MEMORIAL UNIVERSITY

Faculty of Engineering and Applied Science

Fall 2024
ENGI 9837 - Software Engineering Capstone

Final Project Report

Radar Data Viewer

Project Supervisor

Prof. Reza Shahidi

Group Members

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

Submission Date

December 13, 2024

Introduction

Overview: High-frequency (HF) radar and X-Band radar data will be visualized using web-based software called the Radar Data Viewer Project. With plans to add more files and animations in the future, it will initially concentrate on showing radar data files in a B-Scan format. Additionally, the web application will provide important information like the transmission frequency, bandwidth, maximum range, and the time and location of the radar data collection. Despite its limited capabilities, the initial version of the software will be available on various operating systems, including Windows, Ubuntu, and others, or in web browsers. Furthermore, in further software iterations, we intend to incorporate sophisticated data analysis functionalities such as spectrum and parameter extraction methods.

Objective: Our objective is to create an intuitive and user-friendly web application for HF and X-Band radar data visualization that will make it simpler for users to comprehend and deal with radar data by enabling them to upload, browse, and analyze radar data files with ease.

Scope of MVP

Included Features:

1. A single HF radar file needs to be displayed in B-Scan Format. It would be good if the system could display multiple consecutive files either separately or as an animation.
2. Display key parameters of a data file such as the information of data acquisition, transmission frequency, bandwidth, particularly for HF radar, and location of the data for each one.
3. The system should primarily run on the web. However, If the system architecture is designed properly, it is possible to run this in major OS like Windows, Linux, etc.
4. The system architecture should be able to handle algorithms with various wave spectrums and parameter extraction.

Excluded Features:

1. Working with Both HF radar data files (in SORT files) and X-Band Radar Data, visualizing them in different formats simultaneously, such as B-Scan and scan-converted formats, is not required.
2. For now, the software is not required to run on cross-platforms such as Windows, Linux, etc
3. Implementation of various wave spectrums and parameter extraction algorithms is not required now.

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

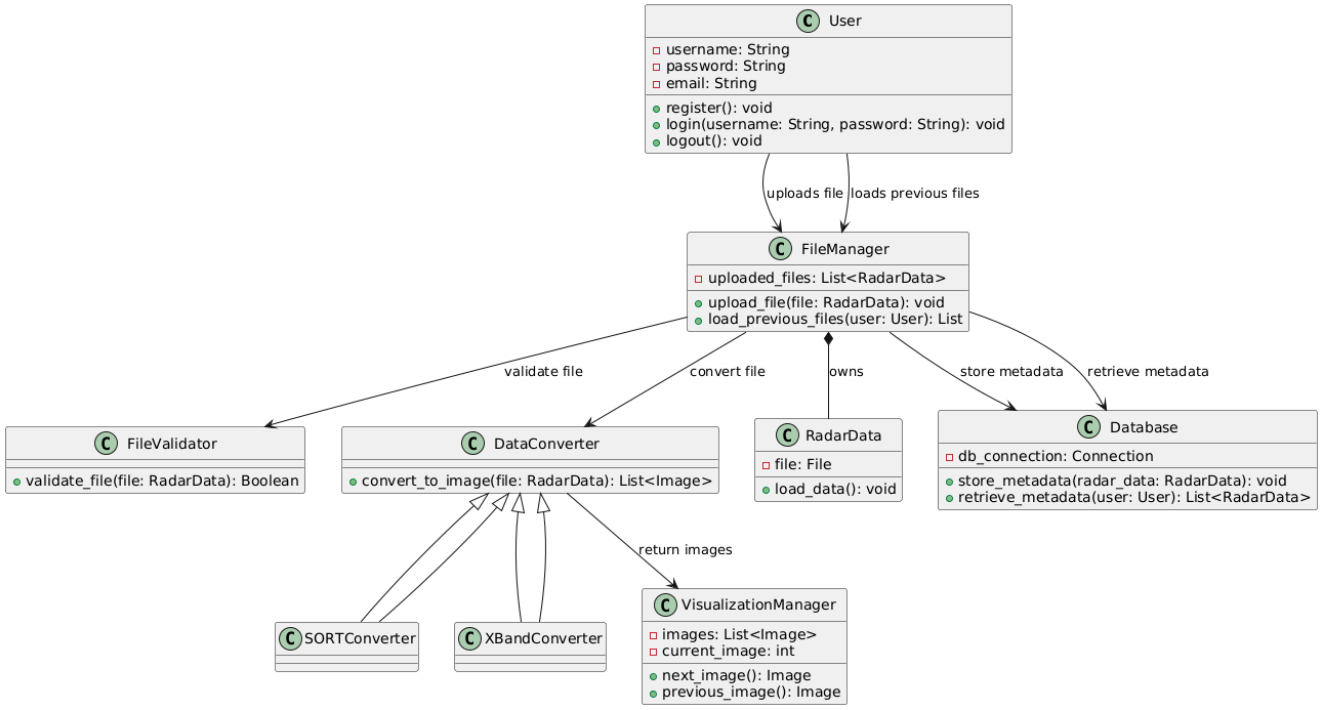


Fig.: Class Diagram

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:

a. The user logs in to the system using the Login Service.

b. 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.

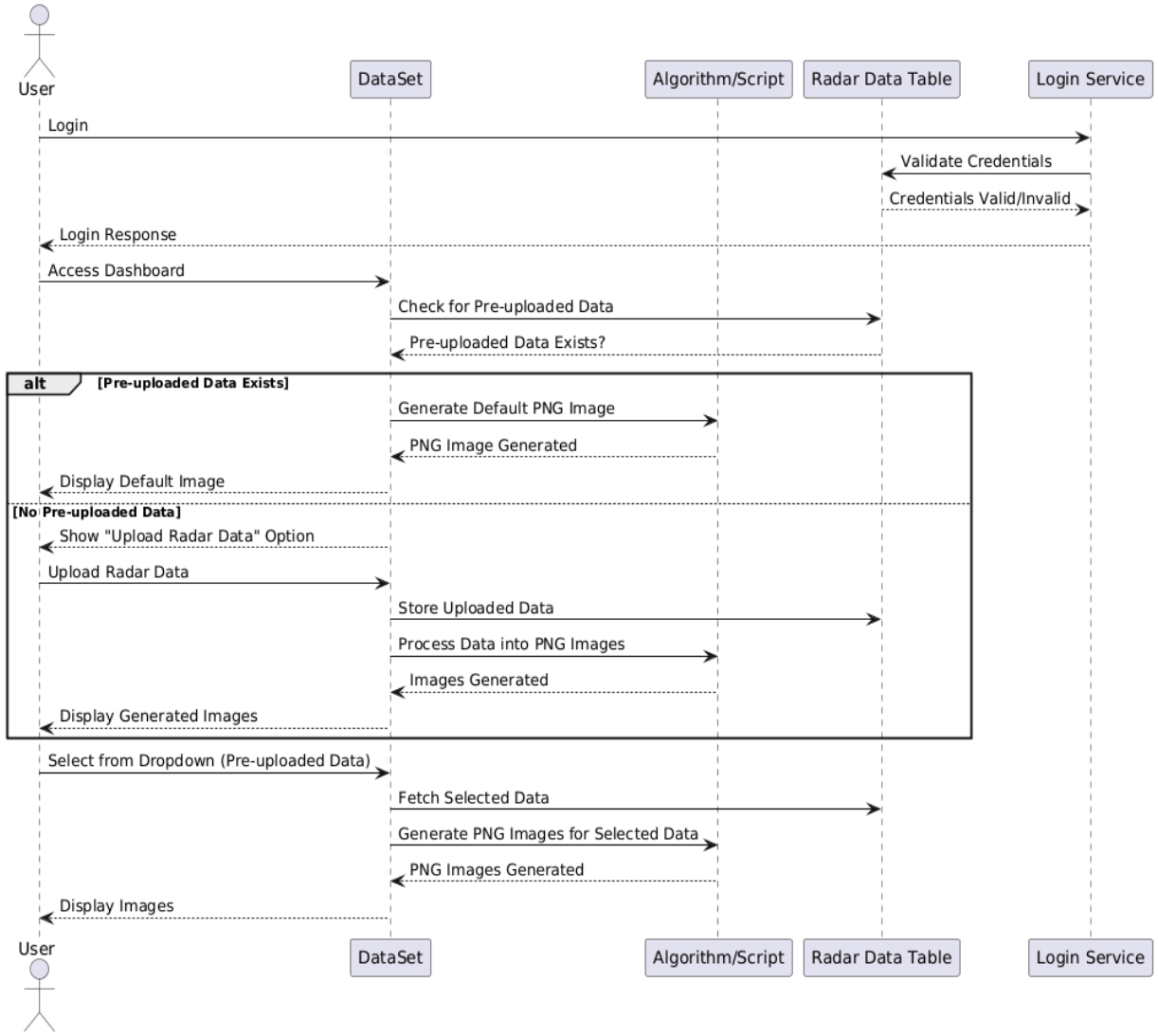


Fig.: Full Process Sequence Diagram

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:

a. The user enters their login credentials.
b. The system sends these credentials to the Login Service.
c. The Login Service communicates with the User Data Table to validate the credentials.
d. If the credentials are valid, a success notification is sent to the user via the Notification Service.
e. 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.

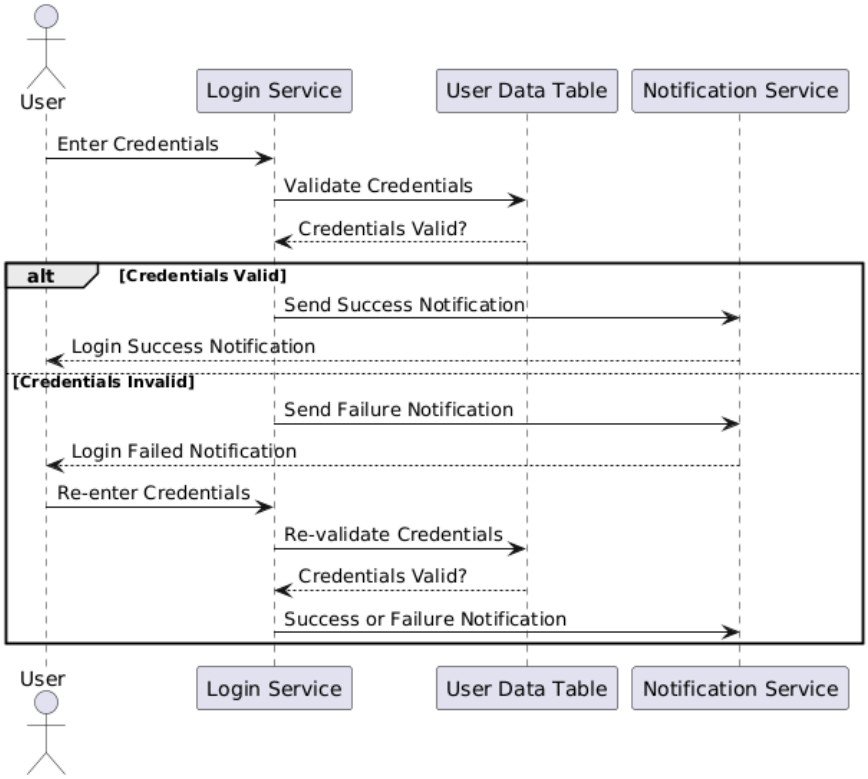


Fig.: Login Service Sequence Diagram

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:

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

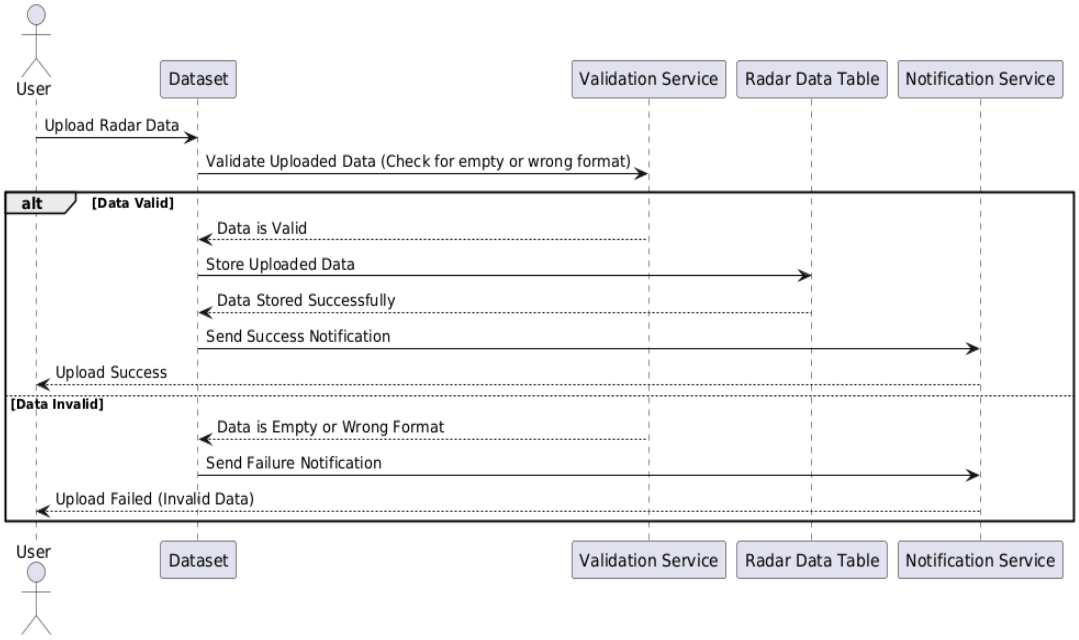


Fig.: Data Upload Validation Sequence Diagram

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:

a. The user uploads radar data through the dashboard.

b. The Dataset service stores the uploaded data in the Radar Data Table.
c. 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.

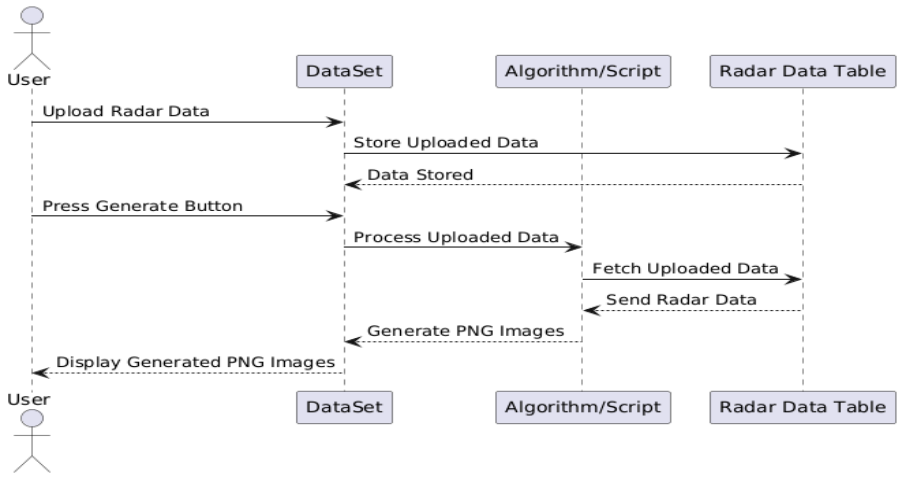


Fig.: Upload and Generate Image Sequence Diagram

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.

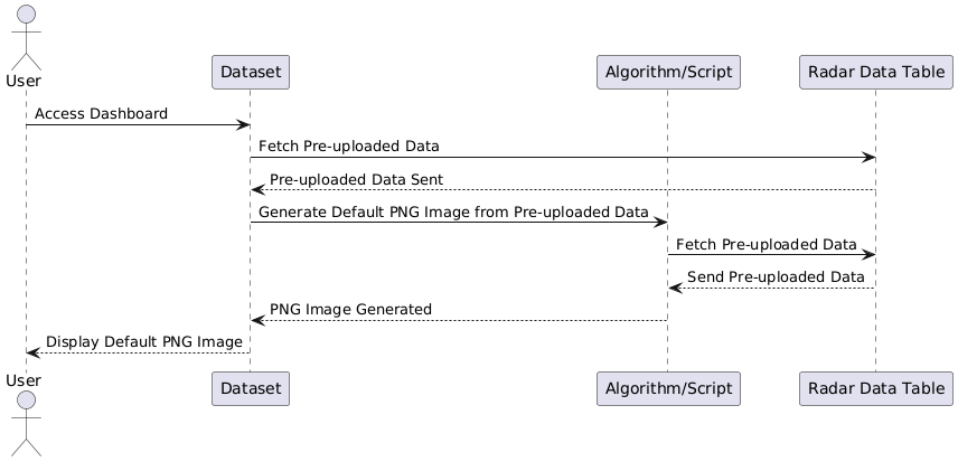


Fig: Pre-uploaded Data Sequence Diagram

System Architecture

Technologies:

1. **Backend:** Django (Python)
2. **Database:** SQLite (Django's default database)
3. **Frontend:** React.js (for building an interactive UI)
4. **Data Processing Libraries:**
    a. NumPy - Used for effective data processing, array manipulations, and numerical calculations, especially when managing radar data.
    b. SciPy Signal - Offers signal processing features, including the ability to apply window functions (such as the Hamming window) to radar data.
    c. SciPy NDImage - utilized for multi-dimensional image processing, including scaling displays of radar data.
    d. Pillow - An Image-handling library (such as one for loading, modifying, and storing radar visualizations)
    e. io and base64 - Python standard libraries for data encoding and decoding (input and output both).
    f. Logging - Used for debugging and monitoring the application's behavior.
    g. Re - Python's regex module for string manipulation.
5. **APIs:** RESTful APIs for seamless frontend-backend communication

System Flow Diagram:

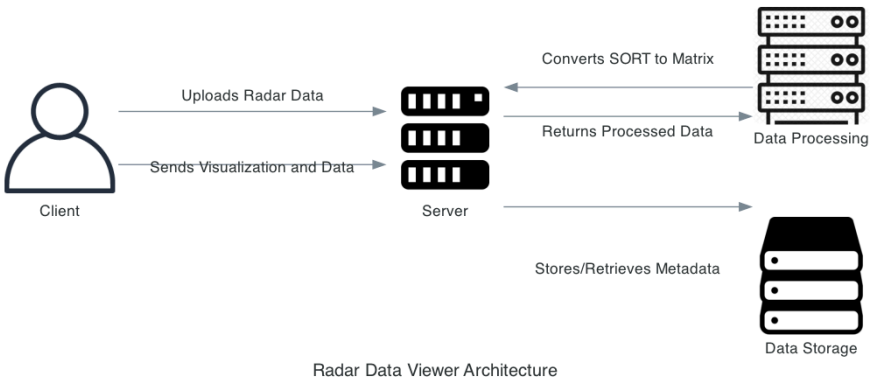


Fig.: High-level architecture

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 SQLite database 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.

Functional Features

1. **File Upload Process**
    a. **Registration:**
        i. A super admin will be created by default in the system.
        ii. The registration page will only be accessible by the super admin, who should be able to create users with credentials such as username and password.
    b. **Landing Page:**
        i. The system's landing page is a login page where users can log in using their credentials.
        ii. The user uses their credentials and the system verifies the data from the server to give authentication access.
        iii. As mentioned above, if the user is a super admin, the user will be able to see an option to view the Registration page to create new user with credentials.
    c. **Loaded Files History and File Upload**
        i. The system routes to the history files page where all the previously uploaded SORT files are shown on a page for the logged-in user. The user can select any of those previously uploaded files and can view all the generated images from the SORT file.
        ii. The user will also have the option to upload any new SORT file as per the choice of the user
    d. **File Selection:**
        i. When the user clicks the Upload File button, the system displays a file picker window to select files from the user's local machine.
        ii. Every file is uploaded based on users. i.e. the uploaded file will be tagged with the username so that every time the user logs in, the system can show all the previously uploaded files.
    e. **File Validation:**
        i. If a non-SORT file is uploaded, the system throws an error:
        “The provided file is not allowed. Please upload a valid SORT file.”
        ii. If a valid SORT file is selected, the file is processed further.

2. Data Transfer to Backend

a. **Frontend Action:**

i. The credentials with username and password are sent in JSON format to the server for processing and authorization.

ii. The selected SORT file is sent to the backend server with the username and other data in JSON format via RESTful API.

3. Backend Processing

a. **Data Processing:**

i. The backend uses libraries like NumPy, SciPy, and Pillow to process the SORT file, converting its data into a matrix format.

b. **Metadata Storage:**

i. Metadata and processed results are stored in the SQLite database.

ii. User-specific data was stored in localStorage to maintain the user priority such as showing all the previously uploaded files. LocalStorage contains data such as user token, username, firstName, lastName, and isLoggedIn value

c. **Image Generation:**

i. Generated images are encoded in base64 format for transmission.

d. **Response to Frontend:**

i. The server provides an authentication response in JSON format whether the the user was authenticated successfully or not.

ii. The backend sends the processed data, including encoded images and metadata, back to the frontend.

iii. The backend server provides response of various API in JSON format such as previously uploaded files of a particular user, all users, multiple images from SORT files and others.

4. Slideshow Feature

a. **Frontend Actions:**
   i. The front end decodes the base64 images provided by the backend.
   ii. All the decoded images and metadata are displayed in a slideshow format.

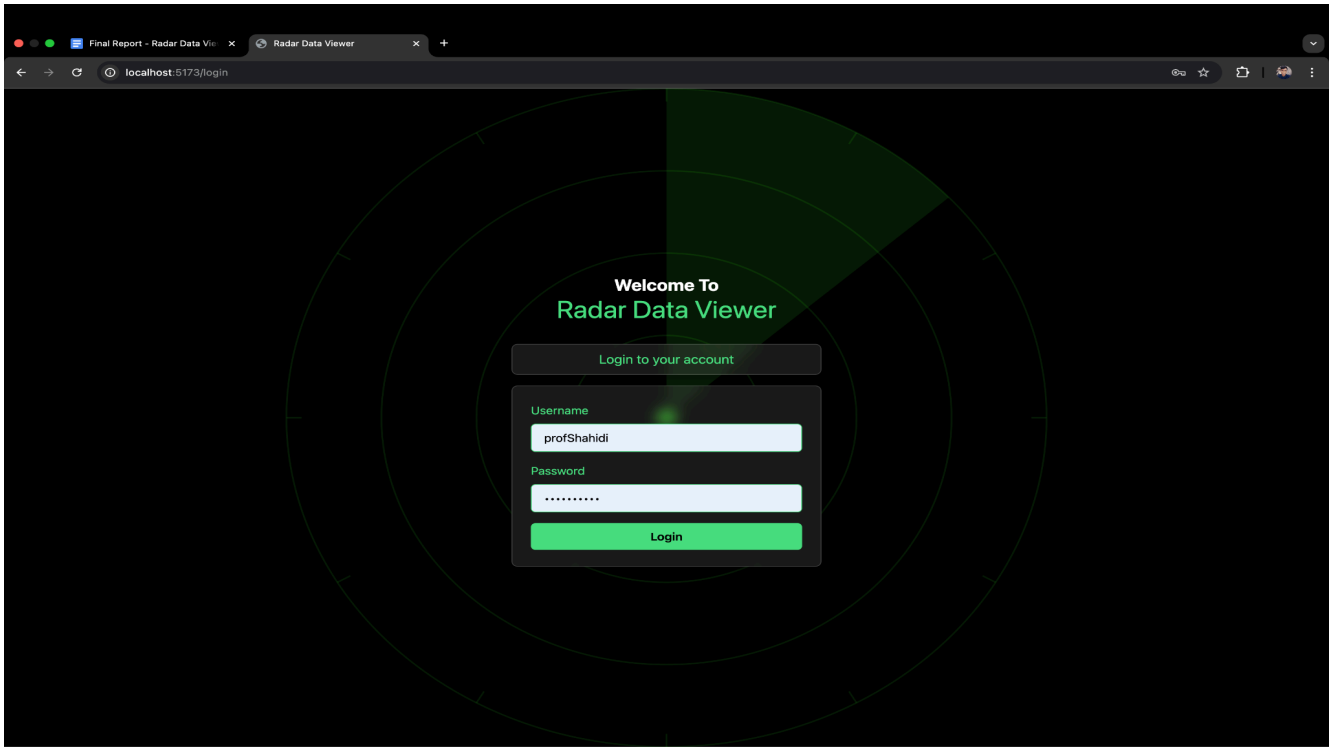
b. **User Controls:**
   i. Play/Pause: Users can start or stop the slideshow at any point.
   ii. Forward/Backward:
      - A Forward button lets users skip to the next image.
      - A Backward button allows navigation to previous images.

Additional Features

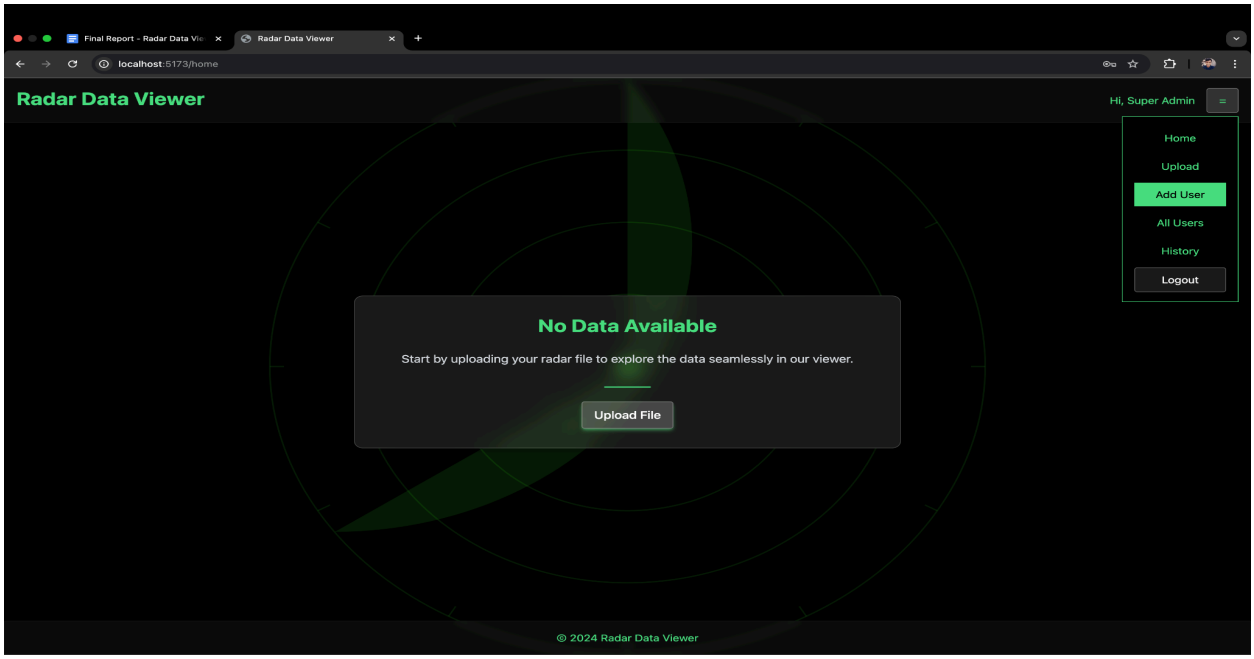
- The CI/CD pipeline has been implemented with build error safety and test case integration.
- More than 22 unit test cases have been written in Django to check the functional features such as the User module and File upload module separately.
- The CI/CD build phase checks the project build errors such as compilation errors and run time environment errors. If the build is completed successfully, it goes to the next phase ‘test phase’.
- In the test phase, the CI/CD pipeline runs all the test cases from the tests.py file and can only forward to the deployment phase if all the test cases are passed.

Design Mockups

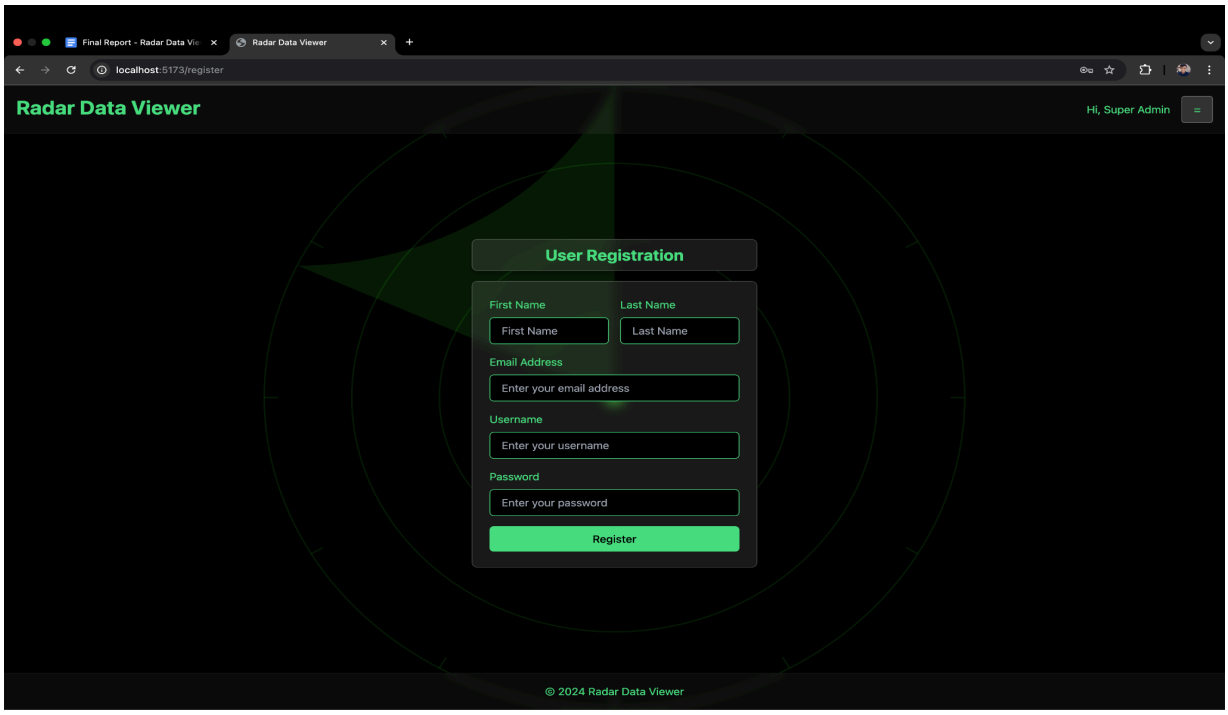
Landing Login Page:



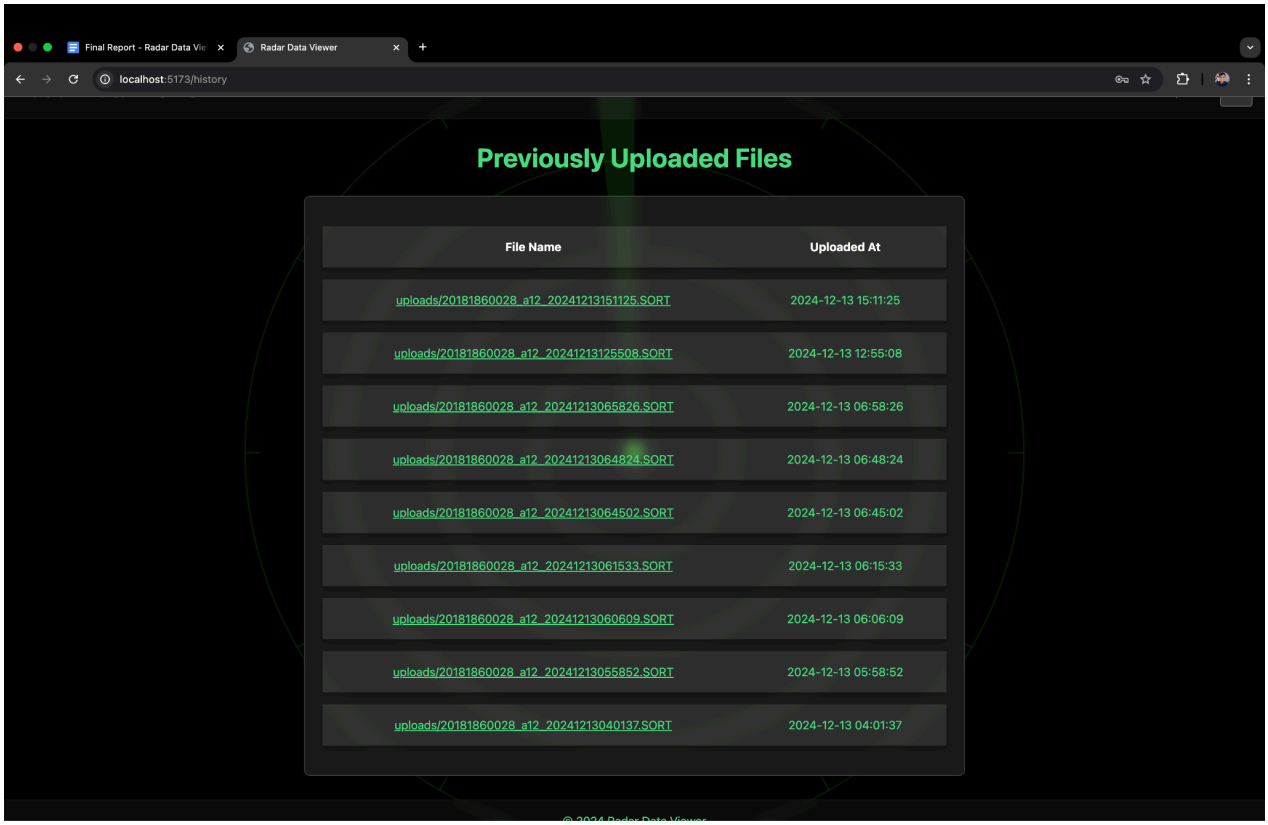
Super Admin with Register option:



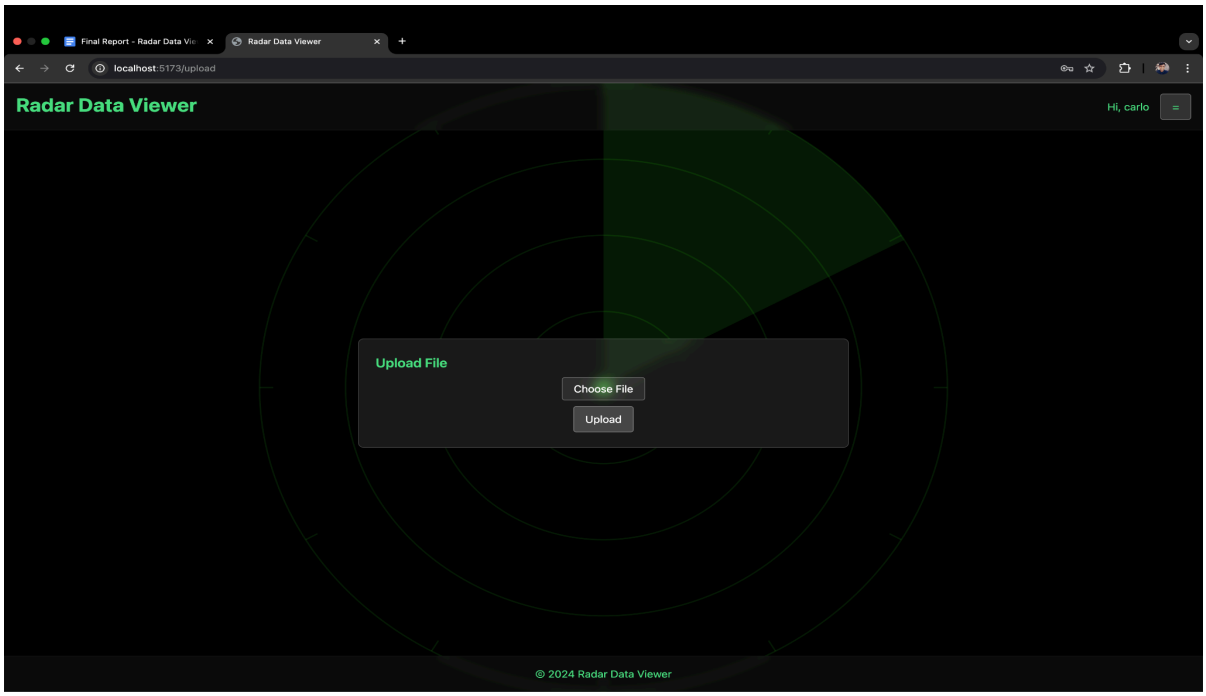
Registration Page:



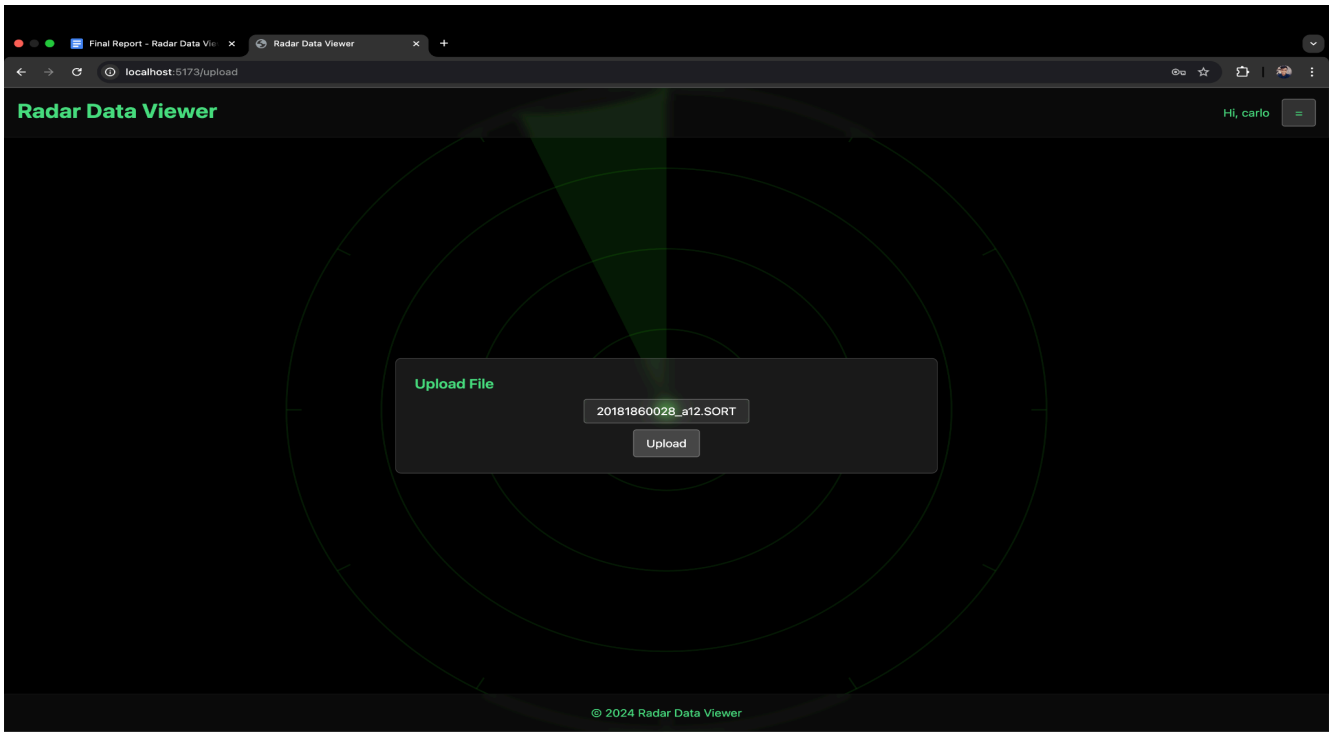
Regular User with Loaded Files:



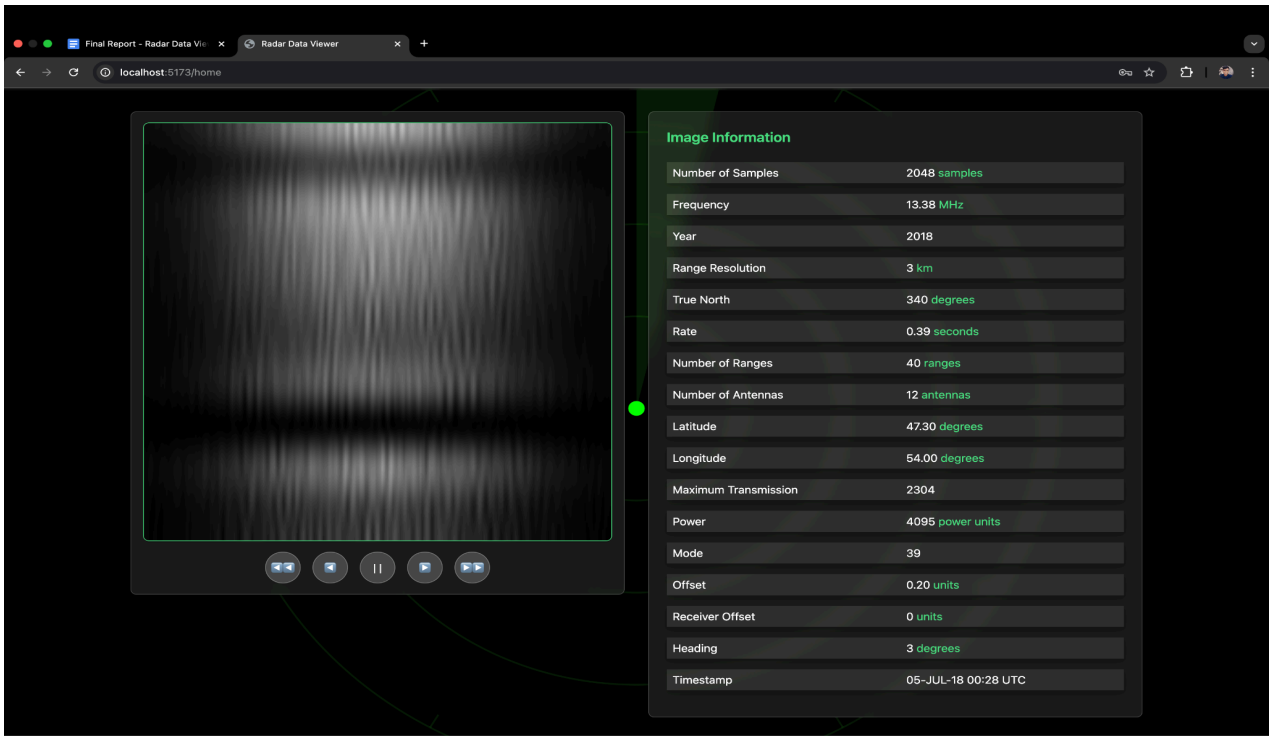
File Upload:



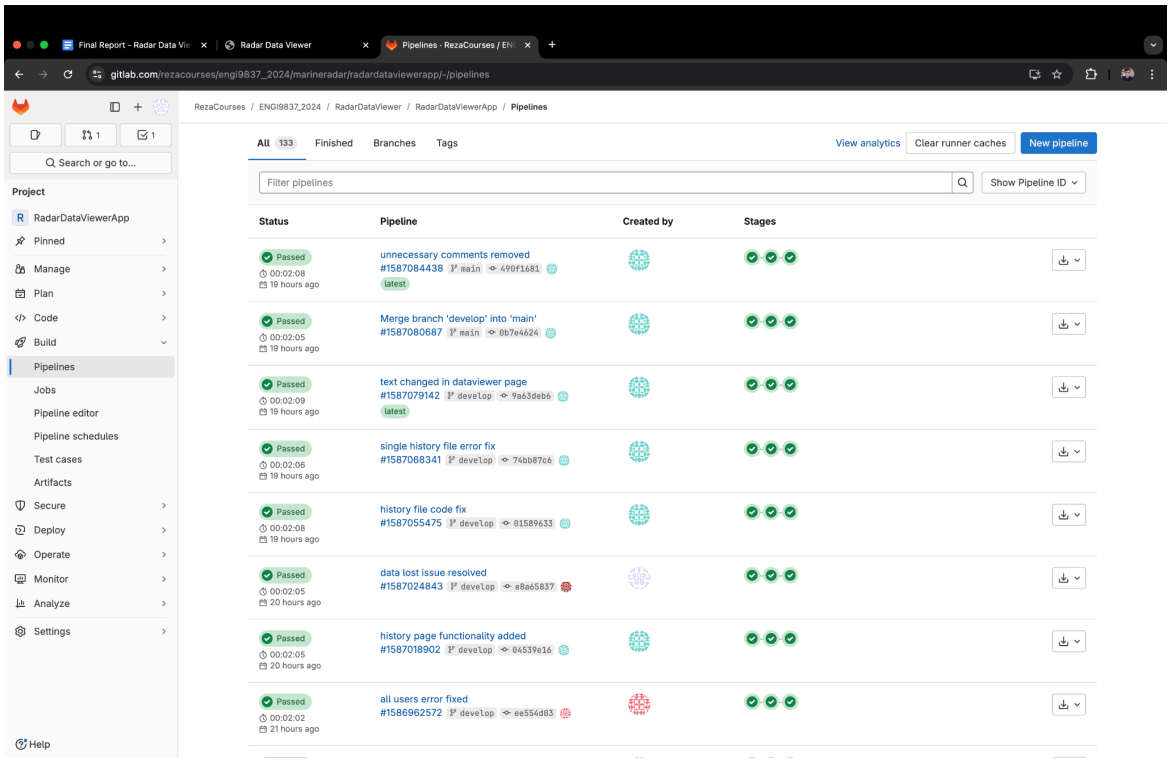
Uploaded File:

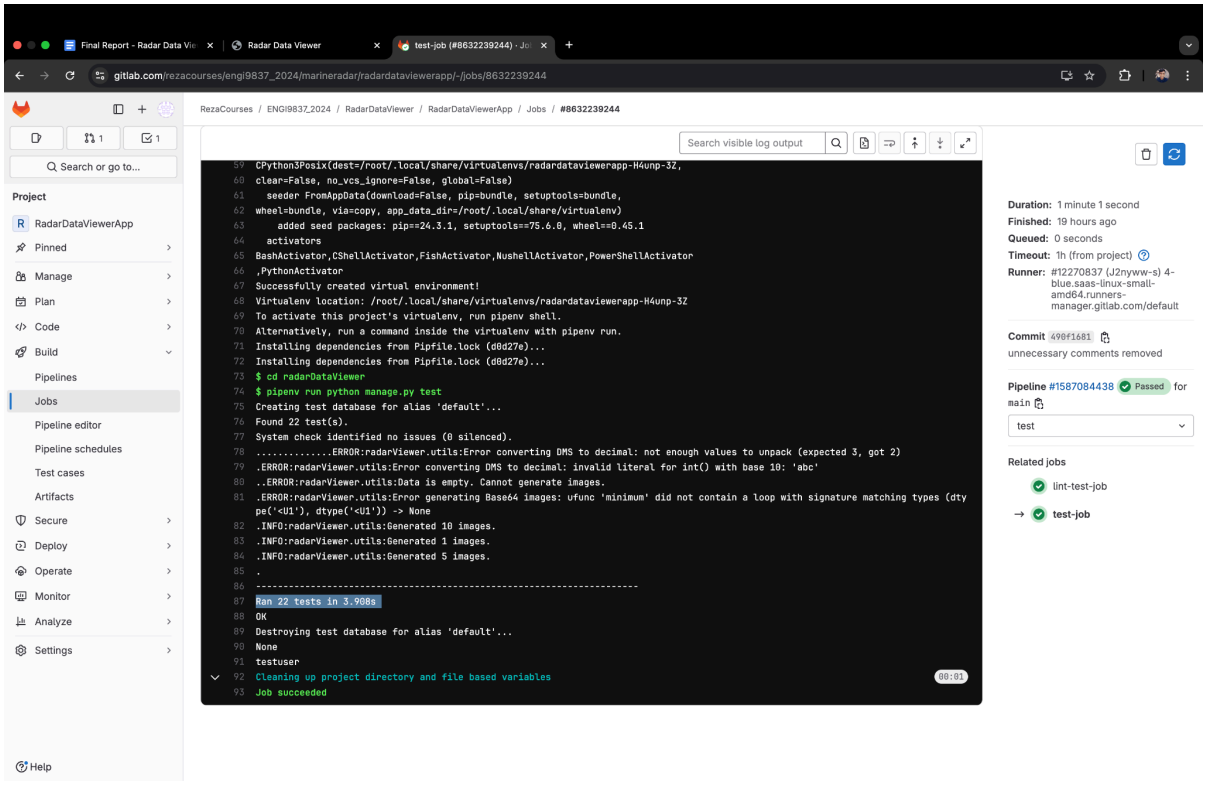


Images With MetaData and Slideshow, Play-Pause-Revert Features :



CI/CD Integration with Test Cases (More than 22 Unit Test Cases):





Test case coverage

This section provides an overview of the test cases implemented for the Radar Data Viewer project. These test cases cover various functionalities, including model behavior, utility functions, image generation, and user authentication. The primary objective is to ensure the system meets its functional and non-functional requirements.

The main objectives of testing are:

- To validate the behavior of the RadarFile model and its related utility functions.
- To ensure the data transformation and image generation processes function as expected.
- To verify the correctness and security of the user authentication mechanism.
- To identify and resolve potential issues in file handling, utility functions, and user login scenarios.

Scope:

The test cases cover:

1. **Model Testing:**
    - Validation of file path generation using timestamps.
    - Verification of the RadarFile model's file storage and retrieval behavior.
    - Testing of string representation for the RadarFile model.
2. **Utility Function Testing:**
    - Transformation of DMS (Degrees, Minutes, Seconds) coordinates into decimal format.
    - Generation of base64-encoded images from numerical data arrays.
    - Handling of edge cases and invalid inputs in utility functions.
3. **User Authentication Testing:**
    - Validation of login functionality with various combinations of usernames and passwords.
    - Ensuring robust handling of edge cases like empty or invalid input fields.

Test Cases Description:

Model Testing

Samples:

- **Test Case 1:** Validation of the timestamped file path function.
    - **Description:** Tests if file paths are generated correctly with timestamps and the proper directory structure.
    - **Result:** Pass.

- **Test Case 2:** Testing RadarFile model field behavior.
    - **Description:** Validates file upload and checks metadata such as uploaded_at.
    - **Result:** Pass.
- **Test Case 3:** String representation of RadarFile.
    - **Description:** Verifies the __str__ method outputs the correct file name.
    - **Result:** Pass.

Utility Function Testing

Samples:

- **Test Case 1:** Conversion of DMS to decimal.
    - **Description:** Tests valid and invalid DMS inputs, ensuring accurate conversion or appropriate error handling.
    - **Result:** Pass.
- **Test Case 2:** Generation of base64-encoded images.
    - **Description:** Verifies the conversion of numerical data arrays into base64-encoded PNG images and tests edge cases.
    - **Result:** Pass.

User Authentication Testing

Samples:

- **Test Case 1:** Valid user with correct password.
    - **Description:** Ensures a valid user can log in successfully.
    - **Result:** Pass.
- **Test Case 2:** Invalid user or incorrect password combinations.
    - **Description:** Tests various invalid login scenarios to ensure the system denies access appropriately.
    - **Result:** Pass.
- **Test Case 3:** Handling of empty input fields.
    - **Description:** Verifies login functionality when fields are left empty.
    - **Result:** Pass.

Summary

- **Total Test Cases Implemented:** 24
- **Test Results:** All test cases passed successfully.
- **Tools Used:**
    - Python unittest framework.

- Django test utilities like SimpleUploadedFile and authenticate.

Observations

- The file-handling mechanism in the RadarFile model is robust and functions as expected.
- The utility functions handle valid, edge, and invalid inputs gracefully.
- The authentication system effectively prevents unauthorized access and handles edge cases.

The implemented test cases confirm the reliability and correctness of the Radar Data Viewer application. The rigorous testing process has ensured adherence to quality standards, paving the way for stable deployment and confident usage.

Testing Approach

To ensure the reliability and usability of the Radar Data Viewer, the following testing approaches were implemented, aligning with the project scope and functionality:

1. **Unit Testing**
    a. **Objective**: Validate individual functions and modules.
    b. **Areas Tested**:
        i. SORT file validation and processing logic in the backend.
        ii. Conversion of radar data to matrix format using NumPy and SciPy.
        iii. Metadata extraction and storage in SQLite.
        iv. Base64 encoding and decoding for image generation.

2. **Integration Testing**
    a. **Objective**: Test the seamless interaction between frontend and backend components.
    b. **Areas Tested**:
        i. RESTful API communication between ReactJS frontend and Django backend.
        ii. Backend response accuracy (images, metadata) and proper frontend display in the slideshow.

3. **Frontend Testing**
    a. **Objective**: Ensure proper UI functionality and user experience.
    b. **Areas Tested**:
        i. File upload functionality and validation messages.
        ii. Slideshow controls (play, pause, forward, backward).
        iii. Decoding and rendering base64 images with proper resizing and scaling.

4. **Performance Testing**

a. **Objective**: Evaluate system performance under varying load conditions.
b. **Areas Tested**:
    i. Processing of large radar files without server lag.
    ii. Real-time responsiveness of the slideshow feature for consecutive radar files.

5. **Error Handling Testing**
    a. **Objective**: Ensure stability and user-friendly error reporting.
    b. **Areas Tested**:
        i. Handling invalid or corrupted SORT files.
        ii. Graceful fallback mechanisms for incomplete or inconsistent metadata.
        iii. Robust error messages for unexpected file formats or data.

6. **Cross-Browser Testing**
    a. **Objective**: Verify compatibility with targeted browsers.
    b. **Areas Tested**:
        i. Proper functionality on Google Chrome and Mozilla Firefox using local server setups.

7. **CI/CD Testing**
    a. Objective: To ensure Automated build error checks, run unit tests during pipeline execution and deployment only after passing all test cases
    b. Areas Tested:
        i. Fetching all the test cases from tests.py, implementing all those test cases in virtual environment of Django using python docker image.

Challenges and Mitigations

Challenges Faced:

- Learning to process radar data and optimize images for display.
- Decoding Images for Slideshow in React JS.
- Resizing Images for Visualization
- Play-Pause-Revert Feature Implementation
- Integrating Django Backend in a Virtual Environment
- Dynamic Metadata Extraction: Parsing .SORT files with incomplete or inconsistent metadata.
- Image Normalization: Handling division-by-zero errors when intensity ranges were identical.
- Large File Processing: Managing real-time processing of large radar files without server performance issues.

- Backend-Frontend Integration: Aligning APIs with frontend requirements for seamless functionality.
- Error Handling: Addressing unexpected data formats while maintaining system stability.
- CI/CD: Integrating test cases in GitLab. The GitLab environment does not recognize the python version and none of the python libraries were not recignized in virtual environment created to work in Django

Mitigation Strategies:

- Utilized well-documented libraries like NumPy and SciPy for visualization and n-dimensional Image handling.
- Explored online communities to learn how to effectively handle base64-encoded images in ReactJS.
- Utilized CSS techniques to maximize window space for image visualization and ensure proper shaping and scaling of images.
- Researched traditional implementations of play and pause functionality through JavaScript forums and ReactJS documentation.
- Referred to the official Django documentation to set up Django within a virtual environment and run in the local server.
- Dynamic Metadata Extraction: Implemented heuristic-based parsing and detailed logging to handle inconsistencies.
- Image Normalization: Added fallback logic using Pillow for adaptive image processing.
- Large File Processing: Used asynchronous processing and memory optimization for better performance.
- Backend-Frontend Integration: Collaborated with the frontend team and conducted iterative testing for smooth integration.
- Error Handling: Designed specific exception handling and custom error messages for debugging.
- Used Python Docker Image to solve Python version problem. Used tests.py file to fetch all the test cases and installed all the dependencies from PipFile as we have used virtual environment.

Improvements After Minimum Viable Product (MVP)

- User module integration for super admin registration and log in for each user for authentication
- Date and time for each image were added for data visualization.
- Units were added in meta data section
- User would be able to see all the previously uploaded files and it’s history with necessary information such as ‘file name’ and ‘uploaded at value

Future Scope

- User profiling update and users should be able to edit their own profile.
- The super admin can make the regular user super admin and also should be able to make any user Active/Inactive. On contrary, An inactive user would not be able to sign in the system and will be redirected to contact with the administrator.
- User should be able to delete particular file from the previously loaded history files and should be able to customize their own history list.
- The system should be able to work with Both HF radar data files (in SORT files) and X-Band Radar Data and formatting them in both B-Scan format and scan converted formats.
- Various wave spectrums and parameter extraction algorithms will be implemented.

Conclusion

An easy-to-use web platform for HF and X-Band radar data visualization and analysis will be provided via the Radar Data Viewer. The Minimum Viable Product, which allows users to see individual radar files in B-Scan format and important metadata, will be our first emphasis. By using more sophisticated visualization techniques and modular integration for wave spectrum analysis and parameter extraction, we intend to gradually increase the project's capabilities.