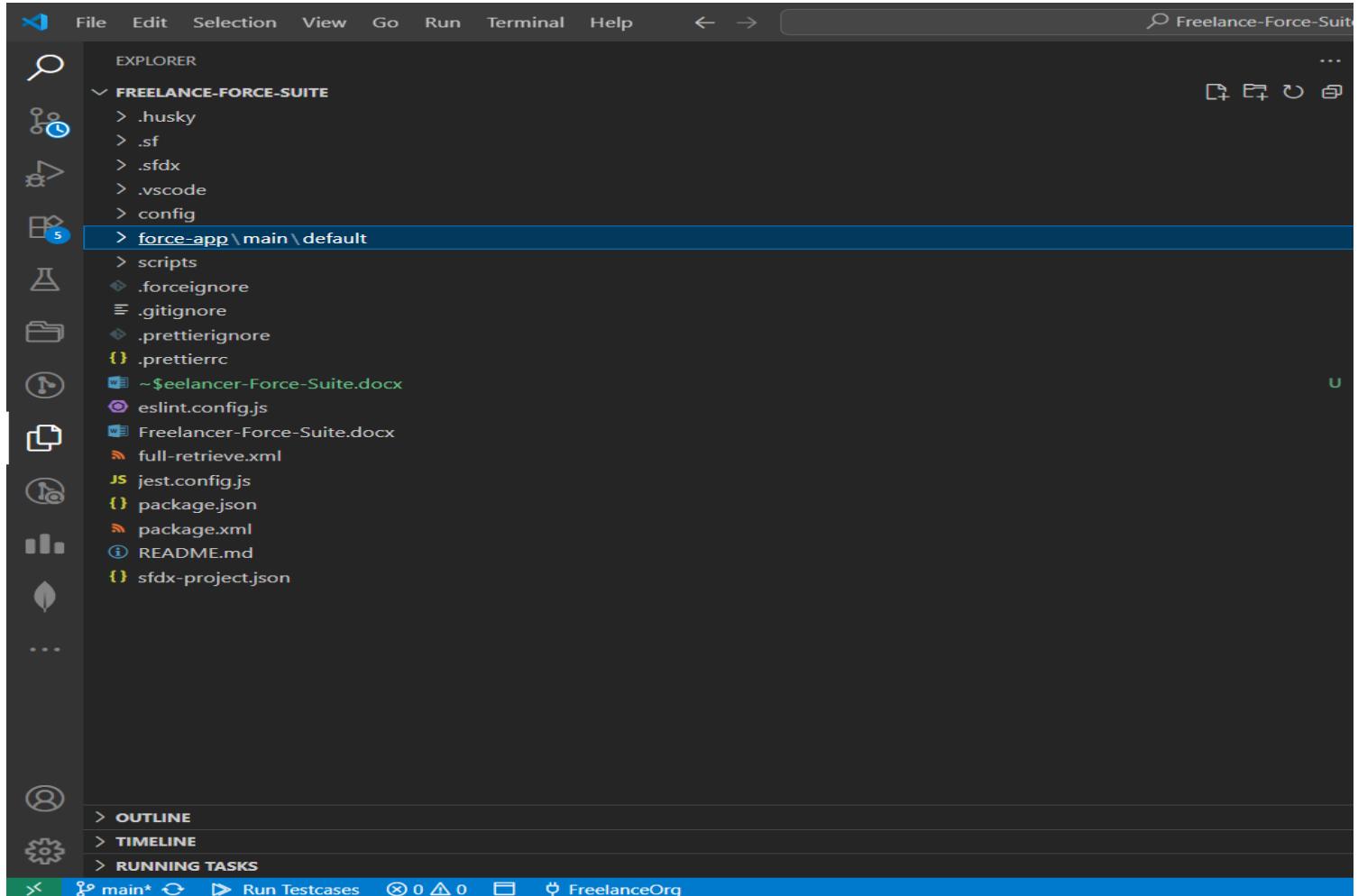
- The Organization-Wide Defaults for all key standard objects (Account, Opportunity) and custom objects (Project__c) have been set to **Private** to ensure maximum data security.

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.



The screenshot shows the GitHub repository page for 'Freelance-Force-Suite' (Public). The repository details are as follows:

- Owner:** YashGiri14
- Name:** Freelance-Force-Suite
- Branches:** 1 Branch
- Tags:** 0 Tags
- Last Commit:** Phase 10 completed by YashGiri14, 3 days ago, 18 Commits
- Code Tab:** Active tab
- Issues:** 0 Issues
- Pull requests:** 0 Pull requests
- Actions:** 0 Actions
- Projects:** 0 Projects
- Wiki:** 0 Wiki
- Security:** 0 Security
- Insights:** Insights
- Settings:** Settings

Code Overview:

File	Description	Age
.husky	feat: Created initial Project custom object	2 weeks ago
.vscode	feat: Created initial Project custom object	2 weeks ago
config	feat: Created initial Project custom object	2 weeks ago
force-app/main/default	Phase 9 completed.	3 days ago
scripts	feat: Created initial Project custom object	2 weeks ago
.forceignore	feat: Created initial Project custom object	2 weeks ago
.gitignore	feat: Created initial Project custom object	2 weeks ago
.prettierignore	feat: Created initial Project custom object	2 weeks ago
.prettierrc	feat: Created initial Project custom object	2 weeks ago
Freelancer-Force-Suite.docx	Phase 10 completed	3 days ago
README.md	Update README.md	2 weeks ago
eslint.config.js	feat: Created initial Project custom object	2 weeks ago
full-retrieve.xml	Phase 4 completed	last week
jest.config.js	feat: Created initial Project custom object	2 weeks ago

Repository Details:

- About:** An open-source Salesforce application designed to help freelancers and solopreneurs manage their projects, clients, and finances in one centralized hub. This project replaces disconnected tools with a single source of truth to enhance productivity and profitability.
- Readme:** Readme
- Activity:** 0 stars
- Watching:** 0 watching
- Forks:** 0 forks
- Releases:** No releases published. Create a new release
- Packages:** No packages published. Publish your first package
- Languages:** [Progress bar]

Phase 3 : Data Modeling & Relationships

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

1. Standard & Custom Objects

o Standard Objects:

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

o 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

o 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

o 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".

o 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

- o 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.

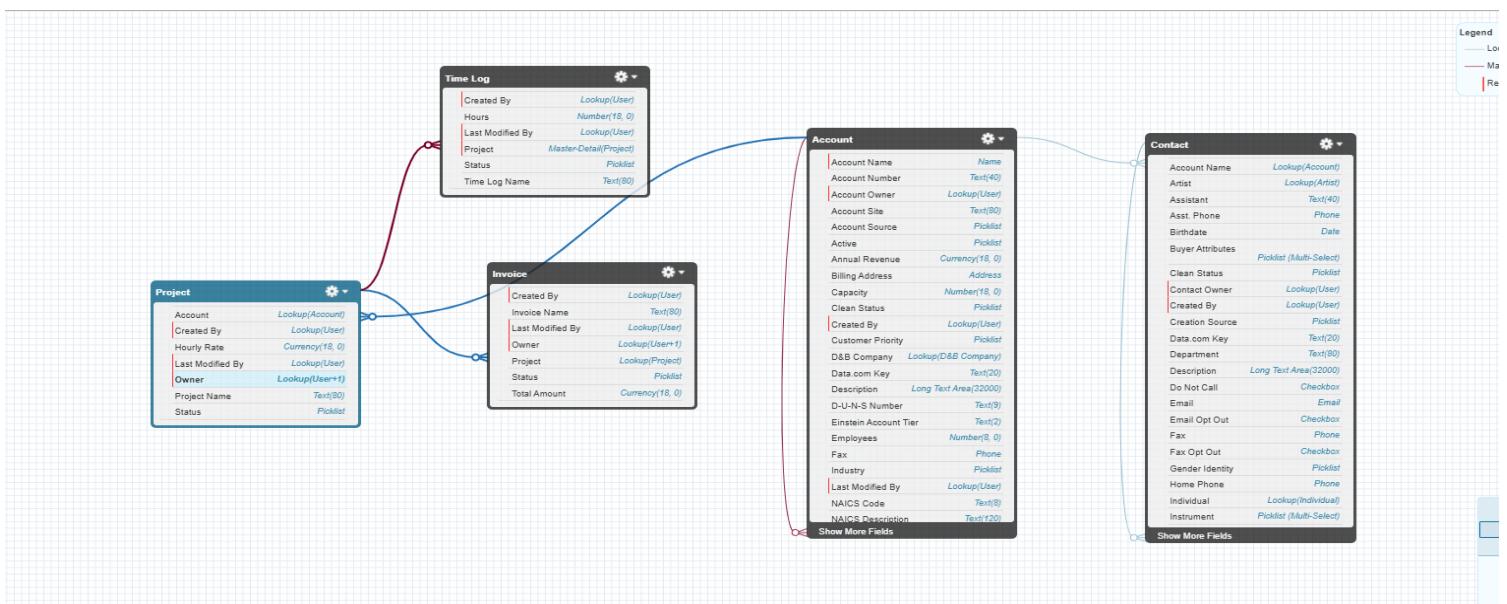
This screenshot shows a Salesforce Project page for 'Website Redesign for Innovate Corp'. The top navigation bar includes Sales, Home, Opportunities, Leads, Tasks, Files, Accounts, Contacts, Campaigns, Dashboards, Reports, Chatter, and a search bar. The page title is 'Project Website Redesign for Innovate Corp'. The 'Related' section contains two items: 'Invoices (0)' and 'Time Logs (0)'. The 'Activity' section shows a toolbar with various icons for creating and managing activities, and a message stating 'No activities to show. Get started by sending an email, scheduling a task, and more.' A note also says 'No past activity. Past meetings and tasks marked as done show up here.'

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.

Time Log Validation Rule

[Back to Time Log](#)

Validation Rule Detail		Edit	Clone	Active	<input checked="" type="checkbox"/>
Rule Name	Date_Cannot_Be_In_Future				
Error Condition Formula	Date__c > TODAY()				
Error Message	You cannot log time for a future date. Please select today's date or a past date.			Error Location	Date
Description					
Created By	Yash Giri, 25/09/2025, 3:31 pm			Modified By	Yash Giri, 25/09/2025, 3:31 pm
		Edit	Clone		

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)

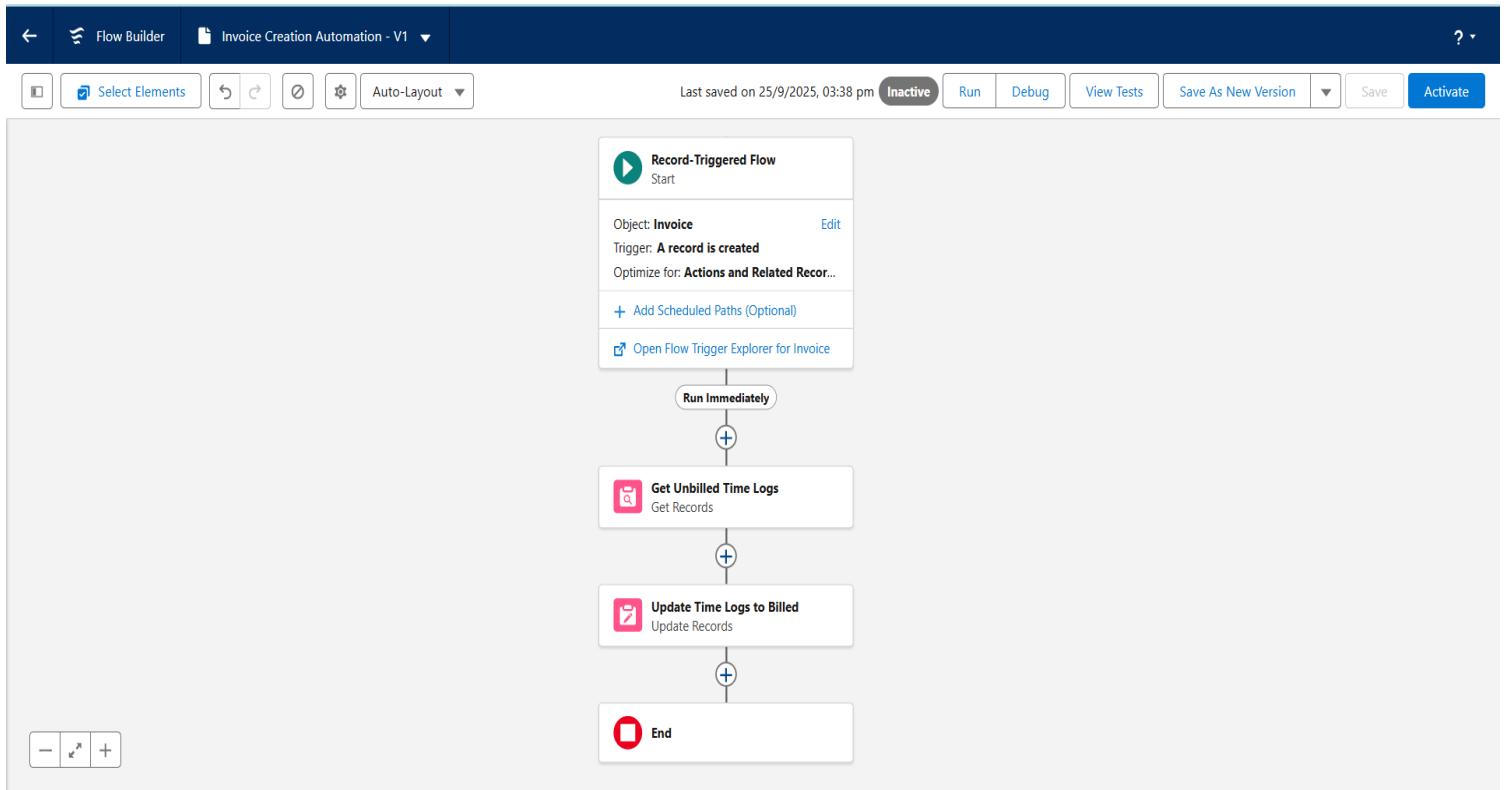
- This is another legacy tool that has been superseded by Flow Builder, which is used for all record-triggered automations in this project.

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

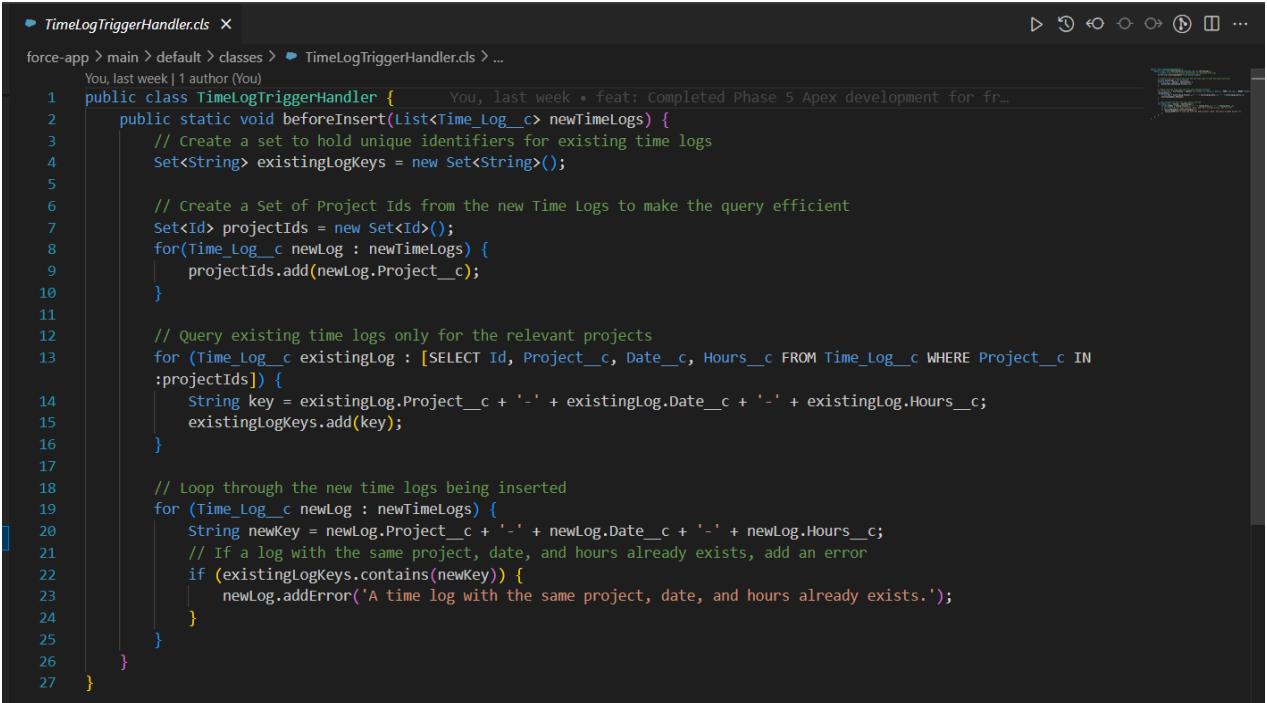
- Created a TimeLogTriggerHandler Apex class to contain the business logic for preventing duplicate time entries.
- Created a MarkInvoicesOverdueScheduled class to handle the logic for a nightly batch job.

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.



The screenshot shows a code editor window with the file 'TimeLogTriggerHandler.cls' open. The code is written in Apex and implements a trigger handler for the 'Time_Log__c' object. It uses a handler pattern where all logic is contained within the 'beforeInsert' method of the 'TimeLogTriggerHandler' class. The code queries existing time logs for relevant projects and checks if the new logs would result in duplicates. If a duplicate is found, it adds an error message to the log record.

```
• TimeLogTriggerHandler.cls •
force-app > main > default > classes > • TimeLogTriggerHandler.cls ...
You, last week | 1 author (You)
1 public class TimeLogTriggerHandler {
2     public static void beforeInsert(List<Time_Log__c> newTimeLogs) {
3         // Create a set to hold unique identifiers for existing time logs
4         Set<String> existingLogKeys = new Set<String>();
5
6         // Create a Set of Project IDs from the new Time Logs to make the query efficient
7         Set<Id> projectIds = new Set<Id>();
8         for(Time_Log__c newLog : newTimeLogs) {
9             projectIds.add(newLog.Project__c);
10        }
11
12        // Query existing time logs only for the relevant projects
13        for (Time_Log__c existingLog : [SELECT Id, Project__c, Date__c, Hours__c FROM Time_Log__c WHERE Project__c IN :projectIds]) {
14            String key = existingLog.Project__c + '-' + existingLog.Date__c + '-' + existingLog.Hours__c;
15            existingLogKeys.add(key);
16        }
17
18        // Loop through the new time logs being inserted
19        for (Time_Log__c newLog : newTimeLogs) {
20            String newKey = newLog.Project__c + '-' + newLog.Date__c + '-' + newLog.Hours__c;
21            // If a log with the same project, date, and hours already exists, add an error
22            if (existingLogKeys.contains(newKey)) {
23                newLog.addError('A time log with the same project, date, and hours already exists.');
24            }
25        }
26    }
27 }
```

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 MarkInvoicesOverdueScheduled 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."

The screenshot shows the 'Apex Classes' section under 'SETUP'. A new scheduled job is being created with the following details:

- Job Name:** Mark Overdue Invoices Nightly
- Apex Class:** MarkInvoicesOverdueSched
- Schedule Using:** Schedule Builder
- Frequency:** Weekly (selected)
- Recurs every week on:** Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday (all checked)
- Start:** 25/09/2025 (25/09/2025)
- End:** 25/10/2025 (25/09/2025)
- Preferred Start Time:** 3:00 am

All Scheduled Jobs

Help for this Page ?

The All Scheduled Jobs page lists all of the jobs scheduled by your users. Multiple job types may display on this page. You can delete scheduled jobs if you have the permission to do so.

Percentage of Scheduled Jobs Used: 1%
You have currently used 1 scheduled Apex jobs out of an allowed organization limit of 100 active or scheduled jobs. To learn about how this limit is calculated and what contributes to it see the [Lightning Platform Apex Limits](#) topic.

View: [All Scheduled Jobs](#) [Create New View](#)

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other | All

Schedule Apex							
Action	Job Name ↑	Submitted By	Submitted	Started	Next Scheduled Run	Type	Cron Trigger ID
Manage Del Pause Job	Mark Overdue Invoices Nightly	Giri_Yash	25/09/2025, 4:17 pm		26/09/2025, 3:00 am	Scheduled Apex	08eQy00000QSpPX
Del	Metalytics Data Loader Job for Org : 00DQy0000Wz0gb	User_Integration	25/08/2025, 11:28 am	24/09/2025, 10:41 pm	25/09/2025, 10:41 pm	Autonomous Data Loader Job	08eQy00000On8RB

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other | All

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, MarkInvoicesOverdueScheduledTest) for all Apex code to ensure it functions as expected and meets Salesforce's code coverage requirements for deployment.

```
MarkInvoicesOverdueScheduledTest.cls X

force-app > main > default > classes > MarkInvoicesOverdueScheduledTest.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 MarkInvoicesOverdueScheduledTest {
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 MarkInvoicesOverdueScheduled());
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)
  1 @isTest
  2     You, last week + 6 test: Completed Phase 5 Apex development for fr...
  3 Run All Tests | Debug All Tests | You, last week | 1 author (You)
  2 private class TimeLogTriggerHandlerTest {
  3     @isTest
  4         RunTest|Debug Test
  5 static void testDuplicateTimeLogPrevention() {
  6     // Create test data
  7     Account client = new Account(Name = 'Test Client');
  8     insert client;
  9
 10    // CORRECTED: Removed the "Name" field assignment.
 11    // Salesforce will generate the auto-number name automatically.
 12    Project__c project = new Project__c(client_c = client.Id);
 13    insert project;
 14
 15    Time_Log__c initialLog = new Time_Log__c(
 16        Project__c = project.Id,
 17        Date__c = Date.today(),
 18        Hours__c = 2.5
 19    );
 20    insert initialLog;
 21
 22    // Start the test
 23    Test.startTest();
 24
 25    // Attempt to insert a duplicate time log
 26    Time_Log__c duplicateLog = new Time_Log__c(
 27        Project__c = project.Id,
 28        Date__c = Date.today(),
 29        Hours__c = 2.5
 30    );
 31
 32    // Use a try-catch block to verify that the trigger throws an error
 33    try {
 34        insert duplicateLog;
 35        // If the insert succeeds, the test should fail
 36        System.assert(False, 'The trigger should have prevented the duplicate insert.');
 37    } catch (DmlException e) {
 38        // Assert that the error message is the one we expect from our trigger
 39        System.assert(e.getMessage().contains('A time log with the same project, date, and hours already exists.'), 
 40                     'The error message from the trigger was not found.');
 41    }
 42
 43    Test.stopTest();
 44 }

```

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

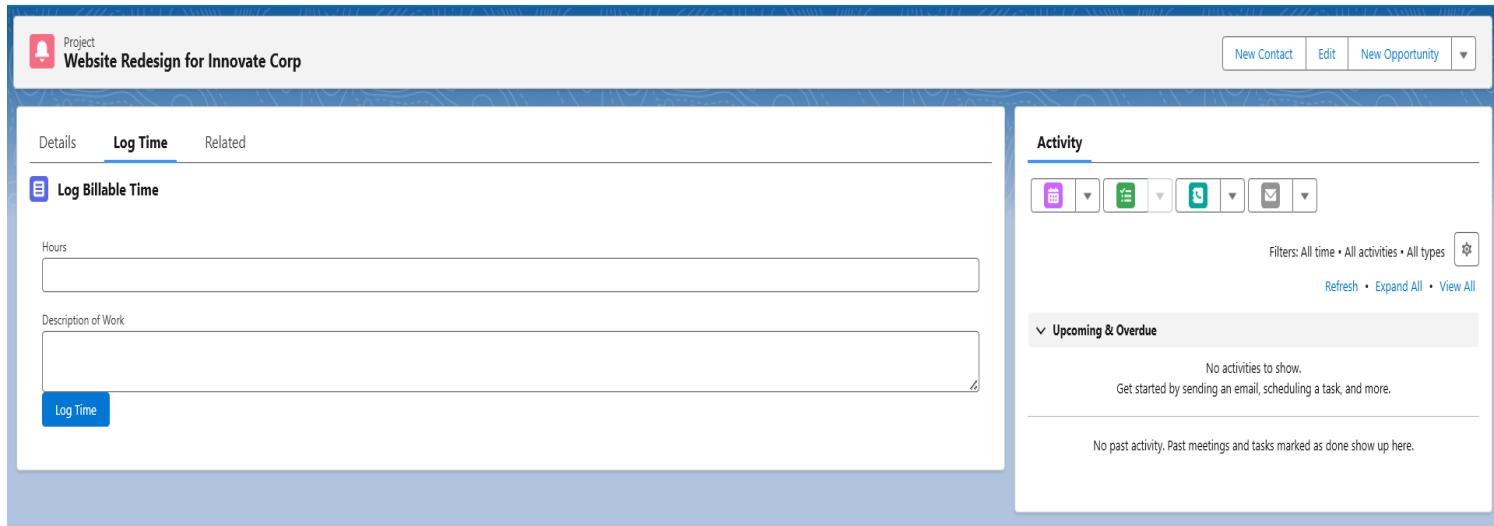
- The home page has not been customized yet but could be enhanced in a later phase with dashboards showing revenue and project status.

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

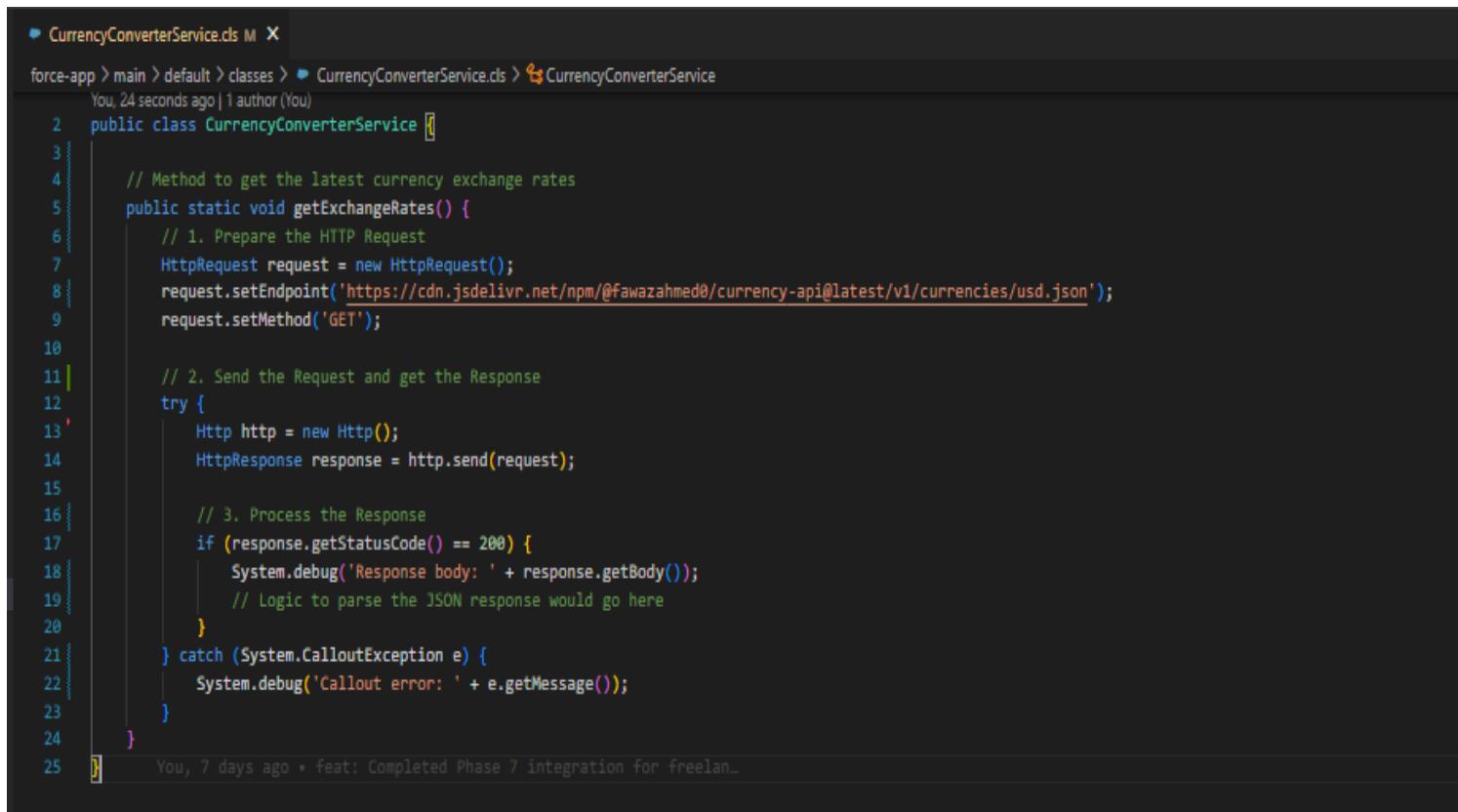
- Named Credentials are not used in this phase but are a best practice for securely storing the URL and authentication details for an API endpoint.

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.



The screenshot shows the Salesforce code editor with the file 'CurrencyConverterService.cls' open. The code implements a static method 'getExchangeRates()' that performs an HTTP GET request to a currency API endpoint. The response is processed to debug the body. The code is annotated with line numbers from 2 to 25.

```
1  public class CurrencyConverterService {
2
3     // Method to get the latest currency exchange rates
4     public static void getExchangeRates() {
5         // 1. Prepare the HTTP Request
6         HttpRequest request = new HttpRequest();
7         request.setEndpoint('https://cdn.jsdelivr.net/npm/@fawazahmed0/currency-api@latest/v1/currencies/usd.json');
8         request.setMethod('GET');
9
10        // 2. Send the Request and get the Response
11        try {
12            Http http = new Http();
13            HttpResponse response = http.send(request);
14
15            // 3. Process the Response
16            if (response.getStatusCode() == 200) {
17                System.debug('Response body: ' + response.getBody());
18                // Logic to parse the JSON response would go here
19            }
20        } catch (System.CalloutException e) {
21            System.debug('Callout error: ' + e.getMessage());
22        }
23    }
24
25 }
```

4. Remote Site Settings

- To allow Salesforce to make the callout, the API's base URL (<https://cdn.jsdelivr.net/npm/@fawazahmed0/currency-api@latest/v1/currencies/usd.json>) was added to the **Remote Site Settings**. This is a mandatory security step to authorize the external domain.



SETUP

Remote Site Settings

Remote Site Details

Remote Site Detail

[Edit](#) [Delete](#) [Clone](#)Remote Site Name **Currency_Exchange_API**Modified By [Yash Giri](#), 25/09/2025, 4:39 pmRemote Site URL <https://cdn.jsdelivr.net>Disable Protocol Security

Description

Active Created By [Yash Giri](#), 25/09/2025, 4:39 pm[Edit](#) [Delete](#) [Clone](#)

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.

The screenshot shows the 'Duplicate Rules' page in the Salesforce Setup. The top navigation bar has a blue header with the word 'SETUP'. Below it, there's a sub-header 'd Duplicate Rules'. The main content area is titled 'Account Duplicate Rule' and shows a single rule named 'Block_Duplicate_Clients'. The rule details are as follows:

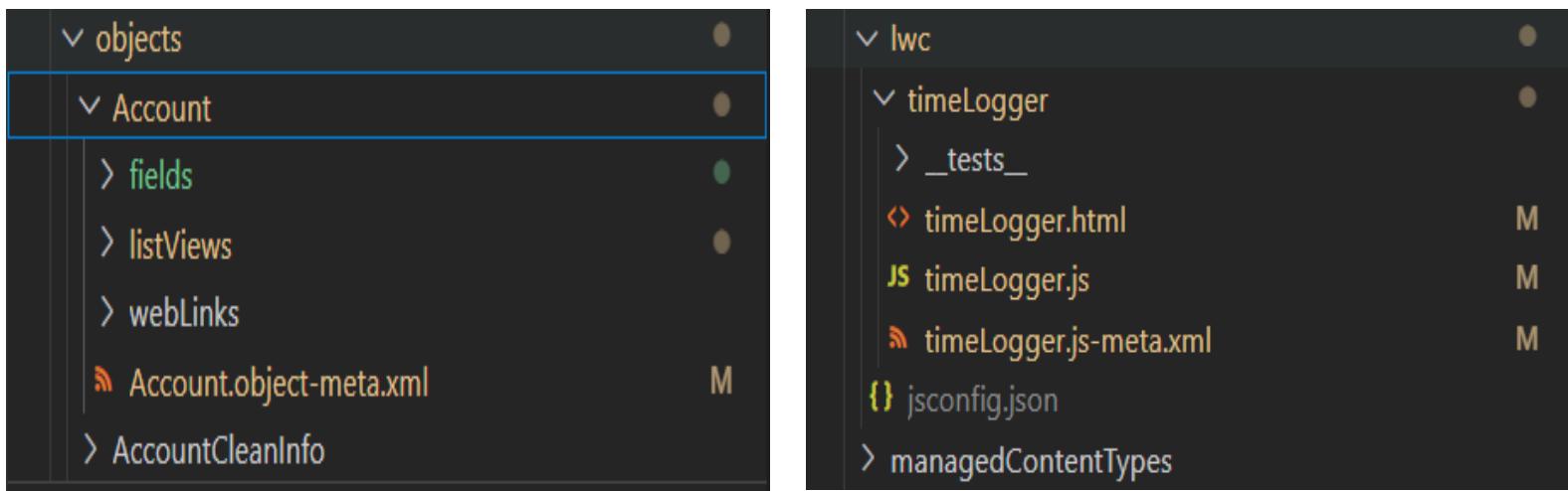
Duplicate Rule Detail		Operations			
Rule Name	Block_Duplicate_Clients	Edit Delete Clone Deactivate			
Description		Order 2 of 2 [Reorder]			
Object	Account				
Record-Level Security	Enforce sharing rules				
Action On Create	Block	Operations On Create <input type="checkbox"/> Alert <input type="checkbox"/> Report			
Action On Edit	Allow	Operations On Edit <input type="checkbox"/> Alert <input type="checkbox"/> Report			
Alert Text	Use one of these records?				
Active	<input checked="" type="checkbox"/>				
Matching Rule	<input checked="" type="checkbox"/> Standard Account Matching Rule <input checked="" type="checkbox"/> Mapped	Matching Criteria Matching rule for account records. More info			
Conditions					
Created By	Yash_Giri 25/09/2025, 4:44 pm	Modified By Yash_Giri 25/09/2025, 4:44 pm			

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.



✓	classes	
↳	CurrencyConverterService.cls	M
↳	CurrencyConverterService.cls-meta.xml	
↳	CurrencyConverterServiceTest.cls	
↳	CurrencyConverterServiceTest.cls-meta.xml	
↳	MarkInvoicesOverdueScheduled.cls	M
↳	MarkInvoicesOverdueScheduled.cls-meta.xml	
↳	MarkInvoicesOverdueScheduledTest.cls	
↳	MarkInvoicesOverdueScheduledTest.cls-meta.xml	
↳	TimeLogController.cls	M
↳	TimeLogController.cls-meta.xml	
↳	TimeLogTriggerHandler.cls	
↳	TimeLogTriggerHandler.cls-meta.xml	
↳	TimeLogTriggerHandlerTest.cls	
↳	TimeLogTriggerHandlerTest.cls-meta.xml	
>	cleanDataServices	
>	communities	
>	contentassets	

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.

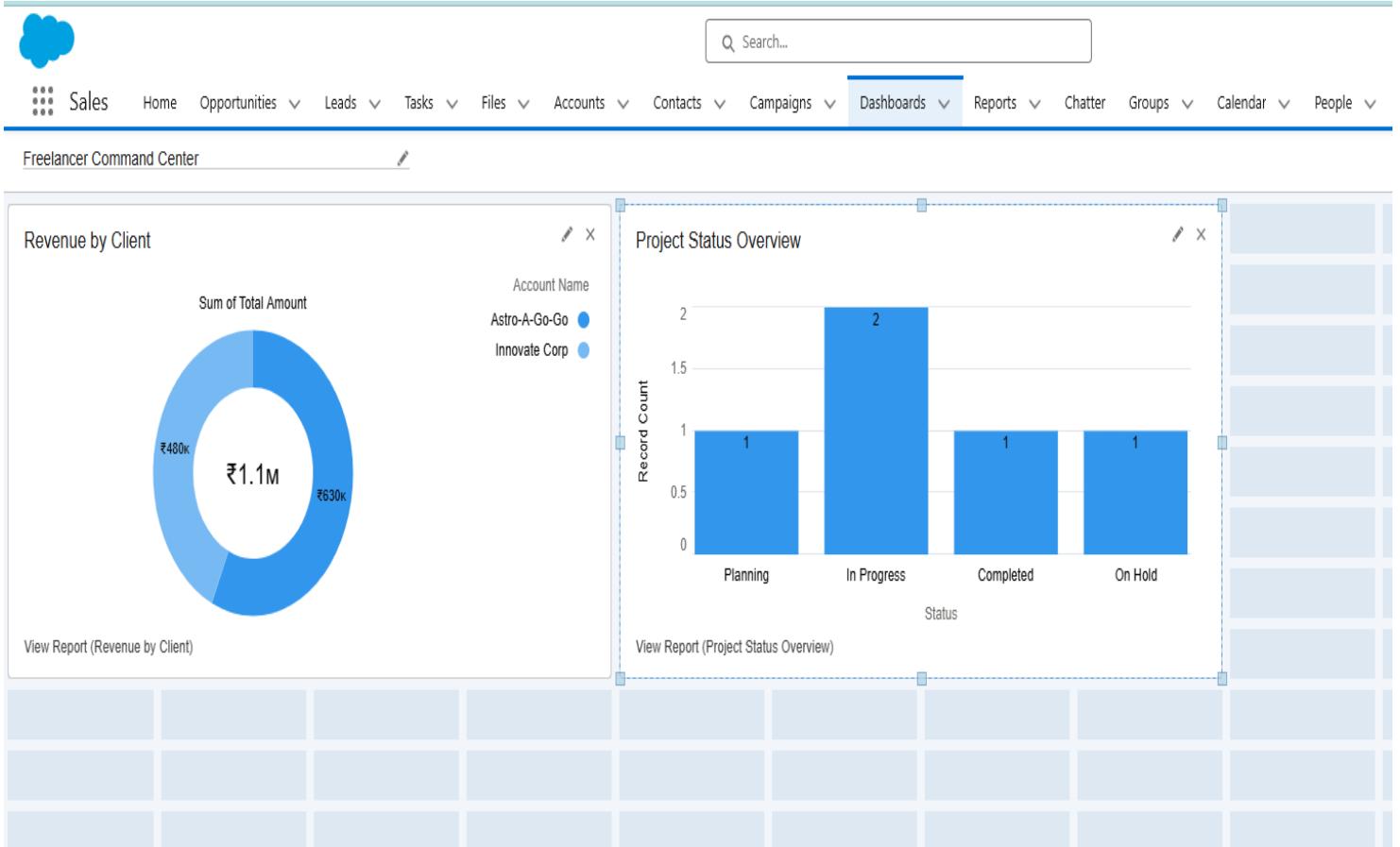
Report: Clients with Invoices			
Revenue by Client			
Total Records		Total Total Amount	
5		₹11,10,000	
Account Name	Project Name	Invoice Name	Total Amount
Astro-A-Go-Go (3)	Marketing Analytics Setup	Invoice 4	₹50,000
	Website Redesign	Invoice 1	₹2,50,000
	CRM Data Migration	Invoice 3	₹3,30,000
Subtotal			₹6,30,000
Innovate Corp (2)	Mobile App Development	Invoice 2	₹4,80,000
	Website Redesign for Innovate Corp	-	-
Subtotal			₹4,80,000
Total (5)			₹11,10,000

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>
