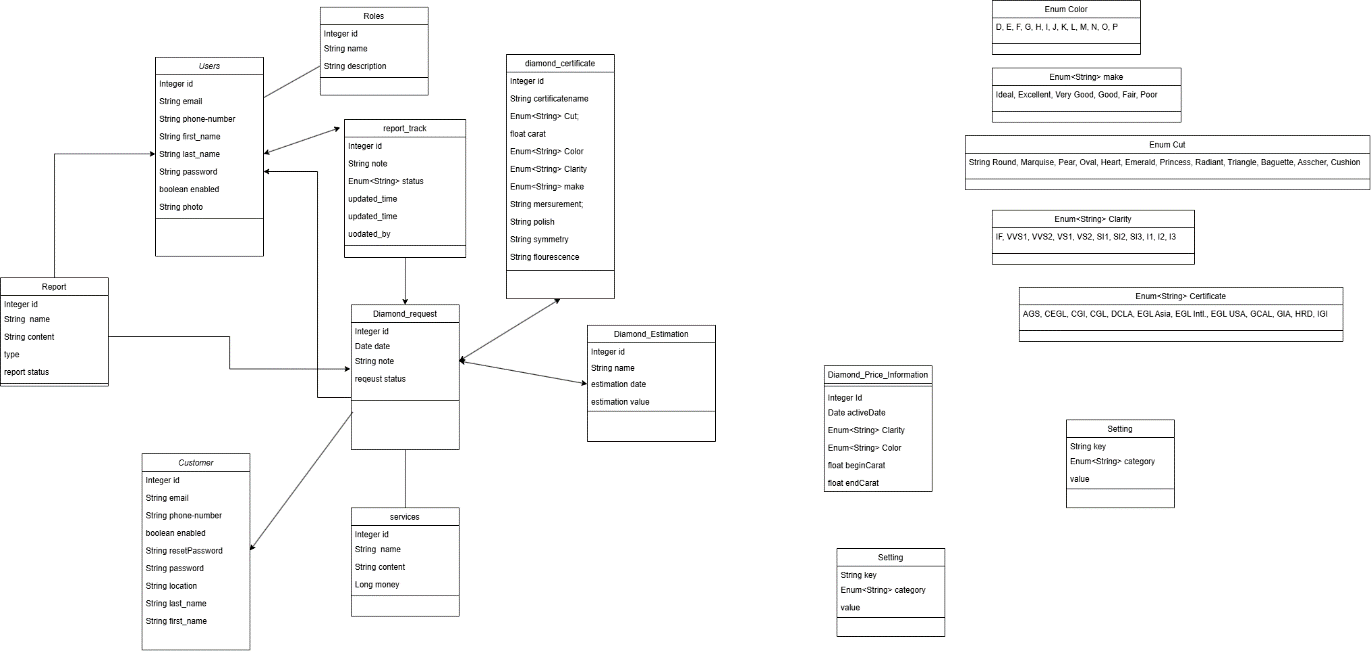
**ERD/DB Diagram:**



**Function requirements:**

1. User Login/Logout
2. Dashboard and Reporting
3. Manage Users
4. Manage Customers
5. Manage Services
6. Manage Request Services
7. Manage Certificates
8. View Home Page
9. Customer Login/Logout
10. Diamond Valuation Estimation
11. Valuation Request Management
12. Handling Unclaimed

**Usecase list:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | User Login/Logout | Login | Admin/Manager/Staff/Valuation Staff | User logs into the system using their credentials. |
| 2 | Logout | User logs out of the system. |
| 3 | Dashboard and Reporting | View Dashboard and Statistics | Admin/Manager | Views the dashboard for statistical analysis and reporting of valuation requests and other key metrics. |
| 4 | Manage User | View User | Admin/Manager/Staff/Valuation Staff | Views details of user accounts. |
| 5 | Create User | Admin | Admin creates a new user account. |
| 6 | Edit User | Admin edits details of an existing user account. |
| 7 | Delete User | Admin deletes an existing user account. |
| 8 | Manage Customer | View Customers | Admin/Manager/Staff/Valuation Staff | Views details of customer profiles. |
| 9 | Create Customer | Admin/Staff/Manager | Creates a new customer profile. |
| 10 | Edit Customer | Edits details of an existing customer profile. |
| 11 | Delete Customer | Deletes an existing customer profile |
| 12 | Manage Services | View Services | Admin/Manager/Staff/Valuation Staff | Views details of existing services. |
| 13 | Create Service | Admin | Admin creates a new valuation service. |
| 14 | Edit Service | Admin edits details of an existing service. |
| 15 | Delete Service | Admin deletes an existing service. |
| 16 | Manage Request Services | View Valuation Request | Admin/Manager/Staff/Valuation Staff | User views details of valuation requests. |
| 17 | Submit Valuation Request | Staff/Customer | User submits a valuation request for a diamond. |
| 18 | Process Valuation Request | Staff/Valuation Staff | User processes the valuation request and assigns it for valuation. |
| 19 | Update Valuation Request Status | User updates the status of the valuation request as it progresses through the valuation process. |
| 20 | Approve Sealing Report | Manager | Manager approves the sealing report for unclaimed valuation results. |
| 21 | Approve Commitment Form | Manager approves the commitment form for lost receipts. |
| 22 | Manage Diamond Certificate | View Diamond Certificate | Admin/Manager/Staff/Valuation Staff | User views details of diamond certificates. |
| 23 | Create Diamond Certificate | Valuation Staff | Valuator creates a diamond certificate based on the valuation results. |
| 24 | Edit Diamond Certificate | Valuator edits details of an existing diamond certificate. |
| 25 | Print Diamond Certificate | Prints the diamond certificate for the customer |
| 26 | Delete Diamond Certificate | Deletes an existing diamond certificate |
| 27 | View Home Page | View Home Page | Customer | Customer views the home page, which introduces the company, services offered, diamond knowledge resources, search and valuation functionalities, and blog. |
| 28 | Login/Logout | Login | Customer logs into the system using their credentials. |
| 29 | Logout | Customer logs out of the system. |
| 30 | Manage Account | Edit Profile | Edits details of account profile. |
| 31 | Diamond Valuation Estimation | Estimate Diamond Value by Criteria | Customer inputs criteria such as diamond origin, shape & cut, measurements, carat weight, color, clarity, cut, proportions, polish, symmetry, and fluorescence to estimate the diamond's value. |
| 32 | Estimate Diamond Value by Certification Number | Customer inputs a certification number issued by the company to estimate the diamond's value. |
| 33 | Valuation Request Management | Submit Valuation Request | Customer submits a valuation request form. |
| 34 | View Valuation Request Status | Customer views the status of their submitted valuation request. |
| 35 | Handling Unclaimed Valuation Results | Request Commitment Form for Lost Receipt | Customer requests a commitment form to receive their diamond sample if they have lost the valuation receipt. |

**UI(Fontend):**

**Technical Requirements:**

Login User

LU1 – Successful Login

LU2 – Fail Login

Create User

# CU3 - Input Validation for Email Field

# CU4 - Input Validation for Required Fields

# CU5 - Handle Image Upload and Preview

# CU6 - Submit Form with Valid Data

# CU7 - Submit Form with Invalid Email

# CU8 - Cancel Form Submission

# CU9 - Load EditUser Component

# Edit User

# EU10 - Fetch and Display User Data

# EU11 - Input Validation for Email Field

# EU12 - Input Validation for Required Fields

# EU13 - Handle Image Upload and Preview

# EU14 - Submit Form with Valid Data

# EU15 - Submit Form with Invalid Email

# EU16 - Submit Form with Duplicate Email

# EU17 - Cancel Form Submission

# Delete User

# DU18 - Open Delete Confirmation Dialog

# DU19 - Cancel Delete Action

# DU20 - Confirm Delete Action

Create Customer

CC22 - Successful Form Submission

CC23 - Form Submission with Invalid Email

CC24 - Form Submission with Missing Required Fields

CC25 - Form Submission with Invalid Phone Number

CC26 - Cancel Form Submission

Edit Customer

EC27 - Render Form with Customer Data

EC28 - Successful Form Submission

EC29 - Cancel Form Submissions

EC30 - Password Field Behavior

Delete Customer

DC31 - Open Delete Confirmation Dialog

DC32 - Cancel Deletion

DC33 - Confirm Deletion

Add Service

AS34 – Validate empty form submission

AS35 – Validate form submission with invalid image file

AS36 – Validate cancel button functionality

Edit Service

ES37 – Successful service edit

ES38 – Missing required fields

ES39 – Cancel button functionality

ES40 – Form Submission with no changes

ES41 – Editing only non-required fields

ES42 – Large file upload for photo

ES43 – Invalid file type for photo

ES44 – Error message display timeout

Delete Service

DS45 – Open and close delete confirmation dialog

DS46 – Delete a service

DS47 – Display success message on successful action

TEST CASE ADD DIAMOND REQUEST

**1. Form Validation**

**Description:** Validate form behavior when submitting with invalid or missing data.

**Test Steps:**

* Attempt to submit the form with invalid or empty fields:
  + Customer not selected.
  + Status not selected.
  + No service selected.

**Expected Result:**

* Validation errors should appear next to respective fields:
  + Error message "Customer is required" should appear if customer is not selected.
  + Error message "Status is required" should appear if status is not selected.
  + Error message "At least one service must be selected" should appear if no service is selected.
* Form should not submit until all required fields are filled correctly.

**2. Form Submission**

**Description:** Test the form submission process.

**Test Steps:**

* Fill all required fields correctly (customer, note, status, and select at least one service).
* Click on the "Save" button.

**Expected Result:**

* Form should submit without validation errors.
* The saveRequest function should be called with correct form values.
* Upon successful submission:
  + A success message should briefly appear ("Add new Request successfully").
  + The user should be navigated to the "/requests" page.

**3. User Interaction**

**Description:** Test user interactions such as selecting a customer, selecting services, and changing status.

**Test Steps:**

* Select a customer from the autocomplete dropdown.
* Select one or more services using checkboxes.
* Change the status using the dropdown.

**Expected Result:**

* Customer ID should be correctly set when a customer is selected.
* Services should be correctly added/removed from service\_ids array based on checkbox selection.
* Status should update as per user selection in the dropdown.

**5. Cancel Button Functionality**

**Description:** Test the functionality of the "Cancel" button.

**Test Steps:**

* Click on the "Cancel" button.

**Expected Result:**

* The user should be navigated to the "/requests" page.

EDIT DIAMOND REQUETS

### **Test Case 1: Form Submission with Valid Data**

Description: Test the submission of the form with valid input data.

Test Steps:

1. Populate all required fields (customer\_id, note, status, and select at least one service).
2. Submit the form by clicking on the "Save Changes" button.

Expected Result:

* Form submission succeeds.
* saveRequest function is called with the correct values.
* Successful navigation to "/requests".
* Success message (Alert severity="success") is displayed.

### **Test Case 2: Form Submission with Invalid Data**

Description: Test the form submission with missing required fields.

Test Steps:

1. Leave one or more required fields (customer\_id, status, or services) empty.
2. Submit the form by clicking on the "Save Changes" button.

Expected Result:

* Form submission fails.
* Validation errors are displayed for each missing required field.
* Error message (Alert severity="error") is displayed indicating the issue.

### **Test Case 3: Editing an Existing Request**

Description: Test the pre-population of the form when editing an existing request.

Test Steps:

1. Mock an existing requestid to simulate editing.
2. Ensure that the form fields (customer\_id, note, status, service\_ids) are correctly pre-populated with the existing request data.

Expected Result:

* Form fields are populated with data fetched from getRequestById.
* The correct customer is selected in the autocomplete based on requestEdit.customer\_phone.
* Services checkboxes reflect the existing service\_ids selected.

### **Test Case 4: Cancel Button Functionality**

Description: Test the "Cancel" button functionality.

Test Steps:

1. Click on the "Cancel" button.
2. Verify that navigation occurs to "/requests" route.

Expected Result:

* Clicking the "Cancel" button navigates the user back to the "/requests" route without making any changes.

DELETE DIAMOND REQUEST

### **Test Case 1: Open Delete Confirmation Dialog**

Description: Test that verifies the delete confirmation dialog opens correctly when clicking the delete button.

Test Steps:

1. Navigate to the Requests component.
2. Locate a request entry in the table.
3. Click on the delete icon/button corresponding to the request.

Expected Result: The delete confirmation dialog should open with the message "Confirm Delete". The request details (such as customer name or ID) should be displayed in the dialog content.

### **Test Case 2: Cancel Deletion**

Description: Test that ensures canceling the deletion does not remove the request.

Test Steps:

1. Navigate to the Requests component.
2. Open the delete confirmation dialog for a request.
3. Click on the "Cancel" button in the confirmation dialog box.

Expected Result: The delete confirmation dialog should close, and the request should remain visible in the table without any changes. No success or error message should be displayed.

### **Test Case 3: Confirm Deletion**

Description: Test that verifies successful deletion of a request when confirming deletion in the dialog.

Test Steps:

1. Navigate to the Requests component.
2. Open the delete confirmation dialog for a request.
3. Click on the "Delete" button in the confirmation dialog box.

Expected Result: The request should be deleted from the UI. A success message should appear briefly, indicating "Delete request with id {request\_id} successfully!". The deleted request should no longer be visible in the table.

### **Additional Considerations:**

* UI Validation: Verify that the delete confirmation dialog displays the correct request details (such as ID, customer name) before confirming deletion.
* Edge Cases: Test with various scenarios, such as deleting the first or last request, deleting multiple requests consecutively, etc.
* Error Handling: Ensure the UI handles errors gracefully, such as network issues or server errors during deletion

ADD SERVICE

**Test Case 1: Validate Empty Form Submission**

**Description:** Test the behavior when the form is submitted without any input.

**Test Steps:**

1. Render the AddService component.
2. Click the "Save" button without filling in any fields.

**Expected Result:**

* Validation errors should be displayed for all required fields (name, money, photo).

**Test Case 5: Validate Form Submission with Invalid Image File**

**Description:** Test the behavior when the form is submitted with an invalid image file format (e.g., a text file).

**Test Steps:**

1. Render the AddService component.
2. Fill in the "Name" field with Test Service.
3. Fill in the "Money" field with 100.
4. Upload an invalid file format (e.g., a .txt file) in the "Photo" field.
5. Click the "Save" button.

**Expected Result:**

* A validation error message "Image is required" should be displayed for the "Photo" field.

**Test Case 6: Validate Cancel Button Functionality**

**Description:** Test the behavior when the "Cancel" button is clicked.

**Test Steps:**

1. Render the AddService component.
2. Click the "Cancel" button.

**Expected Result:**

* The user should be navigated to the /services page.
* No data should be submitted or saved.

EDIT SERVICE

**Test Case 1: Successful Service Edit**

**Description**: Ensure that the user can successfully edit an existing service with valid data.

**Test Steps**:

1. Navigate to the "Edit Service" page for a specific service (e.g., /edit-service/1).
2. Verify that the existing service details are pre-filled in the form fields.
3. Modify the name in the "Name" field (e.g., "Updated Service Name").
4. Modify the amount in the "Money" field (e.g., "200").
5. Modify the content description in the "Content" field (e.g., "Updated Service Description").
6. Upload a new valid image file in the "Photo" field (e.g., a new JPEG or PNG file).
7. Click the "Save" button.

**Expected Result**:

* The form submits without errors.
* The user is redirected to the "/services" page.
* A success message "Edit Service {id} successfully" is stored in the local storage.

**Test Case 2: Missing Required Fields**

**Description**: Ensure that the form validation works and displays errors when required fields are missing.

**Test Steps**:

1. Navigate to the "Edit Service" page for a specific service.
2. Clear all data from the form fields.
3. Click the "Save" button.

**Expected Result**:

* Validation errors are displayed for the "Name", "Money", and "Photo" fields indicating that these fields are required.

**Test Case 6: Cancel Button Functionality**

**Description**: Ensure that the cancel button redirects the user to the services list without saving the data.

**Test Steps**:

1. Navigate to the "Edit Service" page for a specific service.
2. Modify data in all the fields.
3. Click the "Cancel" button.

**Expected Result**:

* The user is redirected to the "/services" page.
* No changes are saved.

**Test Case 9: Form Submission with No Changes**

**Description**: Ensure that the form can be submitted without any modifications to the initial values.

**Test Steps**:

1. Navigate to the "Edit Service" page for a specific service.
2. Do not modify any of the pre-filled data in the form fields.
3. Click the "Save" button.

**Expected Result**:

* The form submits without errors.
* The user is redirected to the "/services" page.
* A success message "Edit Service {id} successfully" is stored in the local storage.

**Test Case 10: Editing Only Non-required Fields**

**Description**: Ensure that the form can be submitted by modifying only the non-required "Content" field.

**Test Steps**:

1. Navigate to the "Edit Service" page for a specific service.
2. Modify the "Content" field (e.g., "Updated Content Description").
3. Ensure the "Name", "Money", and "Photo" fields retain their original values.
4. Click the "Save" button.

**Expected Result**:

* The form submits without errors.
* The user is redirected to the "/services" page.
* A success message "Edit Service {id} successfully" is stored in the local storage.

**Test Case 11: Large File Upload for Photo**

**Description**: Ensure that the form handles large file uploads gracefully and validates the file size if applicable.

**Test Steps**:

1. Navigate to the "Edit Service" page for a specific service.
2. Enter valid data in the "Name", "Money", and "Content" fields.
3. Upload a large image file (e.g., more than 5MB) in the "Photo" field.
4. Click the "Save" button.

**Expected Result**:

* If there is a file size validation, a validation error is displayed for the "Photo" field.
* If there is no size validation, the form submits successfully, and the large image is processed accordingly.

**Test Case 12: Invalid File Type for Photo**

**Description**: Ensure that the form validation catches invalid file types for the "Photo" field.

**Test Steps**:

1. Navigate to the "Edit Service" page for a specific service.
2. Enter valid data in the "Name", "Money", and "Content" fields.
3. Upload an invalid file type (e.g., a PDF) in the "Photo" field.
4. Click the "Save" button.

**Expected Result**:

* A validation error is displayed for the "Photo" field indicating that only image files (JPEG, PNG) are allowed.

**Test Case 14: Error Message Display Timeout**

**Description**: Ensure that error messages disappear after the specified timeout period.

**Test Steps**:

1. Navigate to the "Edit Service" page for a specific service.
2. Enter valid data in all fields.
3. Simulate an API failure (e.g., by temporarily disabling the API endpoint).
4. Click the "Save" button.
5. Observe the error message.

**Expected Result**:

* An error message is displayed indicating the failure.
* The error message disappears after 3 seconds.

DELETE SERVICE

**Test Case 5: Open and Close Delete Confirmation Dialog**

**Description:** Test if the delete confirmation dialog opens and closes correctly.

**Test Steps:**

1. Mock the getAllServices function to return a list of services.
2. Render the Services component.
3. Click the delete (trash) icon on a service.
4. Check if the delete confirmation dialog opens.
5. Close the dialog and check if it closes properly.

**Expected Result:**

1. The delete confirmation dialog should open.
2. The delete confirmation dialog should close properly when the cancel button is clicked.

**Test Case 6: Delete a Service**

**Description:** Test if a service is deleted correctly.

**Test Steps:**

1. Mock the getAllServices function to return a list of services.
2. Mock the deleteServiceById function to return a success response.
3. Render the Services component.
4. Click the delete (trash) icon on a service.
5. Confirm the deletion in the delete confirmation dialog.
6. Check if the service is deleted and the message is displayed.

**Expected Result:**

1. The service should be deleted and no longer displayed in the table.
2. A success message should be displayed.

**Test Case 12: Display Success Message on Successful Action**

**Description:** Test if a success message is displayed when an action is successfully completed.

**Test Steps:**

1. Mock the getAllServices function to return a list of services.
2. Mock the deleteServiceById function to return a success response.
3. Render the Services component.
4. Perform an action that results in a success (e.g., deleting a service).
5. Check if a success message is displayed.

**Expected Result:**

1. A success message should be displayed after the action is successfully completed.