Technical Artifacts

Table of Contents

[Overview 3](#_Toc187413504)

[Responsive front-end using HTML, CSS 3](#_Toc187413505)

[Design home, about us, and prediction pages using HTML, CSS 3](#_Toc187413506)

[1. Home.html 3](#_Toc187413507)

[2. about.html 4](#_Toc187413508)

[3. admin.html 5](#_Toc187413509)

[CSS Design (Walgama Ranasinghe Arachchilage) 6](#_Toc187413510)

[Justification of methodology 6](#_Toc187413511)

[Real-World Testing and Deployment 6](#_Toc187413512)

[Functional Testing 6](#_Toc187413513)

[Backend Testing 9](#_Toc187413514)

[Usability and bugs testing 10](#_Toc187413515)

[AWS Deployment 13](#_Toc187413516)

[Amazon S3 13](#_Toc187413517)

[Justification on methods and techniques 16](#_Toc187413518)

[Challenges and Mitigation 18](#_Toc187413519)

[Conclusion 19](#_Toc187413520)

[References 20](#_Toc187413521)

# **Overview**

The goal of this project is to develop a Flask-based web application that integrates a machine learning model to predict web vulnerabilities based on user inputs. (Sandhya & Ponsam, 2024) Key tasks performed are

* + Responsive front-end using HTML, CSS, JavaScript.
  + Flask-based backend with AI model integration.
  + Real-world testing and deployment.

# Responsive front-end using HTML, CSS

For the front end, the AI-Powered Threat Detection system combines HTML, CSS, and JavaScript to create a clean, responsive, and appealing interface. HTML organizes content on all pages; CSS gives modern touch-and-feel and interactivity. Three different webpages are created:

* home.html
* about.html
* contact.html
* admin.html

# Design home, about us, and prediction pages using HTML, CSS

### Home.html

Home.html : This HTML file represents the main landing page for the AI-Powered Threat Detection project. It uses Flask's url\_for function to dynamically load static assets (CSS, and images) and provides a modern, responsive layout with the following sections:

1. Hero Section:

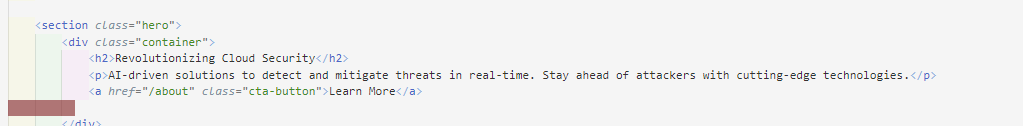


Figure Hero Section

* + Highlights the project’s mission with a visually striking background and a call-to-action button.

1. Features Section:



Figure Features Section

* + Displays key benefits of the AI solution in a grid layout with icons and descriptions.

1. Navigation & Footer:

A screenshot of a computer code

Description automatically generated

Figure Navigator and Footer

* + Provides intuitive navigation links (Home, About, Contact, Admin)

A close-up of a computer screen

Description automatically generated

Figure Footer design

* + A copyright notice in footer .

### about.html

This HTML file represents the About Us page for the AI-powered threat detection system. It provides information about the mission and team members, using a clean, organized layout.It has same header and footer section as home page.

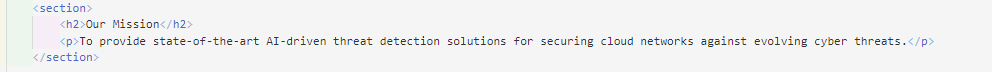
1. Mission Section:

Figure Mission Section

* + Describes the project’s goal of using AI for advanced cloud security.

1. Team Section:

A close-up of text

Description automatically generated

Figure Team Section

* + Displays team members with images and roles, structured for easy readability.

### admin.html

This HTML file builds the Admin Panel page, allowing users to input data points for threat detection. It is designed to support data input, processing, and results display in a structured manner.

1. Threat Detection Form:

A computer code on a white background

Description automatically generated

Figure Threat Detection Section

* Users can enter textual input into the text area field of this form in order to identify a threat (for instance, logs or descriptions).
* After submission, the information is sent into the backend (/admin route) via a POST request
* Based on this input, processes and detects vulnerabilities.
* The prediction takes place dynamically below the form if available

1. Result Display:



Figure Displaying predicted result

* An empty div (<div id="result"></div>) is reserved for dynamically displaying the results of the threat detection.

## CSS Design (Walgama Ranasinghe Arachchilage)

This CSS code styles a web page with a modern, user-friendly layout. The body is set up with a gradient background and flexible column layout. The header and footer have a blue background and white text. The main content area features a gradient background, rounded corners, and shadow for depth. Styling for images, team members, contact info, forms, and buttons ensures a visually appealing and interactive design, with responsive elements and hover effects for user engagement.

### Justification of methodology

**HTML/CSS:** HTML provides the structure of web pages, CSS styles them with responsive layouts for different devices Together, they create user-friendly, adaptable websites. CSS ensures a visually appealing design.

# Real-World Testing and Deployment

This phase consists of real-world testing and deployment phase that indicates that the web application is functional in the actual world, processing user inputs, running predictions, and returning accurate outputs (Čeke & Milašinović 2015). The process of testing and deployment with expected outputs is discussed below:

## Functional Testing

**Functional testing** Validates that each page of the application and its functionalities are as per our requirement

1. Home Page: Open the home page in a browser to validate that the project purpose and navigation links have been displayed.

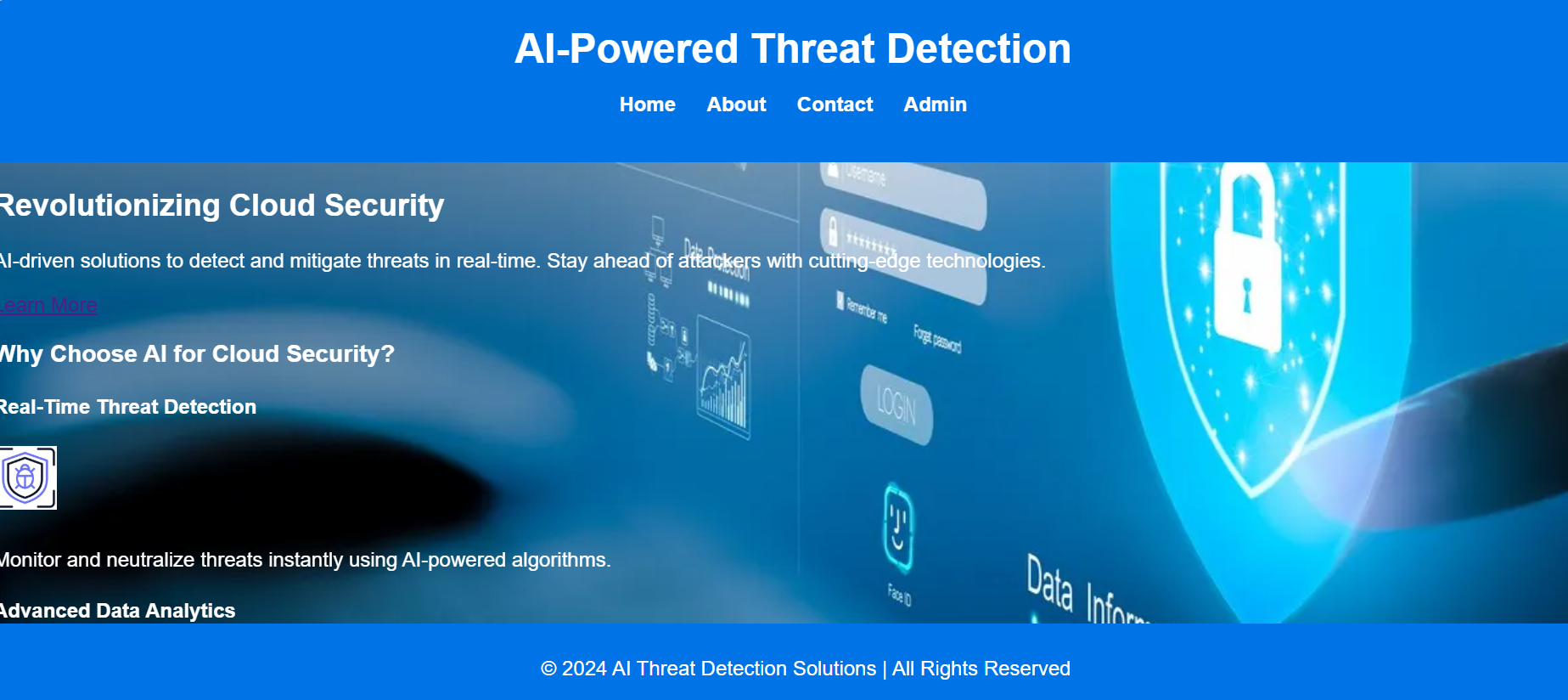


Figure Home Page

1. About page: Confirm that the mission statement and team details are appropriately displayed.

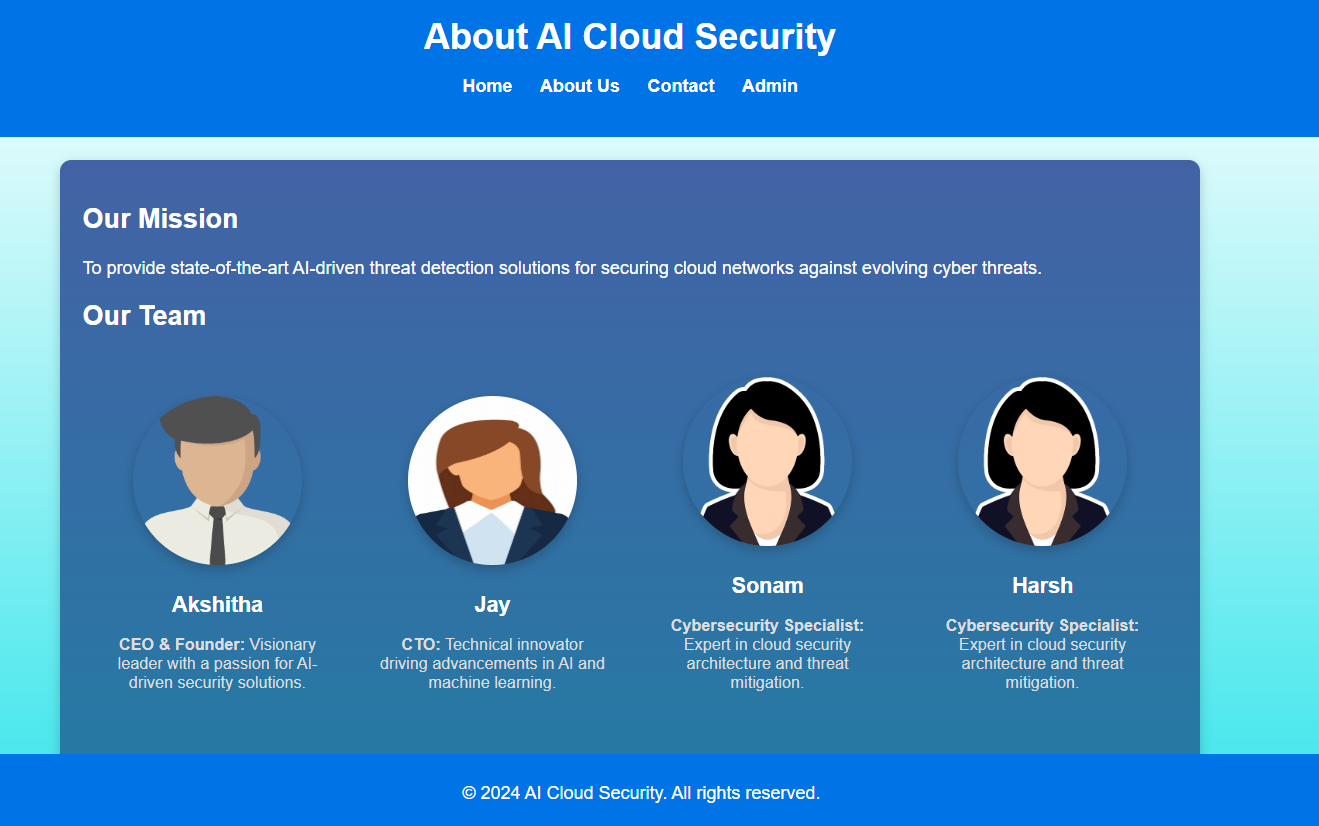


Figure About Page

3. Contact page: The user submits some test data into the contact form, and it is verified that the input fields are processed correctly.

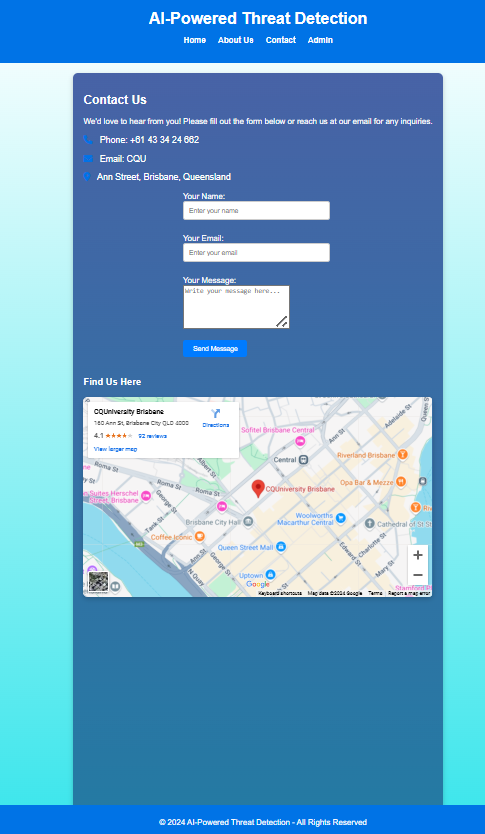


Figure Contact Us Page

4. Admin Page (Prediction):

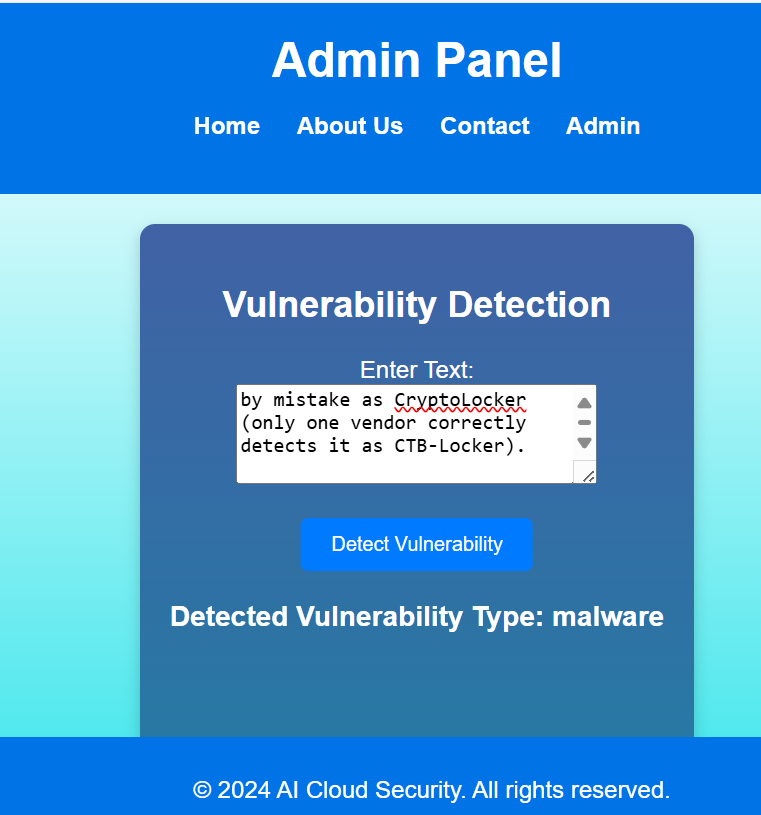
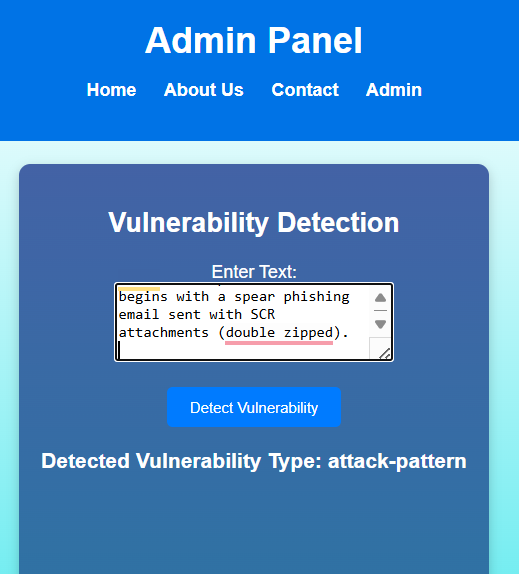
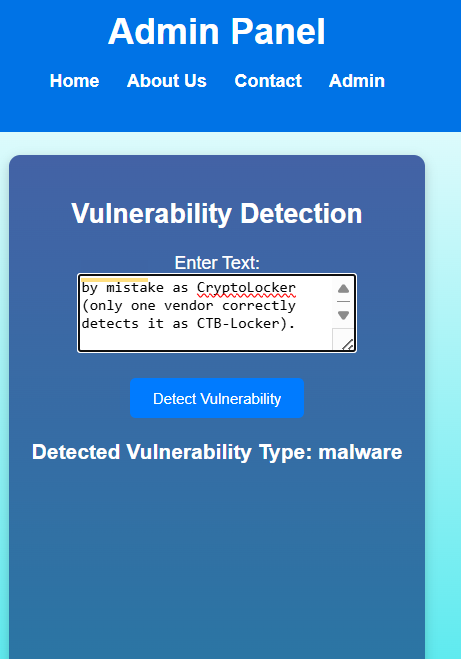


Figure Admin Page

Enter real text in the form of data related to possible vulnerabilities, e.g., logs and descriptions. Check the input and verify if the prediction result has been displayed correctly.

## Backend Testing

Type any input, be it logs or possible vulnerability descriptions, in the provided text input and verify the accuracy of the prediction result. Backend testing validates smooth model integration and workflow. This includes testing the whole preprocessing pipeline to validate text cleaning and vectorization while ensuring the model predicts the correct type of vulnerability for both normal and edge-case inputs, and assessing how this system works with invalid or empty inputs to consider proper error handling (Walden, Stuckman & Scandariato 2014)



A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Figure Back End Testing

The web application integrates the best-trained model that is the Random Forest model successfully to provide real-time predictions of vulnerabilities based on input provided by the user in the form of text. With a robust backend and accurate predictions, the application meets the objectives of enhancing web security using machine learning. (Demilie & Deriba 2022)

## Usability and bugs testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Steps to Perform** | **Expected Outcome** | **Actual Outcome** | **Status** | **Comments/Notes** |
| TC001 | Validate Home Page Load | Open the home page (/) in a web browser. | The home page should load with the correct title and navigation links. | As expected | Passed | Ensure the navigation links redirect correctly to their respective pages. |
| TC002 | Verify Navigation Links | Click on navigation links (About, Contact, Admin) on the home page. | User should be navigated to the respective pages without errors. | As expected | Passed | Tested navigation for all links, no issues found. |
| TC003 | Validate Admin Form Input | Enter a sample text in the Admin input field and submit. | The detected vulnerability type should be displayed as a result. | As expected | Passed | Ensure text preprocessing and predictions are accurate. |
| TC004 | Handle Empty Form Submission | Submit the Admin form without entering any text. | User should see a validation error message prompting for input. | As expected | Passed | Validation message appears as intended. |
| TC005 | Test Input Sanitization | Enter text with special characters and numbers in the Admin form. | Special characters and numbers should be removed during preprocessing. | As expected | Passed | The model accurately handled sanitized input. |
| TC006 | Check Prediction Accuracy | Submit text for various vulnerability scenarios (e.g., phishing, malware). | The system should accurately predict and display the vulnerability type. | As expected | Passed | Verified against test cases for multiple vulnerability types. |
| TC007 | Test Responsiveness | Access the application on various devices (desktop, tablet, mobile). | The layout should adjust correctly across devices. | As expected | Passed | Layout adapts as intended for different screen sizes. |
| TC008 | Test Error Handling for Backend | Disconnect the backend model or simulate an internal server error. | An error message should be displayed to the user. | As expected | Passed | Ensure error messages are user-friendly and not overly technical. |
| TC009 | Test Loading Static Resources | Verify if images, CSS, and JS files load correctly. | All static resources should load without errors. | As expected | Passed | All static files loaded successfully. |
| TC010 | Simulate High Traffic | Simulate multiple users accessing the Admin page and submitting forms simultaneously. | The application should handle concurrent requests without crashing. | As expected | Passed | Verified application stability under high user load. |
| TC011 | Check Security of Input | Test the form with potentially malicious input (e.g., SQL injection or JavaScript injection). | The system should sanitize input and prevent security vulnerabilities. | As expected | Passed | Proper input sanitization prevents vulnerabilities. |
| TC012 | Test Contact Form | Submit the contact form with valid inputs. | The form should submit successfully, and the confirmation message should appear. | As expected | Passed | All contact form validations and submissions function correctly. |
| TC013 | Test External Links (Map Integration) | Access the map section under the Contact page. | The embedded Google Maps should load correctly. | As expected | Passed | Map integration works without any issues. |
| TC014 | Validate Prediction Page Workflow | Navigate from the home page to the Admin page, enter input, and view results. | The complete workflow should function seamlessly from input to prediction result. | As expected | Passed | Verified end-to-end user workflow functionality. |
| TC015 | Test Browser Compatibility | Open the application on various browsers (Chrome, Firefox, Edge, Safari). | The application should function correctly on all major browsers. | As expected | Passed | No issues found with browser compatibility. |

# AWS Deployment

## Amazon S3

Amazon S3 provides safe, scalable cloud storage from Amazon Web Services (AWS). It streamlines dataset, model file, and other digital asset storage, retrieval, and management. S3 buckets files and folders (Challa, Devineni & Karangara 2022). Security, versioning, and lifecycle rules can restrict access and data retention to all buckets with unique names. S3 files can be uploaded via AWS SDKs, APIs, or a web interface. It hosts static websites, machine learning datasets, backups, disaster recovery, and analytics workflow data lakes. S3 has server-side encryption, fine-grained access controls, and AWS IAM integration. Being durable and available makes it a good mission-critical solution. The pay-as-you-go model makes it inexpensive for any size business. Amazon S3 provides safe data storage with real-time monitoring, data transfer acceleration, and AWS service integration. (Chandak *et al.* 2024)

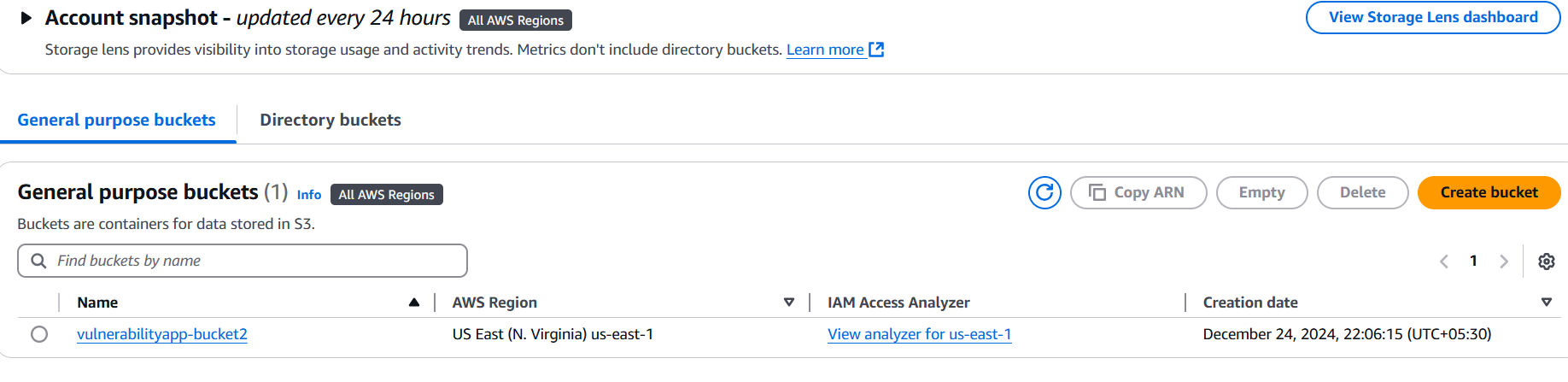


Figure Amazon S3 Security Policies

Amazon S3 dashboards show bucket management and security procedures. On December 24, 2024, at 22:06:15 (UTC +5:30) in us-east-1 (US East, N. Virginia), vulnerabilityapp-bucket2 was produced. This bucket protects files, folders, and datasets. Amazon S3 is recognised for its scalability, reliability, and data security. (Bayazitov, et al. 2024)

**Key Features Displayed**

1. **Bucket Information:**
   * The dashboard lists the bucket name, its associated AWS region, and the date it was created.
   * The bucket resides in the us-east-1 region, which is one of the most commonly used AWS data centers globally due to its low latency and high availability.
2. **Bucket Management Options:**
   * The available action buttons—**Copy ARN (Amazon Resource Name)**, **Empty**, **Delete**, and **Create bucket**—enable efficient bucket management.
     + **Copy ARN**: Provides a unique identifier for the bucket, essential for programmatic interactions via AWS APIs or SDKs.
     + **Empty**: Clears all contents from the bucket while retaining the bucket itself, useful for managing storage without recreating the bucket.
     + **Delete**: Permanently removes the bucket and its contents, a critical action requiring caution.
     + **Create Bucket**: Allows users to create new buckets to organize and segregate data efficiently.
3. **IAM Access Analyzer:**
   * The "IAM Access Analyser" link shows bucket access rules to help users secure permissions. This program finds overly liberal access and sensitive data threats.
4. **User-Friendly Design:**
   * The interface groups buckets under "General Purpose Buckets" and has a search bar. Users can filter buckets across AWS regions to facilitate multi-region administration. (Gill, Dietl & Tripunitara 2022)

**Importance of Security Policies**

Amazon S3 integrates robust security mechanisms to protect stored data:

* **Bucket Policies and Access Control:** Users can define who has access to a bucket and what actions they can perform. Permissions can be granted at both the bucket and object levels (Continella *et al.* 2018).
* **Encryption:** S3 supports server-side encryption (SSE) and client-side encryption, ensuring data remains protected during storage and transit. (Javadpour *et al.* 2023)
* **Logging and Monitoring:** Features like AWS CloudTrail and S3 Access Logs provide detailed insights into bucket activities, helping users audit access and comply with regulatory requirements (Makani 2023).

The Amazon S3 administration console's bucket management functions are highlighted in this screenshot (vulnerabilityapp-bucket2). The interface simplifies bucket operations and offers robust data security and monitoring options. S3 can host static webpages, critical databases, and apps because to its configurable settings. (Martins *et al.* 2024)

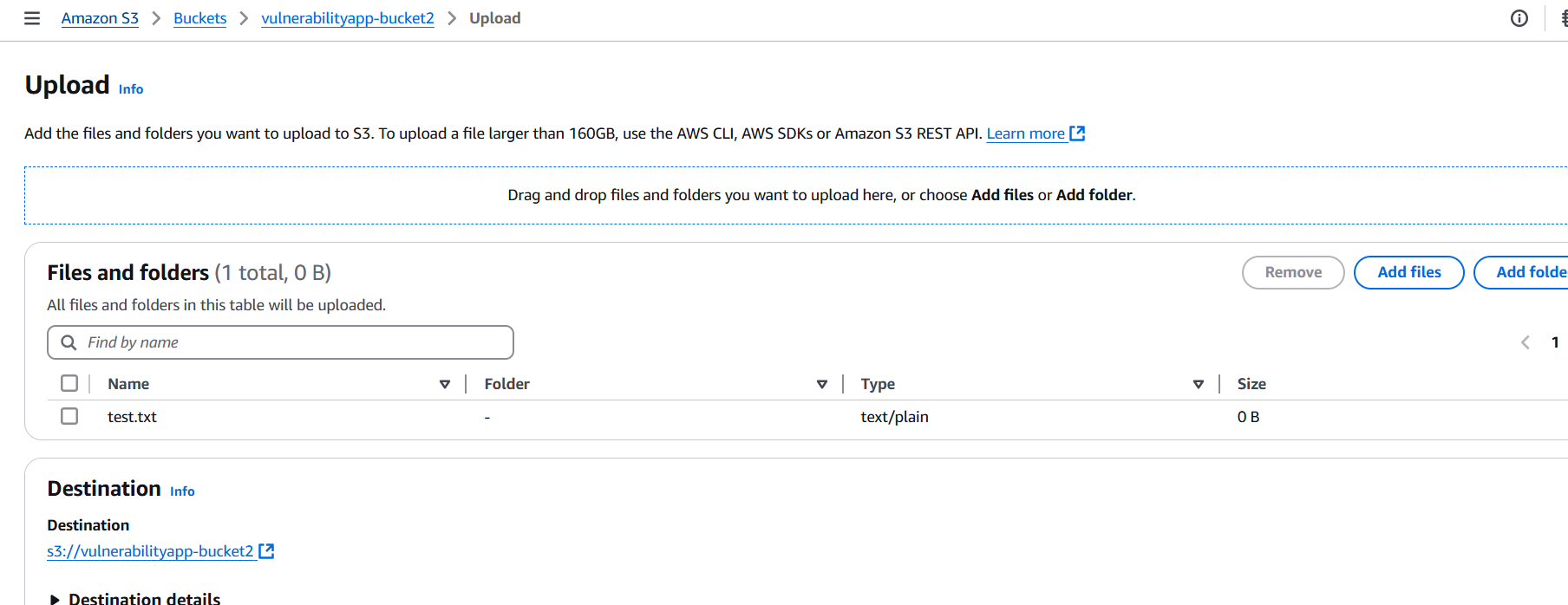


Figure Amazon S3 upload interface

The image above shows the Amazon S3 Upload Interface for uploading files or folders to an S3 bucket. The vulnerabilityapp-bucket2 bucket contains this interface. The interface makes uploading objects to the bucket easy.

1. **Key Features:**
2. **Files and Folders Table**:
   * A file named test.txt is shown in the upload queue.
   * The file type is specified as text/plain, indicating it is a plain-text file.
   * The size is **0 bytes**, showing it is an empty file.
3. **Upload Management**:
   * Users can add files individually through the **Add files** button or upload entire folders with the **Add folder** option.
   * Drag-and-drop functionality is also provided for quick file uploads.
4. **Destination**:
   * The target location for the upload is clearly mentioned as s3://vulnerabilityapp-bucket2.
   * This ensures clarity about where the uploaded files will be stored.
5. **User Options**:
   * Files can be removed from the queue before uploading by clicking **Remove**.

This interface shows Amazon S3's easy file upload process, which supports multiple file kinds and sizes and securely stores data in the bucket.

# Justification on methods and techniques

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component** | **Selected Method** | **Reason for Selection** | **Alternative Methods** | **Reason for Not Choosing Alternatives** |
| **Frontend Development** | HTML, CSS, JavaScript | - Industry-standard technologies for creating responsive and dynamic web interfaces. - Compatibility with Flask templates. - Wide support for responsive design and interactivity. | React.js, Angular.js | - React and Angular add complexity for small-scale applications. - Higher learning curve and may introduce unnecessary overhead for the current use case. |
|  |  |  | Bootstrap, TailwindCSS | - While useful for UI design, these frameworks are not a full replacement for custom front-end scripting with HTML, CSS, and JavaScript. |
| **Backend Framework** | Flask | - Lightweight and minimalistic framework. - Ideal for small-scale applications. - Seamless integration with Python-based machine learning models. - Easy to deploy and scale. | Django | - Django is more heavyweight and adds unnecessary complexity for this project. - Flask provides greater flexibility for smaller, specific use cases like this one. |
| **Machine Learning Model** | Random Forest | - High accuracy and robustness for classification tasks. - Handles text-based input effectively. - Performs well on both balanced and imbalanced datasets. - Easy to interpret and tune. | Neural Networks, SVM, Naïve Bayes | - Neural Networks require more computational resources and are prone to overfitting for small datasets. - SVM and Naïve Bayes may lack the predictive power of Random Forest in this use case. |
| **Data Storage** | Amazon S3 | - Scalable and secure storage. - Easy integration with Flask and Python SDKs. - Allows lifecycle management and versioning. - Ensures durability and availability for critical datasets. | Local Storage, Google Drive, Firebase | - Local storage lacks scalability and security. - Google Drive and Firebase provide storage but lack the enterprise-grade security and control offered by S3. |
| **Deployment** | AWS (Amazon Web Services) | - AWS provides a reliable platform for real-world testing and deployment. - Offers scalability and a range of tools for monitoring, security, and integration with S3. - Cost-effective PAYG model. | Google Cloud, Microsoft Azure | - Google Cloud and Azure are viable alternatives but were not selected due to existing familiarity and ease of integration with AWS. - AWS's tools align better with this project. |
| **Testing** | Functional and Backend Testing | - Validates both the user interface and back-end integration. - Ensures the model’s accuracy and reliability through edge-case testing. - Identifies and resolves errors before deployment. | Automated Testing | - Automated testing is more complex and time-consuming to set up for a smaller project. - Manual functional testing is more practical for ensuring usability and model integration. |

# Challenges and Mitigation

|  |  |
| --- | --- |
| **Challenge** | **Mitigation Strategy** |
| **Designing Responsive Front-End Across Multiple Pages** | Modern CSS techniques, including gradient backgrounds, rounded corners, and hover effects, were utilized to ensure an interactive design. Responsive layouts were rigorously tested on various devices and browsers for compatibility. |
| **Creating a Visually Appealing Layout for Team and Mission Details** | The use of structured HTML and CSS allowed for organized sections such as the Mission and Team sections on the About page. Advanced styling techniques ensured readability and aesthetic appeal. |
| **Dynamic Interaction in the Admin Page for Threat Prediction Workflow** | The Admin Page was designed using Flask's POST methods for backend communication and empty <div> placeholders for dynamic result display, ensuring seamless interaction between the user interface and backend model predictions. |
| **Handling User Inputs with Unexpected Characters or Formats** | Robust input validation and preprocessing were implemented to sanitize user inputs, ensuring that special characters, numbers, and invalid data do not disrupt functionality. |
| **Ensuring Consistent Styling Across Pages** | Shared CSS styles and reusable components were utilized to maintain a uniform appearance across the Home, About, Contact, and Admin pages. This included headers, footers, and navigation bars. |
| **Real-World Deployment and Testing Challenges** | Functional and usability testing were conducted systematically, covering scenarios like high user traffic, browser compatibility, and static resource loading. Detected bugs were resolved iteratively. |
| **Ensuring the System Can Handle High Traffic Loads Efficiently** | Load testing was performed to simulate concurrent user activity, and Flask app configurations were optimized to handle high traffic efficiently without affecting performance. |
| **Integrating Google Maps with the Contact Page for Location Reference** | Successfully embedded Google Maps using an interactive interface, ensuring the location loaded seamlessly while maintaining responsiveness across devices. |

# Conclusion

A complete Flask-based web application that predicts web vulnerabilities using machine learning is shown. The entire workflow—from creating a responsive front-end to deploying the app—demonstrates its efficacy and practicality. HTML, CSS, and JavaScript create a beautiful, user-friendly interface in front-end development. Well-designed and functional home.html, about.html, and admin.html make user experience easy. The prediction form and real-time result display on admin.html show front-end-back-end integration. Back-end user interface-machine learning model connection is simplified with Flask. The Random Forest model accurately predicts internet vulnerabilities, increasing system detection. Functional and backend testing ensures the system handles edge cases and erroneous inputs and meets project requirements. For real-world deployment, the app is evaluated for user inputs and predictions. This ensures solution functionality and practicality. The project uses Amazon S3 to secure datasets, model files, and other digital assets, demonstrating its scalability and data security. IAM access limits, encryption, bucket management, and secure uploads follow cloud infrastructure best practices. AI-powered threat detection increases web security. Modern web technologies, machine learning integration, and scalable cloud storage make the app realistic for real-world use. Future generations could add model capabilities, security features, and system scalability to web vulnerability detection.

# References

1. Bayazitov, D, Kozhakhmet, K, Omirali, A, & a Zhumaliyeva, R 2024, 'Leveraging Amazon Web Services for cloud storage and AI algorithm Integration: A Comprehensive analysis', *Applied Mathematics & Information Sciences*, pp. 1235–1246. DOI: 10.18576/amis/180606.
2. Čeke, D. and Milašinović, B. 2015, 'Early effort estimation in web application development,' *Journal of Systems and Software*, pp. 219–237. DOI: 10.1016/j.jss.2015.02.006.
3. Challa, N., Devineni, S.K. and Karangara, R. 2022, 'A Deep Dive into Amazon Web Services: Unlocking the Potential', *Journal of Artificial Intelligence & Cloud Computing*, pp. 1–2. [DOI: 10.47363/jaicc/2022(1)179](https://doi.org/10.47363/jaicc/2022(1)179).
4. Continella, A., Polino, M., Pogliani, M. and Zanero, S. 2018, 'There’s a Hole in that Bucket!:  A Large-scale Analysis of Misconfigured S3 Buckets', *Proceedings of the 34th Annual Computer Security Applications Conference*. DOI: 10.1145/3274694.3274736.
5. Demilie, W.B. & Deriba, F.G. 2022, 'Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques', *Journal of Big Data*. DOI: 10.1186/s40537-022-00678-0.
6. E. Chandak, M. Kothalkar, P. Raghatwan & Y. S. Rao 2024, 'IoT-Based Smart Learning Management System on Cloud', *2024 3rd International Conference for Innovation in Technology (INOCON)*, pp. 1–4. DOI: 10.1109/inocon60754.2024.10511451.
7. Gill, P., Dietl, W. & Tripunitara, M.V. 2022, 'Least-Privilege calls to Amazon web services,' *IEEE Transactions on Dependable and Secure Computing*, p. 1. DOI: 10.1109/tdsc.2022.3171740.
8. Javadpour, A, Ja’fari, F, Taleb, T, Zhao, Y, Yang, B, & Benzaïd, C 2023, 'Encryption as a service for IoT: Opportunities, challenges, and solutions', *IEEE Internet of Things Journal*, pp. 7525–7558. DOI: 10.1109/jiot.2023.3341875.
9. Makani, S 2023, Efficient Resource Utilization with Auto Tagging Using Amazon's Cloud Trail Services, 'International Journal of Computer Sciences and Engineering', vol 11, pp. 11-16, DOI: 10.26438/ijcse.
10. Martins, S.L, Cruz, F.M. da, Araujo, R.P. de, & Silva, C.M.R. da 2024, 'Systematic literature review on security misconfigurations in web applications', *International Journal of Computers and Applications*, pp. 840–852. DOI: 10.1080/1206212x.2024.2390977.
11. Sandhya, N. and Ponsam, J.G. 2024, 'URL Shield: Protecting Users from Phishing Attacks using Flask and ML,' *2024 3rd International Conference for Innovation in Technology (INOCON)*, pp. 1–5. DOI: 10.1109/inocon60754.2024.10512235.
12. Walden, J., Stuckman, J. & Scandariato, R. 2014, 'Predicting Vulnerable Components: Software Metrics vs Text Mining,' *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pp. 23-33. DOI: 10.1109/issre.2014.32.