Project Documentation: PharmaForce Salesforce Implementation

General Overview

Project name: PharmaForce Urgent Opportunities Component Implementation

Client: PharmaForce

Industry: Pharmaceutics

Executive Summary: PharmaForce required a custom solution to manage their sales opportunities effectively, with an emphasis on highlighting urgent opportunities, thus facilitating seamless team collaboration. The implemented solution enhances the Opportunity management process by providing visibility, notifications, and streamlined data entry directly on the Account Record Page.

Business-Related Details

Requirements

1. Urgent Opportunities Management:
   - Highlight urgent opportunities using a dedicated flag on the Opportunity object.
   - Allow team members to view and create urgent opportunities within the Account Record Page.
2. Notifications:
   - Notify team members when an urgent opportunity is created.
   - Notify the creator of the urgent opportunity with a success message.
3. Ease of use:
   - Provide a user-friendly interface for creating and managing urgent opportunities.
   - Maintain data accuracy by automatically linking new opportunities to their respective accounts.

Functional Features

- Urgent Opportunities Table: is designed to display all urgent opportunities related to specific account, includes the Opportunity Name, Stage, Amount, and Close Date fields as data table columns and a “New opportunity” button for quick creation.
- Custom Modal for Opportunity Creation: enables users to input the Opportunity Name, Stage, Amount and Close Date fields and includes “Cancel” and “Save” buttons.
- Automated Notifications: account team members are notified about new urgent opportunities and the creator receives feedback.

Technical Details

1. Setting Up the Git Repository and Branch Management

- Created a Git repository: “PharmaForce-Urgent-Opportunities-Component-Implementation”;
- Cloned the repository locally:

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\b.gemil> git clone https://github.com/bgemil/PharmaForce-Urgent-Opportunities-Component-Implementation.git
```

```
PS C:\Users\b.gemil> cd PharmaForce-Urgent-Opportunities-Component-Implementation
PS C:\Users\b.gemil\PharmaForce-Urgent-Opportunities-Component-Implementation>
```

- Created a new branch for the first ticket:

```
PS C:\Users\b.gemil\PharmaForce-Urgent-Opportunities-Component-Implementation> git checkout -b feature/ONBD-41
```

- Created a new SFDX project in Visual Studio Code:

```
PROBLEMS 9   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\b.gemil\PharmaForce-Urgent-Opportunities-Component-Implementation> sfdx force:project:create -n
```

- Staged and committed the changes:

```
PS C:\Users\b.gemil\PharmaForce-Urgent-Opportunities-Component-Implementation> git add .
```

```
PS C:\Users\b.gemil\PharmaForce-Urgent-Opportunities-Component-Implementation> git commit -m "Set Up the Git Repo & Bran
ch Management"
```

- Pushed the branch to the remote repository:

```
PS C:\Users\b.gemil\PharmaForce-Urgent-Opportunities-Component-Implementation> git push origin feature/ONBD-41
```

- Finally, created a pull request from the GitHub interface.

2. Creating the User and Assigning the Custom Permission Set in Salesforce

- Created the “**PharmaForce Urgent Opps Admin**” permission set.
- Granted object and field-level permissions for the Opportunity object in said set.

- Created a user (**StdTestUser**) with the Standard Platform User profile.
- Assigned the permission set to the newly created user.

3. Customizing the Salesforce Org with the Company Branding

- Uploaded and applied the company logo.
- Modified the custom color theme.
- Created a new app: **PharmaForce**.

4. Developing the Urgent Opportunities Datatable Component with Search and Pagination

- Created the `UrgentOpportunitiesController` Apex class, including two methods with the following signatures:
  - `public static List<Opportunity> fetchUrgentOpportunities(String searchKey, Id accountId, Integer offset, Integer limitValue)`, which fetches a specific subset of urgent opportunities based on the `searchKey` and pagination parameters;
  - `public static Integer fetchTotalRecordCount(String searchKey, Id accountId)`, which provides the total number of urgent opportunities matching the search criteria.

- Executed a script in Anonymous Apex to populate the database with urgent opportunities.

- Built the Lightning Web Component: `urgentOpportunitiesDataTable`, which makes calls to the methods from the aforementioned Apex class in order to populate its columns with the necessary data. The following methods were added to implement the search, pagination, and data loading functionalities:
  - `loadOpportunities()`, which calls the methods that fetch and count all the urgent records for the related account and updates the component's data properties (`opportunities` and `totalRecords`) or displays an error toast if an exception occurs;
  - `handleSearch(event)`, which captures the search key from the event's input field, resets the offset for pagination and reloads the opportunities by calling `loadOpportunities()`;
  - `handlePrevious()` and `handleNext()`, which modify the offset by the `limitValue` accordingly for pagination purposes;

- Added the 'New Opportunity' button.

- Created a Custom Modal Component: `createUrgentOpportunity`, which allows the user to input the data needed to create a new urgent opportunity. The implemented handler methods are:
  - `handleFieldChange(event)`, which manages changes in the input fields and updates the corresponding property with the new value;
  - `handleCancel()`, which dispatches a cancel event to notify the parent component that the modal should be closed without saving changes;

- `handleSave()`, which validates that all required fields are populated; if validation passes, it creates an opportunity object with the user-provided data and then dispatches a save event with the opportunity details as part of the event's detail property;

- Added a method in the `UrgentOpportunitiesController` class:
  - `public static void saveNewOpportunity(Opportunity opportunity)`, in order to call it from the `UrgentOpportunitiesDataTable`;

- Updated **UrgentOpportunitiesDataTable.js** by adding:
  - `@track isModalOpen = false;`
  - `handleNewOpportunity()`, which opens the modal for creating a new urgent opportunity by setting the `isModalOpen` variable to `true`;
  - `handleCancel()`, which closes the new urgent opportunity modal by setting the `isModalOpen` variable to `false`;
  - `handleModalSave(event)`, which manages the logic for saving a new urgent opportunity.

5. Implementing Notifications for Urgent Opportunities

- Created a new Custom Notification in Salesforce.
- Created a new Apex supporting class: `NotificationsAndEmailsSender`, which encapsulates the logic for sending notifications and preparing email messages.
- Created a new Apex trigger: `NewUrgentOpportunityTrigger`, which checks if the creator is part of the account team and notifies them and the other account team members using the `sendNotificationsAndEmails` method from the `NotificationsAndEmailsSender` class.

6. Implementing Error Handling and Creating ErrorLog

- Created the ErrorLog Custom Object.
- Added Custom Fields: ClassName, MethodName, Level, Message, StackTrace.
- Implemented the error logging mechanism by creating the `ErrorHandler` class containing one method, `logError(Exception e, String className, String methodName, String level)`.
- Added error handling to the already built logic by analyzing the reasons the existent classes could fail and by adding **try…catch** blocks accordingly.
- Usage for Debugging:
  - Logged errors provide detailed information, including: the class and method where the error occurred (ClassName__c, MethodName__c) and the exception message and stack trace (Message__c, StackTrace__c);
  - Developers can use this information to reproduce and resolve issues in lower environments before deploying fixes to production;
  - Debugging steps include:
    - Reviewing the Message__c and StackTrace__c fields to identify the root cause.
    - Simulating the error in a sandbox using the same input data.
    - Applying and testing the fix.

7. Writing and Implementing Test Classes

- Analyzed each Apex class separately while keeping the following testing principles in mind: **Right-BICEP**, **CORRECT** and **FIRST**.
- The test class implementation followed this structure:

a) `UrgentOpportunitiesControllerTest`

1. Test setup:
   - created **Account** and urgent **Opportunity** record for testing.
2. Tests:
   - **testFetchTotalRecordCount_WithSearchKey**: tests the functionality of **fetchTotalRecordCount** when a valid **searchKey** is provided;
   - **testFetchUrgentOpportunities_WithSearchKey**: tests **fetchUrgentOpportunities** to retrieve opportunities based on a valid **searchKey**;
   - **testFetchTotalRecordCount_Exception & testFetchUrgentOpportunities_Exception**: test exception handling for the methods when invalid parameters or simulated failure occurs;
   - **testSaveNewOpportunity_Exception**: tests exception handling for **saveNewOpportunity** when invalid data is provided (e.g., Name = null).
3. Edge cases:
   - **null** values for searchKey;
   - the forceFailure property triggers errors to ensure correct exception handling;
   - testSaveNewOpportunity_Exception ensures that required fields are **validated**.
4. **Alignment with principles:**

   Right-BICEP
   - **Right**: Tests validate core controller functionality (record count, fetch logic, error handling).
   - **BICEP**:
     - **Boundary**: Covers null parameters and missing fields.
     - **Inverse**: Simulates failures to test exception handling.
     - **Cross-check**: Compares results with expected outcomes.
     - **Error**: Verifies robust exception handling.

   CORRECT
   - **Conformance**: Matches functionality to business requirements.
   - **Range**: Tests valid and invalid inputs.
   - **Existence**: Ensures proper handling of required fields and exceptions.
   - **Cardinality**: Validates record count against expectations.

   FIRST
   - **Fast**: Tests execute efficiently.
   - **Independent**: Tests don't depend on each other.
   - **Self-validating**: Clear assertions validate outcomes.

b) `NotificationsAndEmailsSenderTest`

1. Test setup:
   - created **Account**, **Opportunity**, **User** and **AccountTeamMember** records.
2. Tests:
   - **testSendNotificationsAndEmails**: validates the functionality of the **sendNotificationsAndEmails** method.
3. Edge cases:
   - **valid** inputs;
   - **empty/invalid** parameters.
4. **Alignment with principles:**

Right-BICEP

- **Right:** Tests core functionality of sending notifications and emails.
- **BICEP:**
  - **Boundary:** Confirms behavior with valid parameters and email list size.
  - **Cross-check:** Compares actual results (email list size) with expected outcomes.

CORRECT

- **Conformance:** Ensures functionality aligns with notification and email logic requirements.
- **Range:** Covers valid inputs but lacks null/invalid input scenarios.
- **Existence:** Validates that an email is added to the list.
- **Cardinality:** Verifies the exact number of emails generated.

FIRST

- **Fast:** Runs efficiently with minimal data setup.
- **Independent:** Operates independently of other tests.
- **Self-validating:** Assertions clearly confirm the result.

c) `NewUrgentOpportunityTriggerTest`

1. Test setup:
   - created **Account**, **User** and **Account Team Member** records.
2. Tests:
   - testTriggerLogic: Validates the core functionality of the NewUrgentOpportunityTrigger;
   - testTrigger_ExceptionHandling: Tests the trigger's exception handling and error logging.
3. Edge cases:
   - **valid** and **invalid** inputs (opportunities);
   - **inappropriate** error logging.
4. **Alignment with principles:**

   Right-BICEP

   - **RIGHT:** Tests both success (testTriggerLogic) and failure (testTrigger_ExceptionHandling) scenarios of the trigger.
   - **BICEP:**
     - **Boundary:** Covers valid and invalid inputs for opportunities.
     - **Inverse:** Checks trigger response to missing fields or invalid configurations.
     - **Cross-check:** Compares actual results with expected outcomes.
     - **Error:** Validates exception handling and error logging.
     - **Performance:** Focuses on correctness rather than performance.

   CORRECT

   - **Conformance:** Ensures the trigger aligns with the expected behavior (urgent opportunities and error handling).
   - **Range:** Tests both valid and invalid opportunity configurations.
   - **Existence:** Validates the presence of error logs for exceptions.
   - **Cardinality:** Verifies single error logs and correct flag settings.

   FIRST

   - **Fast:** Tests execute efficiently with minimal setup.
   - **Independent:** Tests operate without interdependency.
   - **Self-validating:** Assertions clearly validate correctness.

d) `ErrorHandlerTest`

1. Test setup: not applicable here.
2. Tests:
   - **testLogError_Success**: Verifies the functionality of the **logError** method when logging an exception is successful;
   - **testLogError_FailureDuringInsert**: Tests the behavior of **logError** when the ErrorLog__c insertion fails.
3. Edge cases:
   - validates the **standard logging behavior**;
   - tests **insertion failure**;
   - simulates **common exception occurrence**.
4. Alignment with principles:

Right-BICEP

- **Right:** Tests core functionality (logging errors) and failure scenarios (handling insert issues).
- **BICEP:**
  - **Boundary:** Tests normal behavior and restricted insert scenarios.
  - **Cross-check:** Verifies logs against simulated exceptions and provided metadata.
  - **Error:** Confirms that insert failures are handled gracefully.

CORRECT

- **Conformance:** Ensures logging aligns with the expected behavior (accurate logs, no unhandled errors).
- **Range:** Tests both successful and failed log insertions.
- **Existence:** Validates that logs are created only under valid conditions.
- **Cardinality:** Ensures exactly one log is created in success cases and none in failure cases.

FIRST

- **Fast:** Runs quickly with minimal test data.
- **Independent:** Tests do not depend on external states or each other.
- **Self-validating:** Assertions confirm both success and failure conditions.

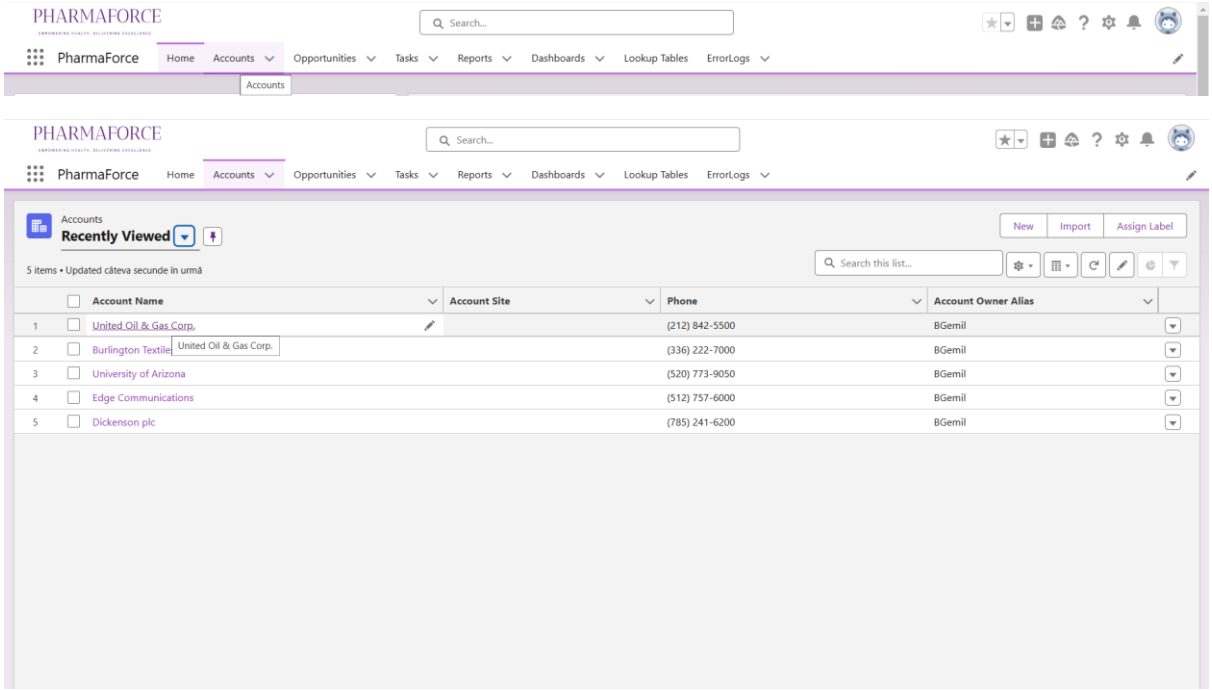
8. Creating Reports and Dashboards for Urgent Opportunities

- Created a new Home Page and set it as App Default.
- Created a new Report component: added a filter for the Urgent field (**Urgent__c** equals **True**), included the required fields as columns and grouped the data by **Stage**.
- Created a new Dashboard containing a Table component, a Bar chart and a Summary metric.
- Added the dashboard to the Home Page.
- Added the Home Page to the App.

User Guide: Steps for Salesforce User

1. View Urgent Opportunities

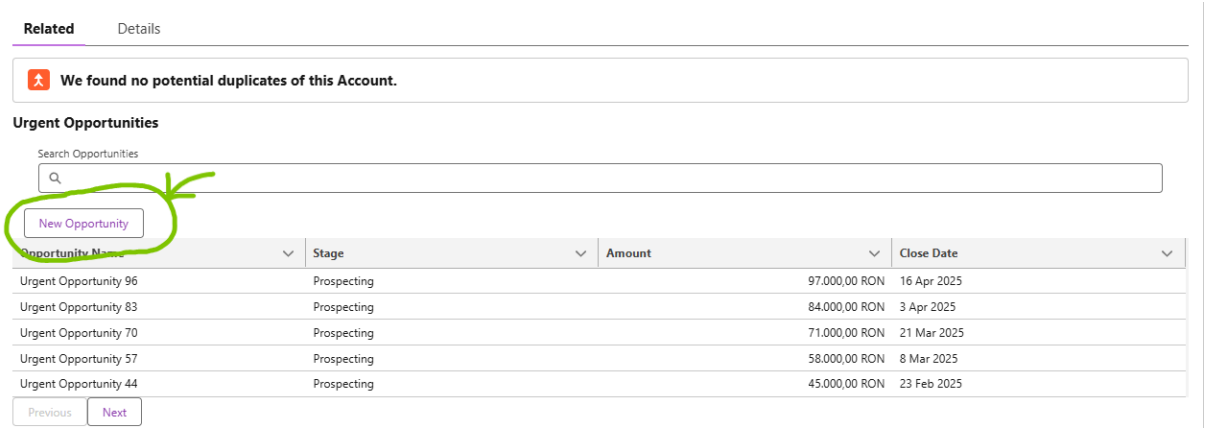
- Navigate to the **Account Record Page**:



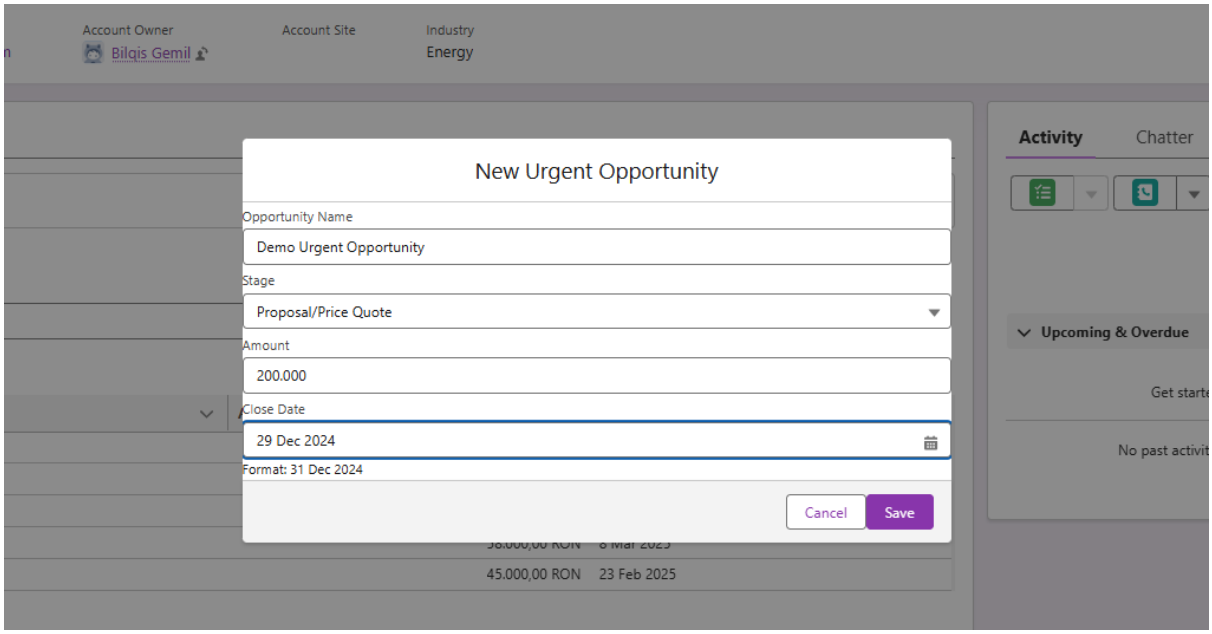
- Look for **the Related tab** to view the urgent opportunities table.

2. Create a New Urgent Opportunity

- Click the **"New Opportunity" button** in the urgent opportunities table:



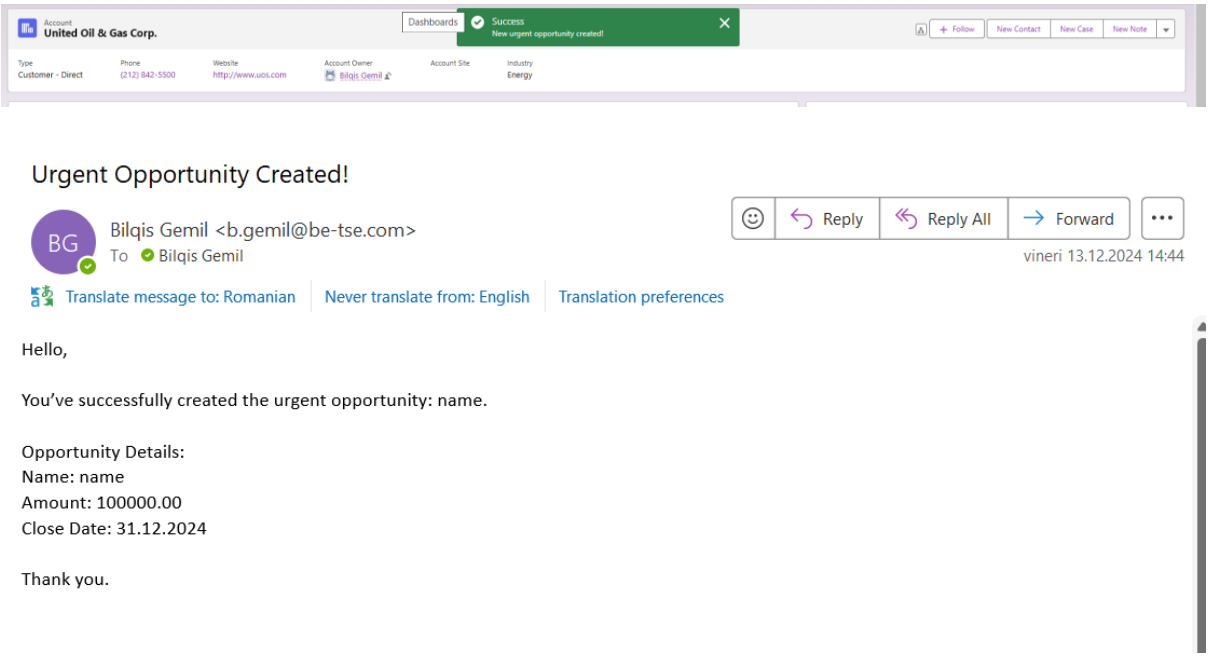
- Fill in the **Opportunity Name, Stage, Amount, and Close Date** field in the modal:



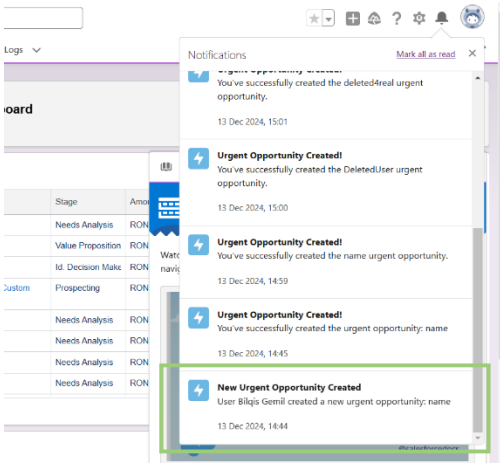
- Click **"Save"** to add the opportunity.

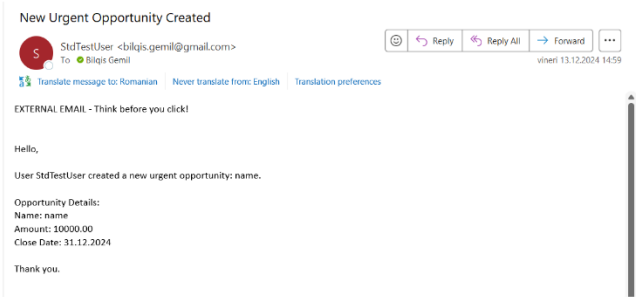
3. Notifications

- Opportunity creators will see **a success message** after saving and will receive **an e-mail message confirming the operation** if they are part of the account team:



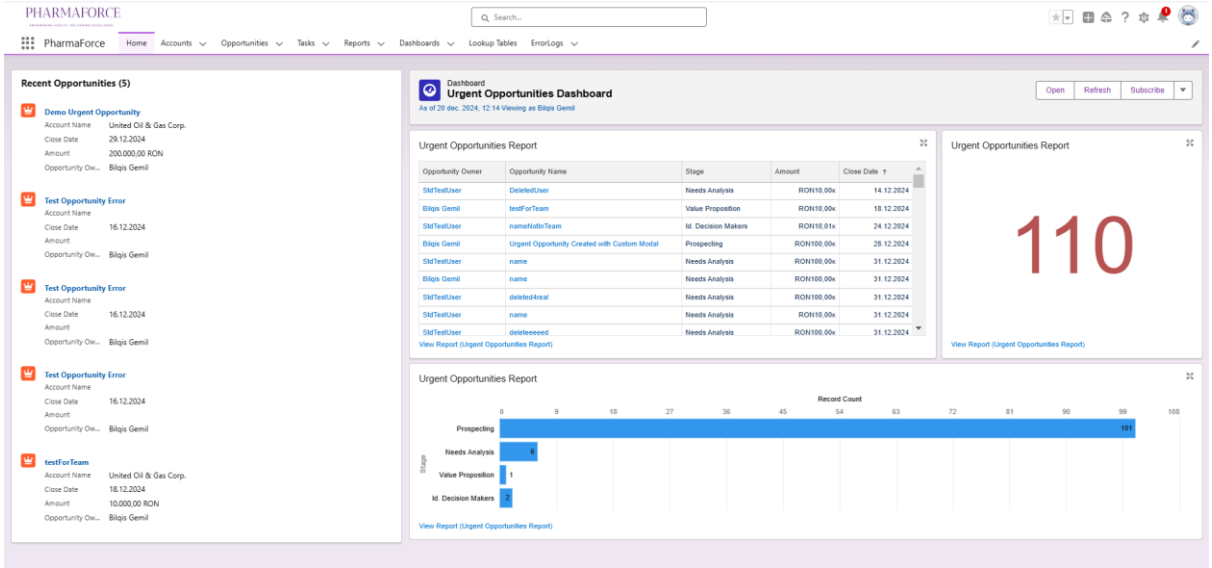
- Other account team members will receive **a notification** and **an e-mail message** about the new urgent opportunity being created:





4. Accessing the Dashboard

- From the navigation bar, click on **Home** to access the default homepage for the PharmaForce app. The dashboard is embedded directly on the Home Page.



- The dashboard contains the following sections:
  - **Recent Opportunities**: Displays a list of the most recent opportunities created or updated.
  - **Urgent Opportunities Report Table**: A detailed table listing all urgent opportunities.
  - **Summary Metric**: Shows the total number of urgent opportunities.
  - **Bar Chart**: Visualizes the distribution of urgent opportunities by stage.

Summary

The PharmaForce Urgent Opportunities Component Implementation fulfills the defined requirements by introducing a tailored solution for managing urgent opportunities in Salesforce. The project delivers:

- A **user-friendly interface** for creating and managing **urgent opportunities** directly on the Account Record Page.

- Enhanced team collaboration through **automated notifications** and streamlined data management.
- A **dashboard** providing a centralized view of urgent opportunities, enabling efficient monitoring and analysis.

The solution adheres to Salesforce best practices, ensuring scalability, maintainability, and alignment with PharmaForce's operational goals. It provides a foundation for improved opportunity tracking while supporting further enhancements based on evolving business needs.