Technical Artefacts

**Harsh Kumar Dharmendra bhai Patel**

Table of Contents

[Design of Contact page 3](#_Toc187422034)

[contact.html 3](#_Toc187422035)

[Flask-based backend with AI model integration 5](#_Toc187422036)

[Home Page 5](#_Toc187422037)

[About Page 5](#_Toc187422038)

[Contact Page 6](#_Toc187422039)

[Prediction Page (Admin) 6](#_Toc187422040)

[Deployment and Real-World Use 7](#_Toc187422041)

[AWS Backup Settings: Automating Backups for Disaster Recovery 7](#_Toc187422042)

[Purpose of AWS Backup 7](#_Toc187422043)

[Key Features of AWS Backup: 8](#_Toc187422044)

[Benefits for Disaster Recovery 8](#_Toc187422045)

[Justification of selected methods 10](#_Toc187422046)

[Challenges and Mitigations 13](#_Toc187422047)

[Conclusion 15](#_Toc187422048)

[References 15](#_Toc187422049)

# Design of Contact page

## contact.html

This HTML file creates the Contact Us page for the AI-Powered Threat Detection project. It includes a user-friendly form, essential contact details, and an embedded map for location reference.

1. Contact Form:

A screenshot of a computer

Description automatically generated

Figure 1 Contact Form

* + Allows users to submit inquiries with fields for name, email, and message.

1. Contact Details Section:

A close-up of a computer code

Description automatically generated

Figure 2 Contact Details Section

* + Displays phone, email, and address with icons for better visual appeal.

1. Embedded Map

A close-up of a computer screen

Description automatically generated

Figure 3 Embedded Map

* + Shows the organization's location using Google Maps for easy navigation.

The Contact Form lets visitors submit queries and comments to the group. It has fields for the user's name, email, and message for organised communication. The form's intuitive design makes it professional and easy to use. This functionality simplifies questions, feedback, and communication for the company. Interactive online apps that aid and engage people need it. The Contact Details and Embedded Map improve usability. With intuitive icons, the Contact Details Section displays the organization's phone number, email address, and physical location. Users may quickly find information in this part without several pages(Meloni & Kyrnin 2018)

. As per (Robbins 2018) the Embedded Map uses Google Maps to show the organization's location, making it easy to find and visit. The online application can give virtual communication and in-person interaction or physical access to the company.

# Flask-based backend with AI model integration

Flask application is developed with the following features



Figure 4 Flask code for back end

## Home Page

The **Home Page** is where users enter the web application. It's implemented using Flask's render\_template method, which dynamically fetches home.html. This page was designed to:

* **Introduce the System**: To help visitors understand the platform, the AI-Powered Threat Detection System's purpose and functionality are discussed.
* **Engage Users**: An engaging hero section with a call-to-action button attracts users.
* **Navigation**: The logical navigation links provided allow users to effortlessly visit other pages like About, Contact, and the Admin Panel. The Home Page improves web security with professionalism and friendliness(Lakshmanarao, Babu & Bala Krishna 2021).

## About Page

Project objectives are described on the **About Page**. Like the Home Page, Flask's render\_template loads about.html. Key page features:

* **Mission Statement**: The project's main purpose is to detect web vulnerabilities and improve cloud security using AI and machine learning.
* **Team Details**: Information on team members, their roles, and project contributions is appealing, fostering greater user trust.

Users can understand the project's purpose and team's talents on the About Page, which stresses worth and vision.

## Contact Page

The **Contact Page** is key to user interaction. Users can ask questions, give feedback, and get help. A simple form in Contact.html has fields for:

* **Name**: To identify the user.
* **Email**: For follow-up communication.
* **Message**: Explain their query. Simply designed, user-friendly form. This page illustrates the app's audience engagement and problem-solving focus.

## Prediction Page (Admin)

The **Prediction Page** predicts web vulnerabilities with functionality. This page allows users to input text data for analysis and is implemented in Flask's /admin route(Singh 2021)

.

**Core Functionalities:**

1. **Text Input**:
   * Users can submit descriptions or records of potential vulnerabilities in the provided text area.
   * A POST request sends form inputs to the backend(Grinberg 2018)

.

1. **Preprocessing Pipeline**:
   * Using the preprocess\_text function, the input text is transformed.
   * Clean and convert input to meet the machine learning model.
2. **Vectorization**:
   * Vectorizer.transform converts cleaned text to integers using a pre-trained vectorizer.
   * This ensures machine learning model input compatibility.
3. **Prediction**:
   * Preprocessed and vectorised input is utilised to forecast vulnerability type using machine learning (best\_model.pkl)(Singh 2021).
   * For accuracy and robustness, the Random Forest model predicts input.
4. **Result Display**:
   * Using label encoder.inverse\_transform form, we forecast vulnerability kind.
   * The result appears dynamically in a <div> element on the Admin Page.
5. **Error Handling**:
   * In order to ensure system functionality and user friendliness, the backend is designed to handle edge cases like invalid or empty inputs(Grinberg 2018)
   * .

## Deployment and Real-World Use

The entire software is installed on AWS using Amazon S3 for datasets and models. Each page's functionality is tested in real life.

* **Home Page**: Displays navigation links and project info appropriately.
* **About Page**: Checks project mission and crew.
* **Contact Page**: Verifies data and form submission.
* **Prediction Page**: Checks machine learning model prediction accuracy and Flask backend integration.

Flask-based web application identifies web vulnerabilities thoroughly. Each page is designed for user comfort and machine learning prediction accuracy. The project improved web security using AI-driven insights thanks to user-friendly interfaces and a robust backend. Enhancing model capabilities and the deployment process for scalability and performance are conceivable.

# AWS Backup Settings: Automating Backups for Disaster Recovery

As per (None Zein Samira et al. 2024) Cloud-based AWS Backup automates and manages backups for AWS resources. This feature was designed to ensure data protection, streamline the disaster recovery process, and meet compliance requirements. Protect critical data efficiently and securely with AWS Backup disaster recovery.

## Purpose of AWS Backup

The production, storage, and management of Amazon EC2, RDS, S3, DynamoDB, EBS volume backups are automated by AWS Backup. These backups are crucial for disaster recovery scenarios when systems or data may need to be restored from failures, cyberattacks, or accidental deletions(Amazon Web Services (AWS), n.d.-a).

## Key Features of AWS Backup:

1. **Centralized Backup Management**: AWS Backup unifies the backup operations for several services under a single interface. Users can create rules to set backup schedules, retention periods, and recovery goals for AWS resources.
2. **Backup Automation**: Setting backup parameters automates the process. This ensures protected sensitive data.
3. **Retention Policies**: AWS Backup users can establish retention policies for backups stored. To meet organisational or legal requirements and optimise storage costs, these policies store backups for a set time.
4. **Cross-Region and Cross-Account Backup**: For enhanced disaster recovery, AWS Backup allows cross-region and cross-account backup. It safeguards sensitive data from regional outages and ensures backups are stored geographically.
5. **Encryption and Security**: AWS Backup ensures data security by encrypting backups both in transport and at rest. AWS Identity and Access Management integration limits who can create, update, and remove backups.
6. **Monitoring and Reporting**: AWS Backup tracks backups in real time with CloudWatch and CloudTrail. Compliance and audits benefit from transparency(Amazon Web Services (AWS), n.d.-a).

## Benefits for Disaster Recovery

1. **Minimized Downtime**: Backups automate speedy and efficient disaster recovery, reducing downtime.
2. **Compliance**: Automated backups and predetermined retention periods help companies meet data protection rules.
3. **Cost Efficiency**: By leveraging automated and policy-driven backups, AWS Backup reduces administrative overhead and optimises storage use, saving money.

AWS Backup is an essential tool for modern cloud systems since it simplifies the disaster recovery process by ensuring consistent, secure, and automatic protection of key resources.

A screenshot of a computer

Description automatically generated

Figure 5 Backup Plans

The "Backup plans" option in AWS Backup shows the disaster protection and data backup management plan. It contains a single backup plan called "DisasterRecoveryPlan" designed to provide backup needs, such as schedules, retention rules, and lifecycle policies. Without a last runtime, this backup plan seems configured but unexecuted. A recent configuration modification occurred on December 24, 2024, 22:15:47 (UTC+05:30) in the plan.

As per Amazon Web Services (AWS) (n.d.-d ) AWS Backup users can create and manage backup plans to automate and simplify the process of safeguarding vital data across AWS services. DisasterRecoveryPlan ensures data may be efficiently stored in case of system failures, accidental deletions, or other unforeseen catastrophes. This plan allows for customisation of backup frequency, retention length, and storage lifecycle transitions.

The interface supports "Create on-demand backup" and "Create backup plan" for policy-driven backups. Users can dynamically adjust data protection for fast and long-term disaster recovery using these features. By leveraging such capabilities, organisations may comply with data protection laws, reduce downtime, and secure digital assets.

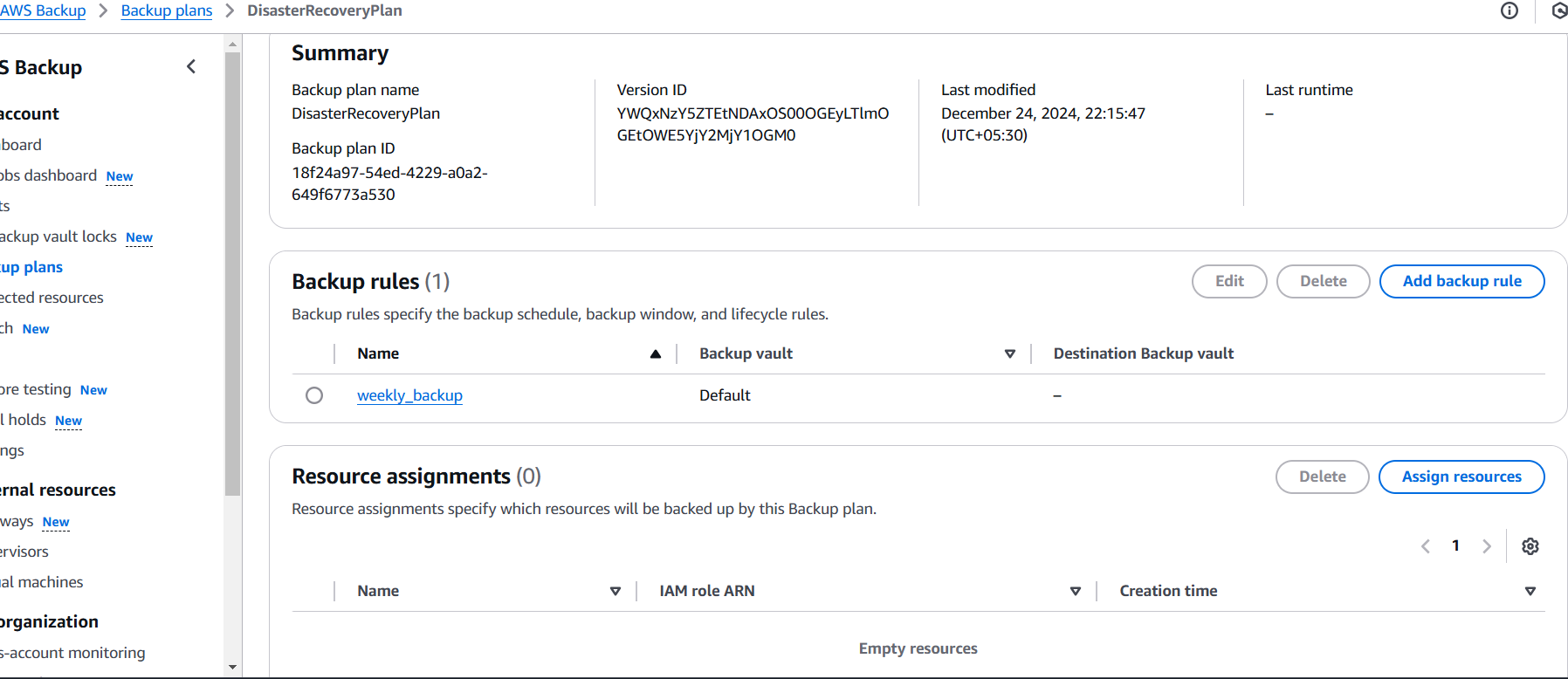


Figure 6 Disaster Recovery Plan

DisasterRecoveryPlan is configured to automate data protection. The Summary includes the backup plan's Backup Plan ID and Version ID for traceability and version management. The plan was updated December 24, 2024, at 22:15:47 (UTC+05:30). Last Runtime is empty, indicating that this backup plan has not yet executed backups.

The backup rule "weekly\_backup" controls backup retention and transition schedule, window, and lifecycle policies. Backups are stored in the default backup vault, a secure repository, according to this regulation. Cross-region backup vaults can be configured(Amazon Web Services (AWS), n.d.-c).

The Resource Assignments column is empty, indicating this backup plan contains no AWS resources. Assigning resources like EC2 instances, RDS databases, and EBS volumes to the backup plan applies the configured rules. These resources can be protected under this plan using the Assign Resources interface option.

It prepares for a solid disaster recovery plan. AWS Backup's adaptability and scalability are highlighted in the DisasterRecoveryPlan, allowing companies to automate data protection, assure compliance, and limit downtime in the case of system failures or data loss. Assigning resources and starting runtime processes is necessary for activating and leveraging this plan.

# Justification of selected methods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component | Selected Method | Reason for Selection | Alternative Methods | Reason for Not Choosing Alternatives |
| Frontend Development | HTML, CSS, JavaScript | - Standard technologies for creating responsive, dynamic, and user-friendly web interfaces. - Easy integration with Flask templates. - Highly customizable. | React.js, Angular.js | - Require additional setup and complexity. - Not necessary for the small-scale, specific nature of this project. |
|  |  |  | Bootstrap or TailwindCSS | - Limited design control without additional custom CSS. - Focused more on styling rather than overall functionality. |
| Backend Framework | Flask | - Lightweight and flexible framework. - Seamless integration with Python-based ML models. - Simplifies routing and template rendering. | Django | - Overly complex for this use case. - Higher learning curve and unnecessary features for the current project. |
| Machine Learning Model | Random Forest | - High interpretability and robustness. - Effective for small and medium datasets. - Handles imbalanced data and text inputs efficiently. | Neural Networks or SVM | - Neural Networks require significant computational resources. - SVM may underperform with larger datasets or text-based inputs. |
| Backup and Storage | AWS Backup and Amazon S3 | - Scalable and secure storage for backup automation. - Supports cross-region and cross-account backups. - Simplifies disaster recovery with automation. | Google Cloud Backup, Microsoft Azure | - Both alternatives are viable but were not chosen due to AWS's superior ecosystem integration with the current project. |
| Deployment Platform | AWS (Amazon Web Services) | - Reliable and widely adopted cloud infrastructure. - Provides tools for seamless integration with backup and storage services. - Cost-effective PAYG model. | Google Cloud, Microsoft Azure | - Google Cloud and Azure offer similar services but are less familiar to the project team. - AWS offers better integration with S3 and Flask. |
| Backup Management | AWS Backup with "DisasterRecoveryPlan" | - Centralized management of backup schedules and lifecycle rules. - Automates disaster recovery tasks for minimal downtime. | Manual Backup Scheduling | - Time-intensive and prone to errors. - Lacks automation, scalability, and cross-region capabilities offered by AWS Backup. |
| Testing Methods | Functional and Backend Testing | - Ensures that the application operates as expected. - Validates the end-to-end functionality, including integration of ML models and data pipelines. | Automated Testing | - Requires significant time and resources to set up. - Not feasible for a project of this scale with defined timelines. |

# Challenges and Mitigations

|  |  |
| --- | --- |
| Challenges | Mitigation Strategies |
| Designing a Responsive Contact Page | Used HTML, CSS, and JavaScript to create an intuitive and responsive contact page. Ensured user-friendly elements such as form validation for user inputs, visual enhancements for icons, and integration of Google Maps for location reference. |
| Ensuring Flask Backend Integration with the Contact Page | Established seamless backend integration using Flask's render\_template and routing features to dynamically render the contact.html file. Tested input submission processes to ensure data was routed correctly for backend processing. |
| Prediction Workflow Implementation for Admin Page | Designed a robust pipeline to preprocess text inputs using NLTK and ensured compatibility with the TF-IDF vectorizer. Leveraged the Random Forest model for accurate predictions, addressing edge cases such as invalid or empty inputs. |
| AWS Backup Configuration for Disaster Recovery | Configured the AWS Backup feature with a "DisasterRecoveryPlan" to automate backup schedules and lifecycle management. Ensured resources such as EC2 instances and EBS volumes were assigned to the backup plan for comprehensive data protection. |
| Handling Scalability and Performance of Cloud Deployment | Leveraged Amazon S3 for scalable storage and Elastic IPs for stable application access. Configured AWS Backup for automated disaster recovery to ensure minimal downtime and enhanced system resilience under high load or failures. |
| Data Security and Compliance in Backup Management | Implemented encryption for backups at rest and in transit. Configured IAM policies to restrict access and ensured compliance with data protection regulations through cross-region and cross-account backups. |
| Complexity in Preprocessing Pipeline for Text Inputs | Simplified text preprocessing with key steps such as stopword removal, lemmatization, and special character removal using NLTK. Validated processed data to maintain compatibility with the machine learning pipeline and improve prediction accuracy. |
| Testing Backup Execution and Plan Effectiveness | Tested the backup plan's functionality with on-demand backups. Regularly monitored backup execution logs using AWS CloudWatch and CloudTrail to validate that retention policies and disaster recovery configurations worked as intended. |
| Ensuring the Prediction Page’s Real-Time Accuracy | Integrated the Random Forest model (best\_model.pkl) with Flask, ensuring accurate real-time predictions. Added dynamic result displays using Flask templates and implemented robust error handling for edge cases like malformed inputs. |
| Cost Management for AWS Backup and Elastic IPs | Monitored the usage of Elastic IPs and backup vaults to avoid unnecessary costs. Configured alerts for idle resources and optimized AWS configurations to ensure a cost-effective cloud infrastructure deployment. |

# Conclusion

The project shows a Flask-based web application with machine learning to forecast web vulnerabilities. The project improves web security and usability through careful design and technology selection. The frontend, designed with HTML, CSS, and JavaScript, ensures an intuitive and responsive interface for users. Flask was chosen as the backend framework for its lightweight architecture and easy interaction with the Python-based Random Forest model, which balances accuracy, robustness, and performance in handling text-based inputs.

Amazon S3 and AWS Backup enabled secure, scalable storage and disaster recovery, improving system stability and data protection. Functional testing ensuring the machine learning model's flawless backend interface and all pages working as planned proved the application's real-world usefulness.

Modern web technologies, scalable cloud services, and machine learning create a web vulnerability detection system at the project. Increased model capabilities, optimised backup, and automated testing could boost efficiency. This application prepares for advanced web security solutions and is scalable and flexible.

# References

Amazon Web Services (AWS) (n.d.-a) *What is AWS Backup?,* AWS Documentation. Available at: https://docs.aws.amazon.com/aws-backup/latest/devguide/whatisbackup.html [Accessed 20 December 2024].

Amazon Web Services (AWS) (n.d.-b) *Security considerations in AWS Backup*, AWS Documentation. Available at: https://docs.aws.amazon.com/aws-backup/latest/devguide/security-considerations.html [Accessed 20 December 2024].

Amazon Web Services (AWS) (n.d.-c) *Creating a backup plan*, AWS Documentation. Available at: https://docs.aws.amazon.com/aws-backup/latest/devguide/creating-a-backup-plan.html [Accessed 20 December 2024].

Amazon Web Services (AWS) (n.d.-d ) *Plan for disaster recovery (DR),* AWS Well-Architected Framework. Available at: https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/plan-for-disaster-recovery-dr.html [Accessed 20 December 2024].

Grinberg, M. (2018) *Flask Web Development*, O'Reilly Media, Inc.

Lakshmanarao, A., Babu, M.R. & Bala Krishna, M.M. (2021, September 1) *'Malicious URL Detection using NLP, Machine Learning and FLASK'*, IEEE Xplore. DOI: https://doi.org/10.1109/ICSES52305.2021.9633889.

Meloni, J.C. & Kyrnin, J. (2018) HTML, CSS, and JavaScript All in One, Sams Publishing.

Zein Samira, N., Wondaferew Weldegeorgise, Y., Osundare, O.S., Oke, H. & Coelis, R. (2024) *'Disaster recovery framework for ensuring SME business continuity on cloud platforms'*, Computer Science & IT Research Journal, 5(10), pp. 2244–2262. DOI: https://doi.org/10.51594/csitrj.v5i10.1620.

Robbins, J. (2018) *Learning Web Design*, O'Reilly Media, Inc.

Singh, P. (2021) Deploy Machine Learning Models to Production, Apress. DOI : https://doi.org/10.1007/978-1-4842-6546-8