



**Faculty of Engineering
and Applied Science**

Fall 2024
ENGI 9837 - Software Engineering Capstone

Project Design Document

Radar Data Viewer

Project Supervisor

Prof. Reza Shahidi

Group Members

1. Eialid Ahmed Joy - 202384815
2. Monirul Islam - 202381823
3. Jakaria Hossain - 202293102
4. B M Al-amin Badhon - 202384559
5. Sadia Noshin Shuchi - 202380413

Submission Date

October 15, 2024

Introduction

The Radar Data Viewer Project will be a web-based software designed to visualize high-frequency (HF) and X-Band radar data. Initially, it will focus on displaying radar data files in a B-Scan format, with plans to expand to multiple files and animations in the future. The web application will also show key details such as when and where the radar data was collected, the transmission frequency, bandwidth, and the maximum range. Although the first version of the software will have limited functionality, it will be accessible from different operating systems like Windows, Ubuntu, etc. In addition to that, we plan to integrate advanced data analysis features, including spectrum and parameter extraction algorithms, in future versions of the software.

High-Level Description

The Radar Data Viewer software is structured into several key modules, each responsible for a specific functionality within the system. These modules work together to provide a seamless experience for loading, visualizing, and analyzing radar data. Below is a high-level description of each module, its behavior, and how it interacts with other modules.

1. Data Loading Module

a. Functionality:

- i. Responsible for importing radar data files (both HF radar and X-Band radar).
- ii. Supports reading from SORT files for HF radar data and other formats for X-Band radar data.

b. Behavior:

- i. Parses radar data files and converts the raw data into a format that can be visualized by the system.
- ii. Extracts key metadata (e.g., acquisition time, frequency, location) and stores it for display.

c. Interaction:

- i. Interacts with the Data Processing Module to send raw radar data for further processing and transformation.
- ii. Passes metadata to the User Interface (UI) Module for display alongside the visualization.

2. Data Processing Module

a. Functionality:

- i. Handles the processing and transformation of radar data for visualization.
- ii. Supports rendering radar data in the B-Scan format and, in future iterations, scan-converted format.

- b. Behavior:
 - i. Converts radar data into a visual format by applying necessary transformations.
 - ii. It may include basic data filtering or noise reduction steps before rendering.
- c. Interaction:
 - i. Receives raw radar data from the Data Loading Module.
 - ii. Sends processed data to the Visualization Module for rendering.
 - iii. Passes wave spectrum data or other extracted parameters (in future iterations) to the Analysis Module.

3. Visualization Module

- a. Functionality:
 - i. Responsible for rendering radar data in graphical form for users.
 - ii. Initially, it supports B-Scan visualization, with future capabilities for scan-converted visualizations and animations.
- b. Behavior:
 - i. Displays radar data in an intuitive and user-friendly graphical format.
 - ii. It supports zooming, panning, and toggling between different visualization modes.
- c. Interaction:
 - i. Receives processed data from the Data Processing Module and renders it for display in the User Interface Module.
 - ii. Interacts with the User Interface Module to reflect changes in visualization based on user inputs (e.g., switching between B-Scan and scan-converted formats).

4. User Interface (UI) Module

- a. Functionality:
 - i. It provides a graphical user interface that users interact with to load data, view visualizations, and access metadata.
 - ii. Displays key radar data parameters (e.g., acquisition date, frequency, location).
- b. Behavior:
 - i. Offers a simple and intuitive interface for selecting radar files and viewing the resulting data.
 - ii. Ensures users can easily toggle between different visualizations and access important metadata.
- c. Interaction:
 - i. Receives radar metadata from the Data Loading Module for display.
 - ii. Sends user input (e.g., file selection, visualization options) to the Data Loading Module and Visualization Module.

- iii. Interacts with the Visualization Module to adjust display settings based on user actions.

5. Analysis Module (Future Scope)

- a. Functionality:
 - i. augments the system with advanced analysis features, including wave spectrum analysis and parameter extraction.
- b. Behavior:
 - i. Processes radar data to extract meaningful parameters like wave height, direction, and ocean current patterns.
 - ii. Provides analytical insights that are displayed alongside the visualizations.
- c. Interaction:
 - i. Receives processed radar data from the Data Processing Module.
 - ii. Sends analytical results to the User Interface Module for display alongside the radar visualizations.

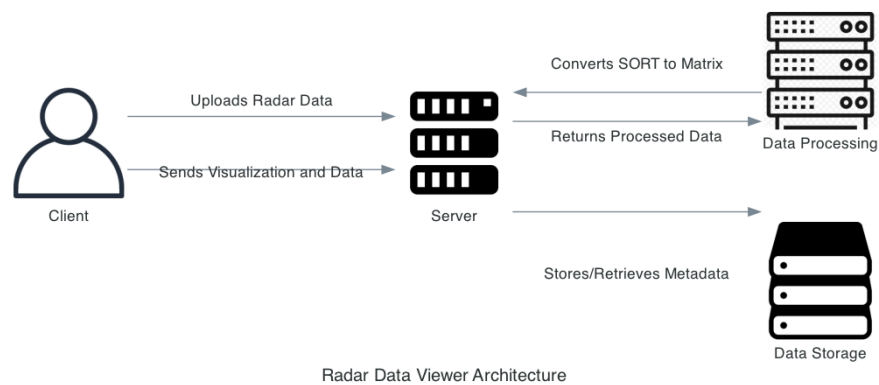
6. Cross-Platform Compatibility Module (Future Scope)

- a. Functionality:
 - i. Ensures that the software runs smoothly on multiple platforms, including Windows, Ubuntu, and potentially other operating systems.
- b. Behavior:
 - i. Abstracts platform-specific functionality, allowing the software to run consistently across different operating systems.
 - ii. It may include a web-based interface for platform-independent use.
- c. Interaction:
 - i. Interacts with all modules, ensuring that the entire system remains compatible and performs well on different platforms.

High-Level Architecture

This high-level architecture describes a web-based client-server model for the Radar Data Viewer. The ReactJS front-end enables users to upload radar files, view visualizations, and access metadata. The Django back-end processes radar data, converts SORT files into matrix format and interacts with PostgreSQL to store metadata and processed results. NumPy and SciPy handle matrix conversion and data processing. The modular system allows for future enhancements like advanced visualizations and wave spectrum analysis.

Fig.: High-level architecture



Use Case Diagram

Participating Actors: The user, The system

Preconditions:

1. The SORT file format is required to upload.
2. The user should be either a new user/registered user
3. To watch the previously loaded files, the user must be logged in.

Flow of Events:

1. The user should be able to register in the system as the new user
 - a. Basic validations such as email validation and password length should be in the system so that the user can provide a valid username and password.
2. The user should be able to log in whenever they try to visit the system
 - a. The system will check the credentials and validate the data. if the credentials are correct, the user will log in else the system will notify the user that the credentials are not correct
3. The user must be able to upload files like SORT files, one file at a time.
 - a. If the uploaded file is not SORT or in any predefined format, the system will notify you that it's an invalid format.
4. A SORT file can have one or multiple images, and the system needs to handle both of these criteria
 - a. If the file is empty, the system needs to handle it and let the user know that the file is empty
5. After uploading, the SORT file must be converted into browser-formatted image files such as PNG, JPG, JPEG, etc.
 - a. If the system cannot process any images from the SORT file in particular, the user needs to be notified.
6. All those images need to be bundled together, and the user should be able to see them one after another. (Like a preview option where the user will click Next to see the next image and the Back option to watch the previous picture.)
7. The system should be able to load some previously uploaded files, if possible, for each particular user. Hence, the user can see their latest uploaded files and see those images.
8. The user needs to be able to log out from the system whenever they prefer to sign out from the system.

Special Requirements:

1. The uploaded SORT file's image must be converted into browser-supported format such as PNG, JPG, or JPEG

2. File size limits for Uploads.

Postconditions:

1. Images are shown to the user for preview
2. The session ends when a user logs out

Following is the use case diagram for Radar Data Viewer:



Fig.: Use Case Diagram

Class Diagram

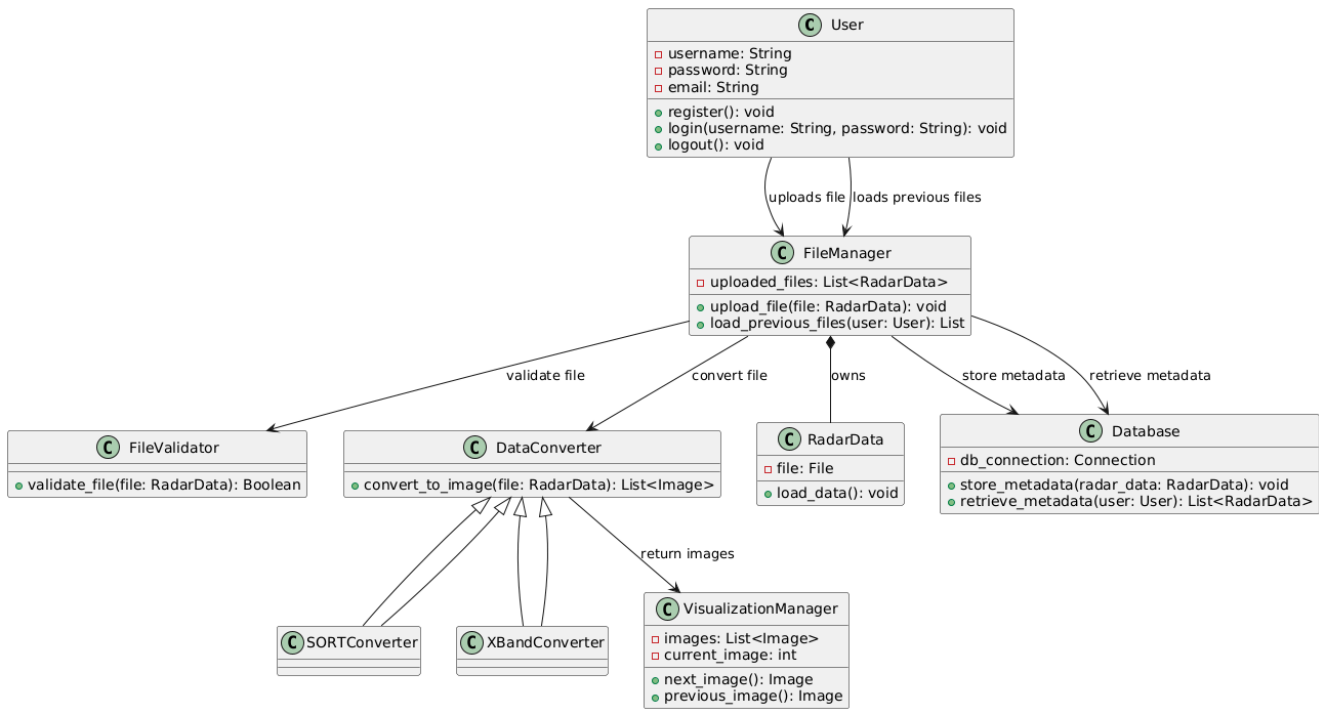


Fig.: Class Diagram

Sequence Diagram

The diagrams cover the core functionalities of the system, from user login validation to handling radar data uploads, image generation, and pre-uploaded data. Each diagram plays a crucial role in outlining the flow and interaction between system components, ensuring clarity in how the system processes user actions and data.

1. Full Process Sequence Diagram

The Full Process diagram provides a comprehensive overview of the entire radar data visualization system, from login to data processing and image generation.

Description:

- The user logs in to the system using the Login Service.
- Once logged in, the user accesses the dashboard, where the system checks for pre-uploaded data from the Radar Data Table.

- c. If pre-uploaded data exists, a default image is generated from this data using the Algorithm/Script service.
- d. If no pre-uploaded data exists, the user is presented with an option to upload new radar data.
- e. After uploading data, the system stores the data and processes it to generate PNG images.
- f. These images are displayed to the user on the dashboard.

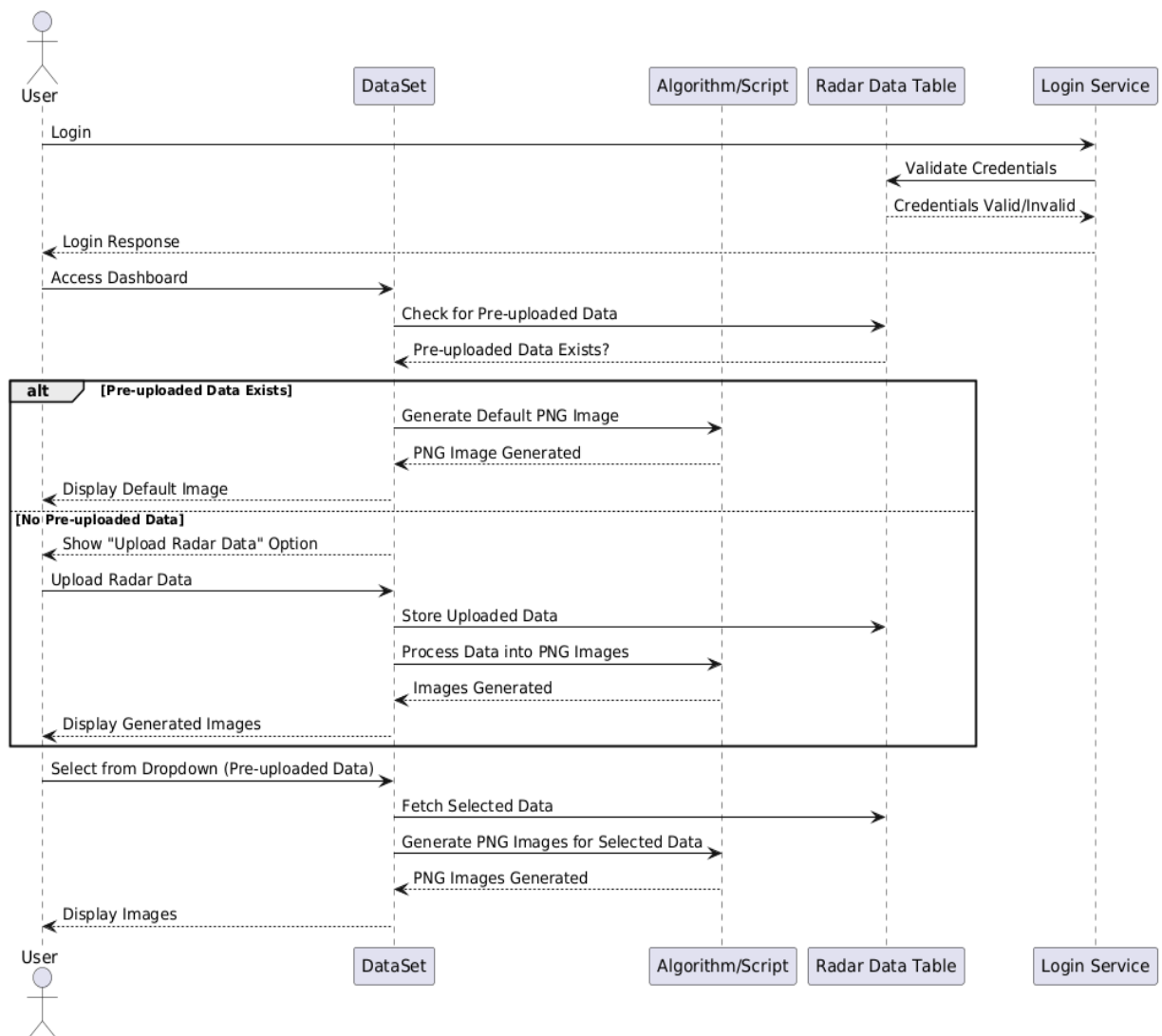


Fig.: Full Process Sequence Diagram

2. Login Service Sequence Diagram

The Login Service diagram outlines the process of user authentication. It describes how the system validates user credentials, whether the login is successful, and the corresponding feedback provided to the user.

Description:

- The user enters their login credentials.
- The system sends these credentials to the Login Service.
- The Login Service communicates with the User Data Table to validate the credentials.
- If the credentials are valid, a success notification is sent to the user via the Notification Service.
- If the credentials are invalid, the user is prompted to re-enter their credentials, and the system re-checks until the login is successful or canceled.

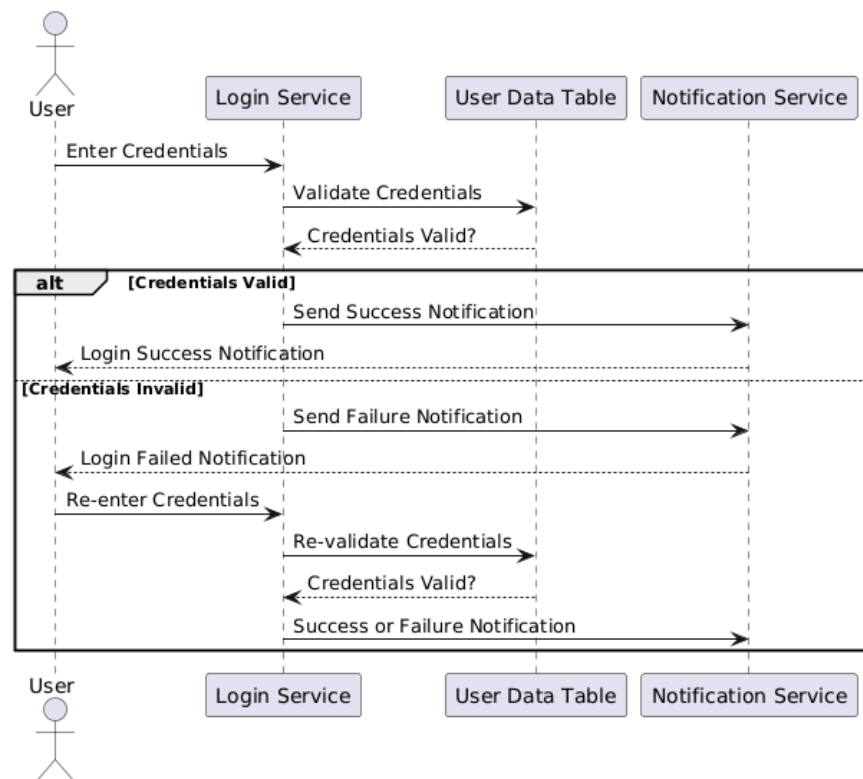


Fig.: Login Service Sequence Diagram

3. Data Upload Validation Sequence Diagram

The Data Upload Validation diagram focuses on validating the radar data during the upload process. It checks if the uploaded data is in the correct format or if it is empty.

Description:

- The user uploads radar data to the system.
- The Dataset service forwards the uploaded data to the Validation Service.
- The Validation Service checks whether the data is valid (i.e., it is not empty and follows the required format).
- If the data is valid, it is sent to the Radar Data Table for storage.
- If the data is invalid, a failure notification is sent to the user, prompting them to upload the correct data format.
- The system provides feedback on whether the upload was successful or failed due to an invalid dataset.

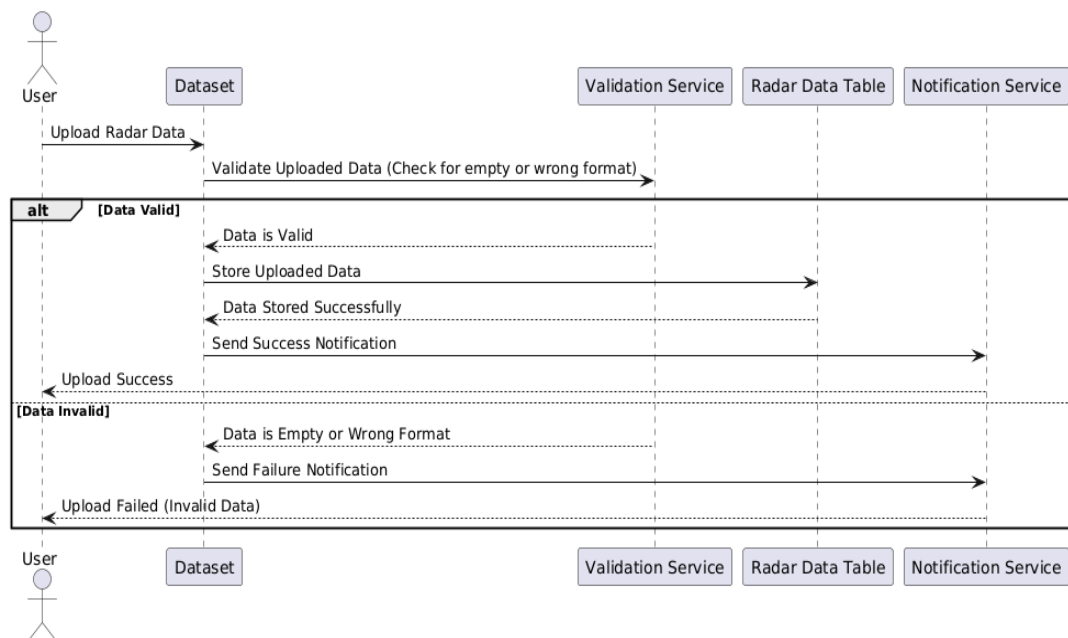


Fig.: Data Upload Validation Sequence Diagram

4. Upload and Generate Image Sequence Diagram

The Upload and Generate Image diagram illustrates how the system handles new radar data uploads and generates PNG images based on the uploaded data.

Description:

- The user uploads radar data through the dashboard.
- The Dataset service stores the uploaded data in the Radar Data Table.
- The user then presses the "Generate" button to process the uploaded data.

- d. The system retrieves the data from the Radar Data Table and sends it to the Algorithm/Script for processing.
- e. The Algorithm/Script processes the data and generates PNG images.
- f. These images are then displayed on the user's dashboard.

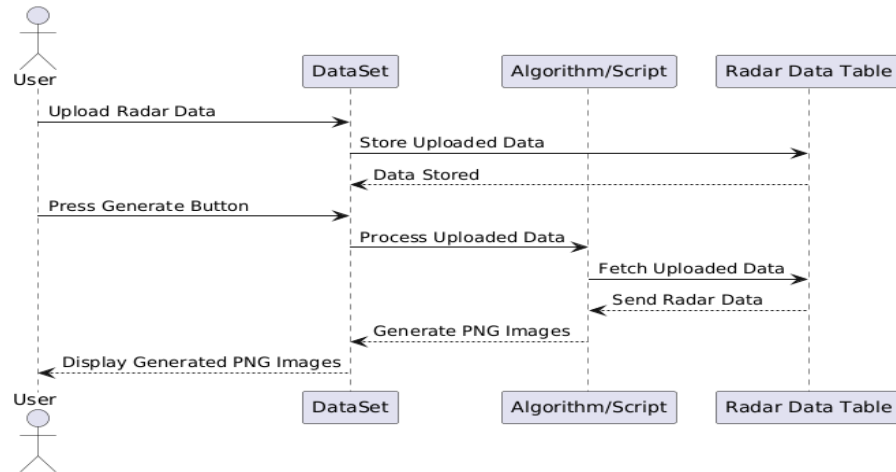


Fig.: Upload and Generate Image Sequence Diagram

5. Pre-uploaded Data Sequence Diagram

The Pre-uploaded Data diagram explains the process of generating images using pre-uploaded radar data. The system retrieves data from a list of all previously uploaded datasets and generates images based on the first uploaded dataset.

Description:

- a. The user logs in and accesses the dashboard.
- b. The system retrieves a list of all pre-uploaded radar datasets from the Radar Data Table.
- c. The user selects a dataset from the dropdown list.
- d. The system sends the selected dataset to the Algorithm/Script for processing.
- e. The Algorithm/Script processes the data and generates PNG images.
- f. These images are displayed to the user on the dashboard.

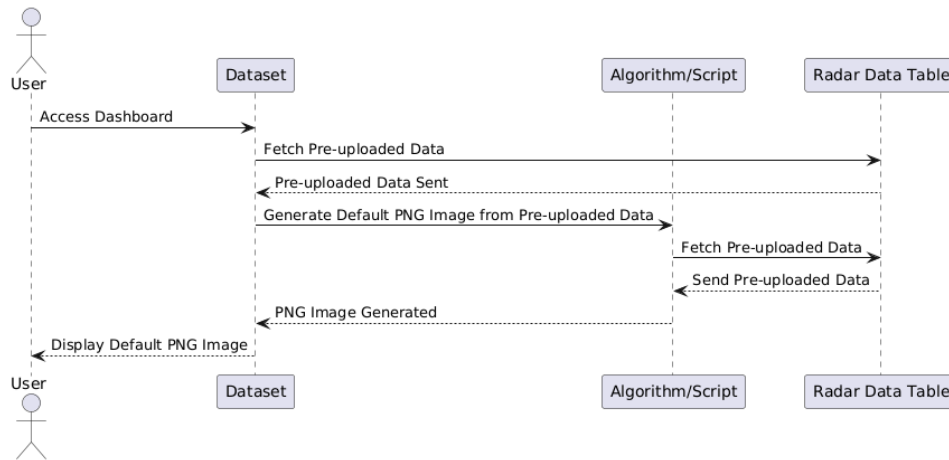


Fig: Pre-uploaded Data Sequence Diagram

Technical Approach

1. Server-side Objectives:
 - a. Handle data loading and processing of radar data (SORT files and X-Band radar files).
 - b. Provide a structured API for client-side communication.
 - c. Prepare the system to accommodate future expansion, such as parameter extraction and data storage for wave spectrum analysis.
2. Server-side Technologies:
 - a. Python: Core language for processing and back-end development.
 - b. NumPy: To convert radar data into matrices and perform matrix operations.
 - c. Pandas: For organizing and handling data pre-conversion. SciPy: For any advanced signal processing.
 - d. Django: To expose APIs so the ReactJS client can communicate with the server.
 - e. PostgreSQL: To store metadata and processed results.
 - f. SQLAlchemy: ORM for PostgreSQL to interact with the database easily.
 - g. Matplotlib: Visualization generation is required on the server for server-side data rendering.
3. Client-side Objectives:
 - a. Allow users to upload and view radar data. Render visualizations of radar data in B-Scan format.
 - b. Display metadata (acquisition time, location, frequency) alongside the visualization.
 - c. Design an intuitive user interface for ease of use.

4. Client-side Technologies:

- a. ReactJS is used to build a responsive and interactive front-end.
- b. Axios will handle API requests to the back end for radar data and metadata.
- c. Chart.js renders radar data visualizations within the React app.

Test Cases

1. File upload with valid SORT file

Test Steps

- a. Navigate to the file upload section.
- b. Select a valid SORT file.
- c. Submit the file for upload.

Expected Result

- a. The system accepts the SORT file, processes it, and converts it to browser-supported image format.
- b. The user is shown a preview of the images.

2. File upload with invalid file format

Test Steps

- a. Navigate to the file upload section.
- b. Select an invalid file format.
- c. Submit the file for upload.

Expected Result

- a. The system rejects the file and displays an error message indicating that the file format is not supported.

3. View previously uploaded files

Test Steps

- a. Navigate to the file load section.
- b. Check if the previously uploaded file is there and select the right one
- c. Hit the load button.

Expected Result

- a. The system loads and displays the images to the user.

4. Image preview navigation

Test Steps

- a. After uploading the SORT file, the user should see the image preview option.
- b. Click the “Next” button to navigate through the images.
- c. Click the “Prev” button to navigate to the previous image.

Expected Result

- a. The system should correctly load the next and previous images without any errors.

5. User Registration

Test Steps

- a. Provide name, email, and password for user registration
- b. Click the “Sign up” button

Expected Result

- a. User registration should be successful and show a proper message.

6. User Login with valid credentials

Test Steps

- a. Provide a valid email address
- b. Provide valid password
- c. Click the “Login” button to login

Expected Result

- a. The system should be logged in properly

7. User Login with invalid credentials

Test Steps

- a. Provide an invalid email/password
- b. Click the “Login” button to log in using each combination stated above

Expected Result

- a. The system should show messages for invalid user or password

8. Reset Password

Test Steps

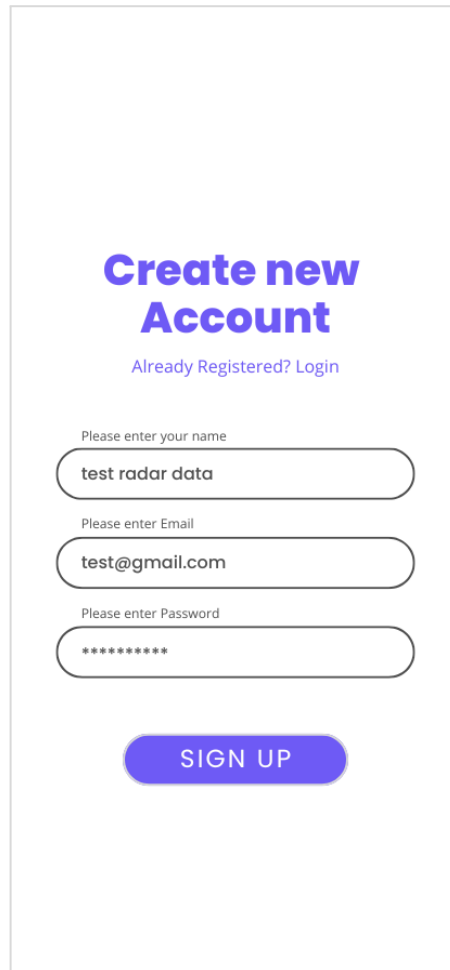
- a. Provide an email to send a new password
- b. Click the “Send” button to reset

Expected Result

- a. The system should send a new password to the user via email

Primary UI design

Registration page:



The registration page features a clean, minimalist design with a white background. At the top, the text "Create new Account" is displayed in a bold, purple font. Below this, a link "Already Registered? Login" is shown in a smaller, purple font. The form consists of three input fields, each with a placeholder text above it: "Please enter your name" (containing "test radar data"), "Please enter Email" (containing "test@gmail.com"), and "Please enter Password" (containing "*****"). All input fields have a rounded rectangular border. At the bottom of the form, there is a prominent purple button with the text "SIGN UP" in white, uppercase letters.

Create new Account

[Already Registered? Login](#)

Please enter your name

test radar data

Please enter Email

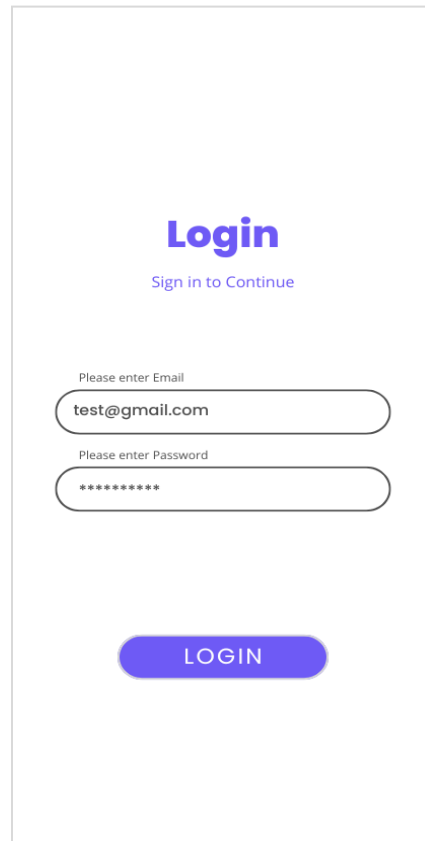
test@gmail.com

Please enter Password

SIGN UP

Fig.: Registration page

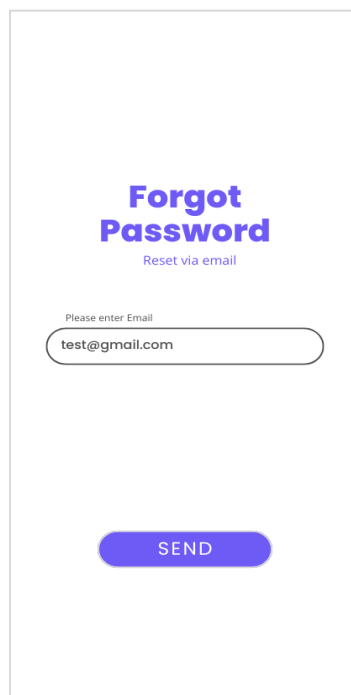
Login page:



The login page features a white background with a light gray border. At the top center, the word "Login" is displayed in a bold, dark blue font. Below it, the text "Sign in to Continue" appears in a smaller, lighter blue font. The form consists of two rounded rectangular input fields. The first field is labeled "Please enter Email" and contains the text "test@gmail.com". The second field is labeled "Please enter Password" and contains a series of asterisks "*****". Below the input fields is a blue rounded rectangular button with the word "LOGIN" in white capital letters.

Fig.: Login page

Reset Password page:



The reset password page has a white background with a light gray border. At the top center, the words "Forgot Password" are written in a bold, dark blue font. Below this, the text "Reset via email" is shown in a smaller, lighter blue font. There is a single rounded rectangular input field labeled "Please enter Email" containing the text "test@gmail.com". At the bottom of the form is a blue rounded rectangular button with the word "SEND" in white capital letters.

Fig: Reset Password page

Dashboard:

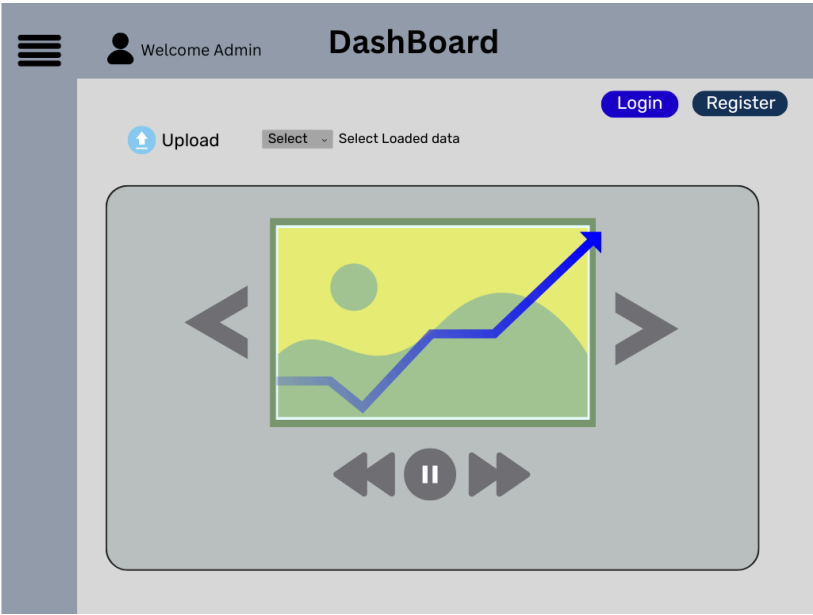


Fig.: Dashboard

View Uploaded file:



Fig.: View Uploaded file