Faculty of Science and Information Technology
Department of Software Engineering
Project Report on

Vulnerability Detection in Source Code Using AI

Supervised by:
Mr. Md. Shohel Arman
Assistant Professor
Department of Software Engineering
Daffodil International University

Submitted by:
Ashraful Alim
ID: 192-35-2828
Department of Software Engineering,
Daffodil International University

This Project Report has been submitted to the Department of Software Engineering (SWE) at Daffodil International University in the fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering

APPROVAL

This project titled on "**Vulnerability Detection in Source Code Using AI**", submitted by **Ashraful Alim (ID: 192-35-2828)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

Chairman

Dr. Imran Mahmud
Professor & Head
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 1

Md Shohel Arman
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 2

Md. Rajib Mia
Lecturer (Senior Scale)
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

External Examiner

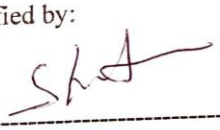
Md Habibur Rahman
Associate Professor
Department of Computer Science and Engineering
Islamic University, Bangladesh

DECLARATION

I state that I completed this project with the supervision of Mr. Md. Shohel Arman, Assistant Professor in the Department of Software Engineering at Daffodil International University. Furthermore, I declare that this project represents my own original work for the Bachelor of Science degree in Software Engineering, and that neither the entire project nor any portion of it has been submitted for any other degree at this or any other institution.

Ashraful Alim
ID: 192-35-2828
Department of Software Engineering,
Daffodil International University

Certified by:



Mr. Md. Shohel Arman
Assistant Professor,
Department of Software Engineering,
Daffodil International University

ACKNOWLEDGEMENT

My project, "Vulnerability Detection in Source Code Using AI", represents a significant step in translating abstract ideas into real-world applications. The entire process has been a deeply enriching experience.

Firstly, I am immensely grateful for the strength and well-being by Almighty Allah, that allowed me to complete this project successfully. I believe that divine guidance from Almighty played a crucial role in reaching my academic achievement.

I also want to acknowledge Daffodil International University, and specifically the Department of Software Engineering, for creating a supportive educational environment to achieve my academic goal. I am particularly thankful to Professor Dr. Imran Mahmud, the Department Head, for his consistent encouragement and guidance during my academic journey.

A special thanks goes to my project supervisor, Mr. Md. Shohel Arman, Assistant Professor in the Department of Software Engineering, Daffodil International University. His insightful advice during the project development, constructive criticism of my work, and unwavering support were essential to the success of this research and the project.

Finally, I extend my acknowledgement and gratitude to all my teachers in my university, fellow students of my department, and everyone who offered support throughout my academic journey with the university. Your encouragement, guidance and contributions have been invaluable for my learning. Thank you for being an essential part of my academic journey and experience.

ABSTRACT

This project delves into the critical need for automated source code security checks in today's modern software development process. The software development process in the industry is growing more complex day by day, the challenge of finding security weaknesses inside source code is also becoming hard. My work, "Vulnerability Detection in Source Code Using AI," offers an AI-driven method to detect and guide users for potential security problems in application source code early in the development phase.

The main objective is to build an intelligent system that learns from deep learning and then uses its learning to analyse application source code and pinpoint where is the vulnerabilities. I am focusing on issues like vulnerable code injection points, insecure object handling by the code, improperly handling user data, and the other misuse of programming functions. The system learns by studying labeled dataset and from a teacher model of both secure and vulnerable code, allowing it to recognize patterns linked to insecure coding inside of an application.

The overall process of my application includes preparing different code samples, extracting key features and labeling, training deep learning models from specialized teacher model, and assessing the system's accuracy and performance. The end goal of my project after successful completion is, to lessen the need for manual application source code review for application developers, educate the users with security best practice and boost the overall security inside application development process.

While the current system successfully identifies several common vulnerabilities, there's room for improvement. Future work includes expanding language support and integrating the system into real-time development tools.

Table of Contents

CHAPTER 1 -

PROJECT PLANNING & INITITATION

1.1 Project Overview

The application, titled "Vulnerability Detection in Source Code Using AI," is developed and engineered to automatically detect and understand security weaknesses within software source code. This system will emphasize easy to use user interaction, scan and detect vulnerable code and pinpoint specific vulnerabilities, including injection vulnerabilities like SQL injection, XSS injection, authentication bugs and insecure functions that are prone to cyberattacks. In this application, developers and security analyst can upload their code file or directly input the code and then receive an instant report outlining the detected issues, and then follow suggested remediation steps from the application. Furthermore, the system aids in managing the analyzed code by keeping track of previous security findings, and produces both detailed and summary reports. The main goal of this project initiative is to help developers to improve software security and minimize the manual workload related with the code review. This will also foster the development of more secure applications.

1.2 Preliminary Analysis & Project Scope Definition

1.2.1 Preliminary Analysis

This project effort arises from a growing and urgent need within the software development industry: the necessity to safeguard applications from developing cyber dangers associated with the application and source code. In today's digital world, where programming and software development is the backbone for our everyday critical services such as banking application, digital healthcare service, digital business, and e-commerce platforms, applications have become prime targets for cyber attackers and malicious actors for unethical profit gain. Conventional security good practices, like manually review application source code, tends to be not only labor-intensive but also time consuming task with application security expertise, high cost for companies, and struggle to keep up with the fast-paced nature of our modern software development as well as newly dicovered security threats.

The primary and also an important challenge of this project is to tackle and manage the inefficiency and scalability issues associated with manual source code review and vulnerability detection for application source code. Developers require tools that are not only accurate and pinpoint to the security issue but also seamlessly integrates into their software development workflow, offering guidance for security issue remediation and also actionable feedback related to security. This proposed system with intelligent AI engine specialized in coding related task aims to leverage the pattern-recognition capabilities of Artificial Intelligence, specifically deep learning, to automate the analysis of source code for vulnerabilities. By detecting complex vulnerabilities like injection of malicious code, malicious JavaScript injection and bypassing authentication logics early in the development lifecycle, the project directly contributes to building more secure, reliable, and trustworthy digital systems for the software industry.

1.2.2 Project Objectives

This project's main objective is to design, create and deploy an intelligent system powered by AI that automatically finds critical security flaws in source code. The main goals for this project are:

- To Create a Custom AI Model: To create and train a deep learning model by fine tuning existing effective model specialized in coding so that it can examine and understand source code and then categorize security flaws.
- To Create an Web Application: The objective is to design and develop an easily navigable and easy to understand web application so that developers and security analyst can submit source code for security analysis and view the analysis output with security issue findings.
- To Provide Actionable Reports: For every scan, the application will produce multiple format report like detailed reports, summary reports that will identify the type and name of a vulnerability, severity of the found issue, the precise line number in the source code where is the security issue and offer a practical and effective remediation guidance.
- To Visualize Security Trends: Place graphical dashboards that summarize scan results of the source code, allowing users to see the vulnerability distribution by severity and monitor the code quality over the time of the different scan.
- To Promote Secure Coding Practices: To serve this project as an educational tool and preventative tool that will help the developers and security analysts to learn about common security pitfalls inside their application source code that can have serious impact and adopt a security-first mindset to always prioritize security during every phase of the development.

1.2.3 Project Scope Definition

The scope for this project is established as below to guarantee that the project can be completed within allotted time and resources:

In-Scope:

- Code Submission: Allows developers and security analysts to submit application source code file for different programming languages like .py, .java, or .js or paste source code straight into the text field.
- Core AI Scanning Engine: The core component of the AI scanning engine will be a fine tuned model (deep learning model) that has been taught from a teacher model that specializes in coding task, to identify a specific collection of common application vulnerabilities.
- Scan Dashboard: A dashboard that displays the history of all vulnerability scans performed on the source code application.
- Detailed Results Page: A dedicated page containing a summary, graphical charts, and a list of vulnerabilities found along with guidance on how to fix those issues.
- Report Generation: A specific page that contains a summary of findings as well as suggestions for resolving them, graphical charts for visualization, and a comprehensive list of vulnerabilities discovered along with recommendations for fixing them.

Out-of-Scope:

- Integration with Real-time IDE: Plugins that can be integrated with IDEs like Eclipse or VS code will not be able to perform real-time security scanning of application source code using this system.
- Automated Code Remediation: The system will suggest necessary security fixes with secure code example for identified vulnerabilities but will not automatically rewrite or modify the user's code and fix the vulnerability.
- Scanning of Compiled Binaries: Only application source code will be analyzed by the application.

- Team-Based Access Control: The system will not provide role-based user permissions or team management feature inside the system. Individual user accounts for developers and security analysts are the main focus of this system.
- Support for All Programming Languages: The model will focus on a limited number of programming language that is used commonly in development. This will ensure high accuracy of vulnerability findings within a given source code.

1.3 Market Feasibility Analysis

1.3.1 Demand and Need in the Market

The application security testing (AST) tool industry is thriving and growing day by day. Businesses of all sizes are now placing a high premium on cybersecurity due to rising frequency of cyber attacks in recent years and expense of data breaches for every major attacks. Static application security testing (SAST) is a dedicated domain of cybersecurity that focuses specially in source code security and examines source code of applications for vulnerabilities that can impact the application and business. The SAST market is a perfect fit for my vulnerability detection project. The following factors fuel the demand:

- Adoption of DevSecOps: In the software industry, there is a great need for automated solutions that developers wants to utilize directly inside their devsecops phases to detect weakness and improve the code quality. As a result of this types of need, the industry-wide shift towards incorporating security into the software development phases is increasing day by day.
- High Costs of Security Breaches: Whenver a cyber incident happens that generally causes a great amount of financial and reputational harm to an organization. This type of cost that is caused by malicious incident is significantly greater than the actual expense of defensive software and hardware deployment and utilization for an organization.

1.3.2 Target Audience

There are two main user groups for which the system is intended:

The primary audience consists of software developers and security analysts that are responsible of developing and delivering secure and effective application for an organization. So that they can scan and find vulnerabilities inside their source code during the development phase and then address and fix those vulnerabilities without being a security domain expert with years of experience in the domain. They actually need an effective tool that is quick, gives accurate output for weakness findings, and offers not only developer-friendly but also understandable feedback with the report.
Another audience for this project is the security analysts. Security analysts are primarily responsible for the security testing of an organization. These security analysts need tools for auditing codebases, complying with international cybersecurity compliance and monitoring vulnerability trends across the projects are necessary for them. Weakness and issue detection accuracy and thorough reporting of those issues are important part of the process for the security team to improve overall security.

1.3.3 Competitive Landscape

The mature application security testing market is home to a number of well-known open-source project and commercial solution:

- Commercial Leaders: Snyk, Veracode, Checkmarx, and GitHub Advanced Security (CodeQL) are market dominanator for enterprise grade application security tools. Along with the integration into enterprise CI/CD pipelines, these tools provide comprehensive solutions that often include SAST, DAST (Dynamic Analysis), and SCA (Software Composition Analysis).
- Open-Source Tools: The developer community makes extensive use of well-known open source tools including SonarQube, Bandit (for Python), and ESLint (for JavaScript security plugins.

1.3.4 Niche and Competitive Advantage

The field of application security is competitve around the world. Though there is competition in the business market, my project can establish a viable niche in the application security domain market. This can be achieved because of the benefits below:

- Emphasis on a Custom AI Model: The primary distinction of my project from many others tool is that, other tools mostly rely on rule-based coding as its vulnerability detection method. This project is unique because of its custom deep learning model that is fine tuned from renowned code specific models. This enables the detection of complex or distinct vulnerability or weakness patterns in application source code that rule-based engines could miss sometime.
- Simplicity and Accessibility: In a business setting, properly deploying and utilizing enterprise technology may be costly and complicated. This project aims to provide a web-based interface that is easy to use, highly accessible, and has a clear design for developers. For small teams, software developers, or educational settings, this makes it ideal.
- Cost-Effectiveness: By focusing on a core feature set and leveraging open-source technologies, the system can operate at a very low cost, making it a tempting alternative for software industry customers.

In conclusion, the project has a growing market. It meets a glaring market need for easily accessible, intelligent, and developer-friendly security technologies.

1.4 Technical Feasibility Analysis

1.4.1 Technology Stack

The use of modern, well-maintained, primarily open-source technology during the development of the project makes it technically feasible.

- Frontend (Web Application): The web application’s user interface will be created as dynamic and responsive page by utilizing the Tailwind CSS. I'll integrate libraries like Chart.js for data visualization and Axios for API interaction in the dashboard or generated report.
- Server and API Backend: Flask will be used to create a Python-based backend. Python programming language is a solid option because of its smooth interaction and community support with AI/ML libraries. File uploads, queue scanning tasks, user authentication (using JWT), and database interactions will all be handled efficiently by this backend technology for the project.
- AI Model Development: To fine tune a student model from a specialized teacher model, TensorFlow and PyTorch libraries will be used to develop the project's core in Python.

Recurrent neural networks (RNNs), such LSTMs or Transformers, which are especially good at interpreting sequential data like source code, will probably be a part of the model design.

- Database: A relational database like PostgreSQL will house user information, scan metadata, and results for source code vulnerability scan by users.
- Infrastructure for Deployment: The system will be built for both on premise and cloud deployment. For the hosting of Frontend of the application, two platforms can be used: Netlify and Vercel. Using deployment platform and tools like Heroku, AWS Beanstalk or Google Cloud Run, the AI model and application backend can be configured as containerized application. For container deployment, docker technology will be used.

1.4.2 Key Technical Challenges & Solutions

- Challenge 1: Training AI models with high accuracy is the main technological challenge for this project. Fine tuning an existing model requires a sizable, high-quality, labeled dataset with both vulnerable and non-vulnerable code samples in multiple programming languages. I'll use publicly available datasets, like NIST's SARD (Software Assurance Reference Dataset), or carefully chosen datasets from GitHub. Hyperparameter tuning, masked dataset, feature engineering (including code tokenization), and data pretreatment will take a significant amount of time.
    - Solution: Public datasets like the SARD (Software Assurance Reference Dataset) from NIST or curated datasets from GitHub will be used. Data preprocessing, feature engineering (such as tokenizing code), and hyperparameter optimizing are all going to require up a significant amount of time.
- Challenge 2: Because AI model inference takes a long time to finish, the second issue involves long scans that could result in request timeouts or server obstructions.
    - Solution: A task queue system, like Redis in conjunction with Celery/BullMQ, will be put into place to solve this. A task will be queued when a user starts a scan. To keep the web server responsive, a different worker process in the server will take care of the assignment, it will finish the scan, and update the database when it's finished.
- Challenge 3: Resource Needs: Deep learning model training requires a lot of resources.
    - Solution: Free services like Google Colab or Kaggle that offer GPU access will be used for preliminary model training and experimentation for this project.

Conclusion: Even though the project has technical difficulties, it is possible to develop and complete the project for accurate and effective outcome. The suggested technology stack is well-established in the software industry, and the main technological obstacles have clearly defined solutions. The success of the project will ultimately depend on how well the primary AI model is developed by fine tuning dataset and trained on the context.

1.5 Financial Feasibility Analysis

1.5.1 Cost Breakdown

As a university project, the primary goal for this project is not making commercial profit but to deliver a functional prototype within a minimal budget. The financial analysis, therefore, focuses on the direct costs of development and deployment.

Financial Summary Table:

Item	Estimated Cost	Justification
Developer Labor	$0	Academic project, developer time is the primary investment.
Software & IDEs	$0	Utilizing open-source and free software.
AI Libraries	$0	Utilizing open-source and free frameworks.
Domain Name	$15 / year	Standard cost for a .com or .dev domain.
Web Hosting	$0	Leveraging free tiers from Vercel, Heroku, or similar.
AI Training Compute	$0	Utilizing free GPU resources from Kaggle.
Total Estimated Cost	~$15 / year	The only tangible cost is the optional domain name.

Conclusion: The project is very financially feasible, and the direct cost of developing and running this system for demonstration purposes is very low due to the strategic use of open-source technologies and the cost-free tiers of contemporary cloud hosting platforms. The project's true value is in its educational output and the working prototype, not in the money it makes.

1.6 Target User Profile and Tentative Elicitation Process

1.6.1 Target User Profiles

Two main user profiles have been identified to attempt to ensure that the system is developed with a clear focus on what users need.

Profile 1: The developer is the main target user.

Archetype: A mid-level software engineer who develops application features together with a small to medium-sized team.
Goals & Motivations: The main goals they have are to meet project deadlines and producing high-caliber, useful code. They might not have thorough, specialized security knowledge, but they regardless want to create safe programs. They like fast, non-intrusive technology that provide fast, practical information that can be used without any research.

Pain Points:

- The deployments might get delayed by the difficulty of traditional security reviews.
- Jargon-filled, complicated security reports that are hard to fully understand by users and take actions to actually remediate them.
- It is expensive and time consuming to transite from their software development environment (IDE) to a different and challenging cyber security analysis platform.
- System Expectations: The user expects an easy web interface that can immediately copy or upload a source code file, show the exact line of code that has vulnerability, and deliver scan

results with low false positive in just a few minutes. The solution they prefer must be practical for their development scenario as well as easy to implement in their environment.

Profile 2: The team lead, secondary target user, or security analyst

Archetype: A senior developer or security an expert who manages an application's source code security and code quality posture.
Goals & Motivations: Codebase audits, risk identification with a list of priority rating, assurances instructions for cybersecurity compliance, and remediation progress reporting are among their goals. The goal of the organization is to minimize the cybersecurity risk and vulnerability findings related to the company's software assets.
Pain Points:

- Analyzing hundreds of thousands of lines of code by manually is not adaptable.
- establishing an accurate overview of the computer security condition of many projects is challenging.
- Clear and detailed data along with visual graphics are necessary part of report. Because when delivering security vulnerabilities report to management; raw technical logs simply are inadequate and hard to understand by the management team.
- System Expectations: The user believes that the system will offer reliable and accurate identification of source code vulnerabilities along with clear levels of indication for better understanding. For their purpose of comparing scans and tracking trends, they need a historical dashboard. During the documentation and communication process, they need the capacity to download an official PDF report to deliver to the stakeholders.

1.6.2 Tentative Requirements Elicitation Process

The user target profiles' system requirements would be gathered verified via the following techniques.

Indirect Elicitation (Preliminary Investigation):

- Competitive Analysis: Analyze modern enterprise tools like Snyk, Veracode etc. and free to use tools like SonarQube, Bandit for their technological innovations inside the application. This makes it easier in developing an understanding of desired features, such reporting formats, supported languages, and the kinds of vulnerabilities found. These goods' feedback from users are reviewed to identify consistent compliments and complaints..
- Literature Review: To learn about the latest techniques for identifying and typical developer security issues, read research papers on AI in software security as well as posts from security blogs (such as the OWASP Foundation).

Direct Elicitation: Establishing the Behavior of a User):

- Surveys: A straightforward and focused on online survey would be developed to evaluate the needs of users. inquiries for engineers might be focused on integration of workflows and efficiency, and while those for analysts might be focused on analytics and reporting (e.g., "Rank the usefulness of PDF reports vs. a user-friendly dashboard"). As an example, "how much time is suitable for scanning to complete?".
- Interviews: They respond to interviews that are semi- structured by assuming the roles of people who have specific profiles. In additionally addressing further questions for requirement

explanation, this qualitative strategy allows an even more thorough examination of their preferred features and issues.

- Prototyping & Walkthroughs: Making use of a tool such as Figma, low-fidelity models mockups and wireframes of the web-based tool would be made. The prototypes would subsequently be shown to the simulations included users in a "cognitive walkthrough," which means that they would have to decide to do things like "start a new scan" or "find the details of a vulnerability." The feedback they gave would directly influence the UI/UX design of the web application, which ensures that the final product is easy to use and user-friendly.

1.7 System Requirements

The backend of the application system consists of multiple interconnected services for effective analysis rather than a single, monolithic application:

- Web Server/API: The frontend of application's web server or API must handle all incoming HTTP requests from users. In this way, it will maintain user identity records, handle requests to start scan, offer perviously performed scan result, and interact with database to modify necessary records.
- AI Worker: Assessing the AI model inferences on the provided application source code is probably the most technologically demanding and important task. This task will be completed by the separately created AI worker. After completing the scan, it uploads the findings back to the application database and also listens for next job from the queue. Long scanning won't cause the web server to stop or create request timeouts because of this different mechanism.
- Database Service: The database service will be used to forward all persistent data, including source code scan results, user login and their password, and a comprehensive analysis of source code vulnerabilities.
- Job Queue/Cache Service: A Redis-like job queue or cache service acts as a message broker between the web server and the AI worker. This mechanism will be implemented inside Each time a scan request is made, an operation will be added to the job queue. It also serves as an immediate cache for session data.

Hardware Requirements:

Processor	1.5GHz or upper
RAM	4 GB recommended
Disk Space	4 GB of available hard disk

1.8 Project Schedule

A project timeline outlines the features that need to be developed, the resources required, and the deadlines for project completion. Given the limited time available for system development, effective scheduling is essential to ensure the project is completed on time. This timeline also serves as a means of communication regarding the tasks that must be accomplished within a short timeframe.

1.8.1 Gantt Chart

Activities	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13
Research													
Requirement Analysis													
Planning													
Designing													
Development													
Testing													
Assessment													
Documentation													

Table 1.1: Gantt Chart

1.8.2 Release Plan or Milestone

The release plan or milestones according to the Gantt Charts is given below:

Activities	**Duration in week**	**Total week**
Research	W1, W2, W3	3
Requirement Analysis	W3, W4	2
Planning	W4, W5	2
Designing	W5, W6	2
Development	W6, W7, W8, W9, W10	4
Testing	W10, W11	2
Assessment	W11, W12	2
Documentation	W12, W13	2
Release	W13	1

CHAPTER 2 -
DESIGN AND IMPLEMENTATION

2.1 Functional Requirements

The specifications for functionality describe the fundamental characteristics that a system must contain. These are the fundamental tasks that the system needs perform effectively in order accomplish what it wants. For my "Vulnerability Detection in Source Code Using AI" project, the following sets of functional requirements have been developed:

- Code Submission: Submit the source code documents or paste code immediately into the the editor for analysis.
- Start Scan: Start the the procedure of scanning for the source code whose services has pasted or uploaded.
- Vulnerability Detection: Use AI models specialized in coding to examine the application source code and find security flaws.
- View Scan Results: Establish detailed records containing the various categories of vulnerabilities, their level of severity objectives, and the code segments that are impacted.
- Download Report: Allow the users to download different format of scan results for documenting or sharing with stakeholder.
- Visualization through Graphics: Create charts and graphs that show the necessary information of vulnerabilities for a given application source code.
- Result History: preserve a history of earlier scans for comparison and monitoring as time passes.
- User Dashboard: Deliver a user dashboard for monitoring data, scans, and access to resources

2.2 Non-functional Requirements

Non-functional requirements for my project defines the quality attributes of the system and describe how the system performs under various different conditions. The security, and usability of the system for users are guaranteed by these important factors. The non-functional requirements for my project titled as “Vulnerability Detection in Source Code Using AI” include:

- Security: Ensure secure authentication, data handling, and safe code storage to protect sensitive information
- Performance: The system should deliver scan results within a reasonable response time, even under heavy load
- Reliability: The system must operate without failure during code scanning and result generation
- Scalability: Support multiple concurrent users and large code files without performance degradation
- Usability: Provide an easy to understand and user-friendly interface for both technical and non-technical users
- Maintainability: The system should be easy to update and extend for future improvements and features
- Privacy: Ensure that users’ code and personal information are kept confidential and not shared without consent

2.3 Development Model

In order to develop my project "Vulnerability Detection in Source Code Using AI", the Agile Software Development Life Cycle (SDLC) methodology is used during the development. In terms of the software development, Agile means an effective development model that proposes an incremental and iterative process of approach that encourages adaptability, flexibility, and ongoing feedback with the different development phases of a software project development. This agile approach is particularly well-suited for challenging and dynamic project development that incorporate both user-facing application features and a specialized deep learning model development.

My project was divided into several short iterations, each lasting one to two weeks of process. Every iteration with the development phase focused on delivering incremental improvements of activities for successful completion of the project, whether in the deep learning model fine tuning for coding tasks, the user interface of the application, or the overall system integration with other necessary components. This allowed to respond quickly from complex to moderate challenges that raised during development of the project, incorporate feedback for better user interaction experience, and refine both the core AI functionality and the web application features.

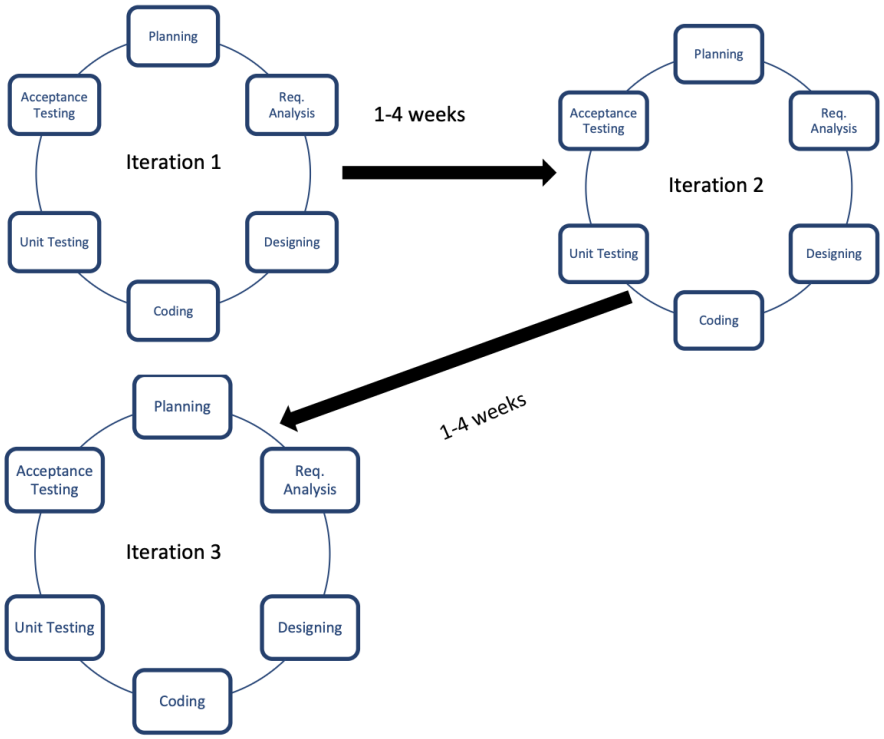


Figure 3.1: Agile SDLC Model

This Agile-based approach enabled parallel development of the AI model and the web interface, ensuring rapid delivery, continuous improvement, and alignment with user expectations. It also facilitated smooth integration of the model's output into the application, providing users with real-time, interpretable, and actionable vulnerability reports.

2.4 Use Case Diagram

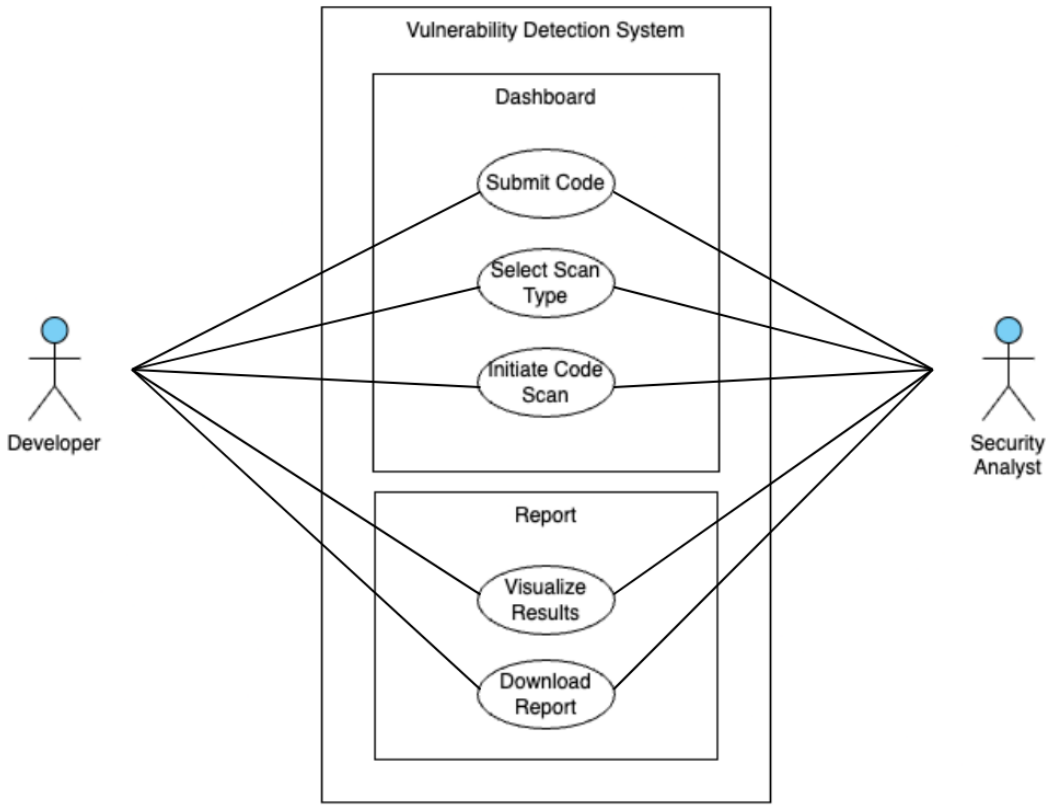


Figure 3.2: Use Case Diagram

2.5 Use Case Description

Submit Code

Field	Details
Use Case Name	Submit Code
Actor	Developer, Security Analyst
Description	Allows users to submit source code for vulnerability analysis either by uploading a file or pasting code directly into the system.
Preconditions	User is authenticated and logged into the system
Main Flow	1. User navigates to the code submission interface 2. System displays submission options (upload file or paste code) 3. User selects submission method: - Option A (Upload): User clicks "Upload File" and selects source code file from local system - Option B (Paste): User clicks "Paste Code" and enters code in the text area 4. System stores the code submission in the database 5. System displays confirmation message with submission details
Alternative Flows	Empty code submission - System prompts user to provide valid code content
Postconditions	• Code is successfully stored in the system • User can proceed to initiate scan
Business Rules	Code content must not be empty

Initiate Code Scan

Field	Details
Use Case Name	Initiate Code Scan
Actor	Developer, Security Analyst
Description	Enables users to start the AI-powered vulnerability detection process on previously submitted code.
Preconditions	• Code has been successfully submitted • AI engine is available and operational
Main Flow	1. User selects a code submission from their submission history 2. System displays submission details and scan options 3. User clicks "Start Scan" button 4. System sends code to AI engine for analysis 5. System updates scan status to "Processing" 6. AI engine completes vulnerability analysis 7. System updates scan status to "Completed"
Alternative Flows	• AI engine unavailable - System displays error message and suggests retry later • Multiple concurrent scans - System queues additional scans

Postconditions	• Scan record is created with unique scan ID • Vulnerability analysis results are stored • User can proceed to view results
Business Rules	Maximum concurrent scans per user: 1

View Scan Results

Field	Details
Use Case Name	View Scan Results
Actor	Developer, Security Analyst
Description	Allows users to view detailed vulnerability analysis results from completed scans.
Preconditions	At least one scan has been completed successfully
Main Flow	1. User navigates to scan results section 2. System displays list of completed scans with basic information 3. User selects a specific scan to view details 4. System retrieves detailed scan results from database
Alternative Flows	No scan results available
Postconditions	User has viewed vulnerability analysis results

Download Reports

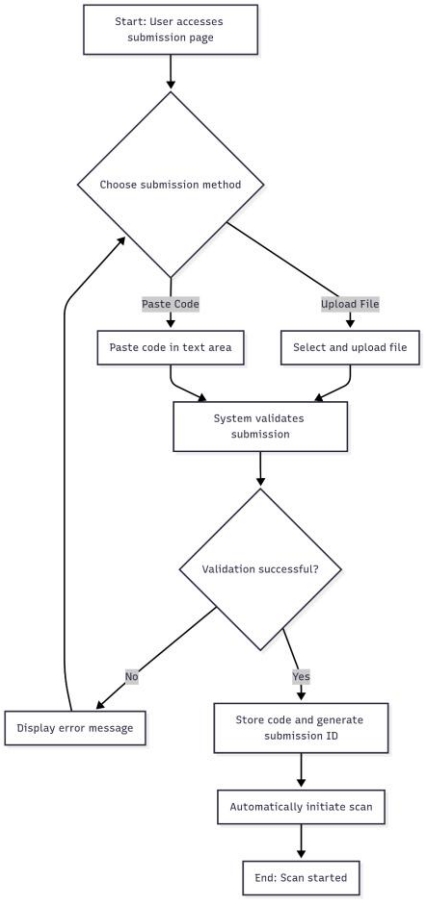
Field	Details
Use Case Name	Download Reports
Actor	Developer, Security Analyst
Description	Enables users to download previously generated vulnerability reports to their local system.
Preconditions	At least one report has been generated
Main Flow	1. User navigates to reports section 2. User selects report to download 3. System prepares download link and initiates file transfer 4. User's browser downloads the report file
Alternative Flows	Report file not found, Download interrupted
Postconditions	Report file is successfully downloaded to user's local system

Visualize Scan Results

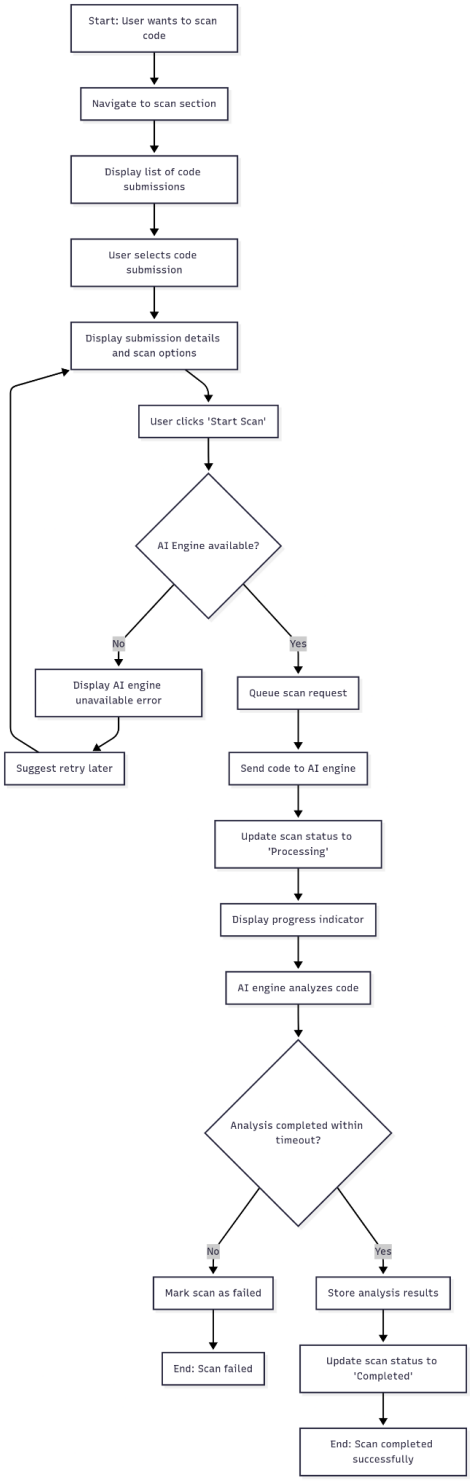
Field	Details
Use Case Name	Visualize Scan Results
Actor	Developer, Security Analyst
Description	Provides graphical representations and charts of vulnerability scan results for better understanding and analysis.
Preconditions	• Scan results are available with sufficient data for visualization • Visualization engine is operational
Main Flow	1. User accesses scan results visualization section 2. System analyzes available data and determines suitable visualizations 3. System displays visualization dashboard with chart
Alternative Flows	• Insufficient data for visualization • Visualization rendering failure
Postconditions	Visual representations of scan data are displayed

2.6 Activity Diagram

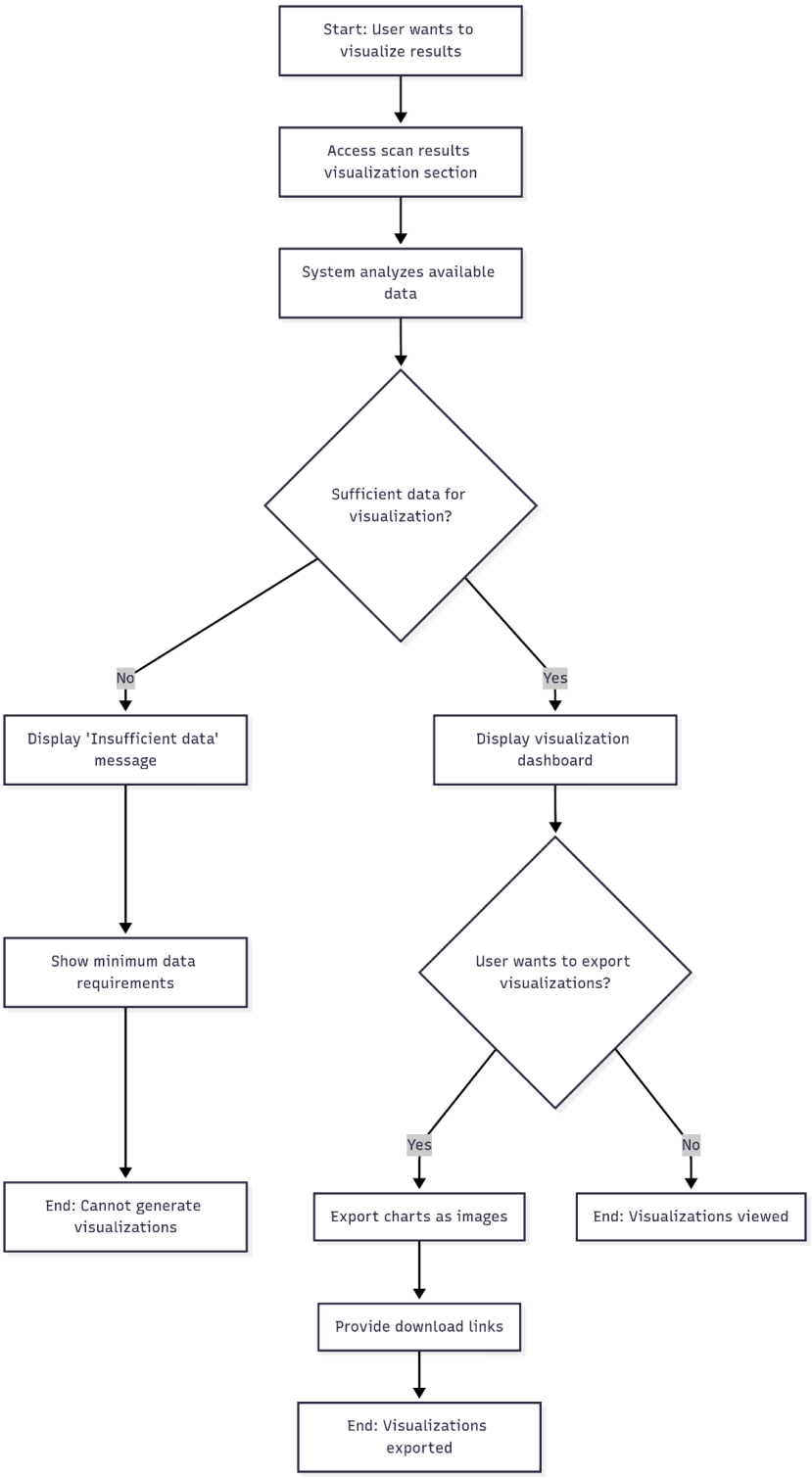
Submit Code



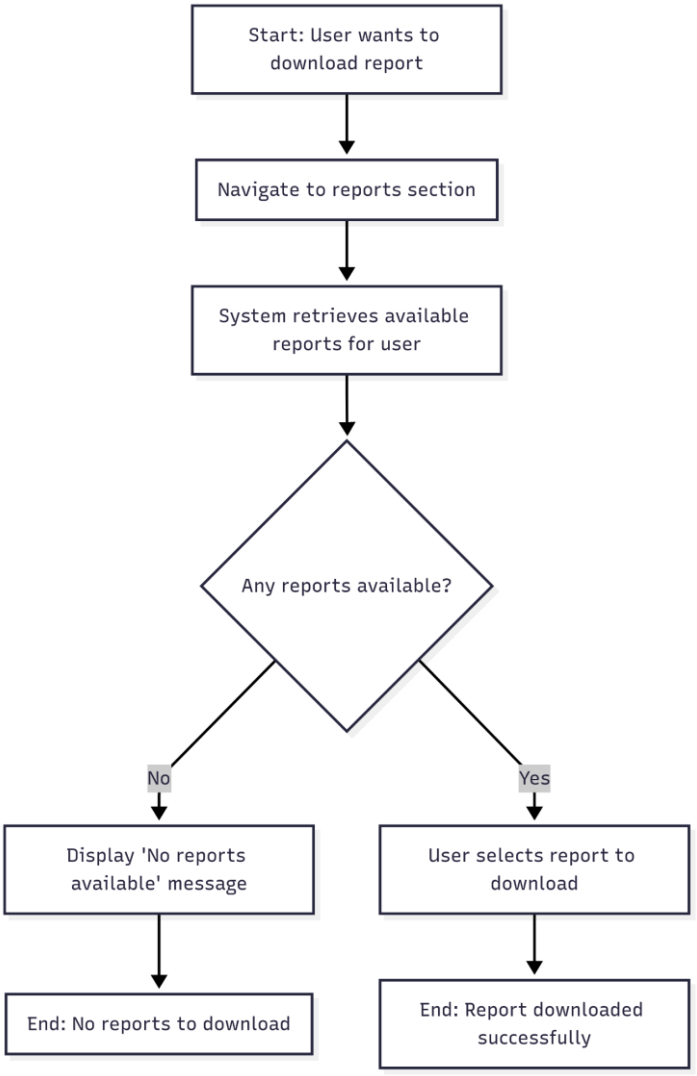
Scan Code



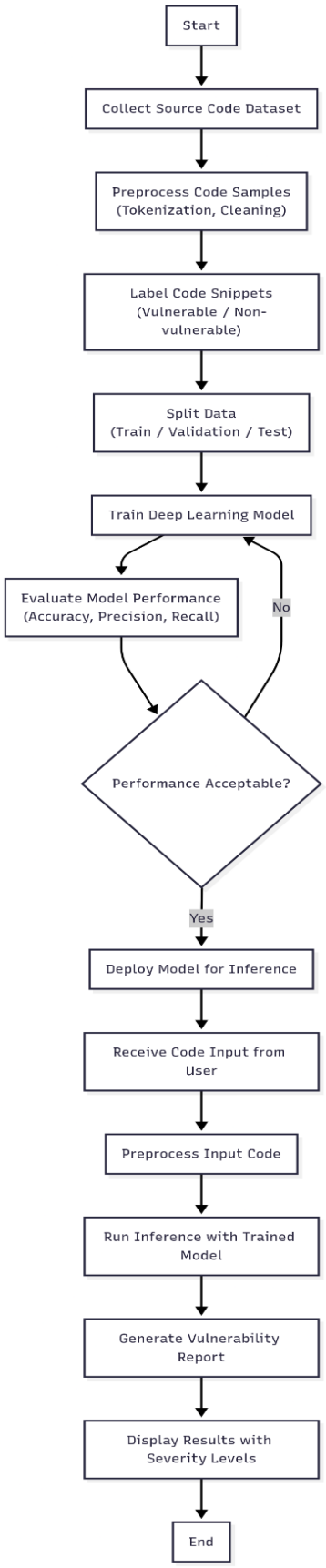
Visualize Results



Download Report

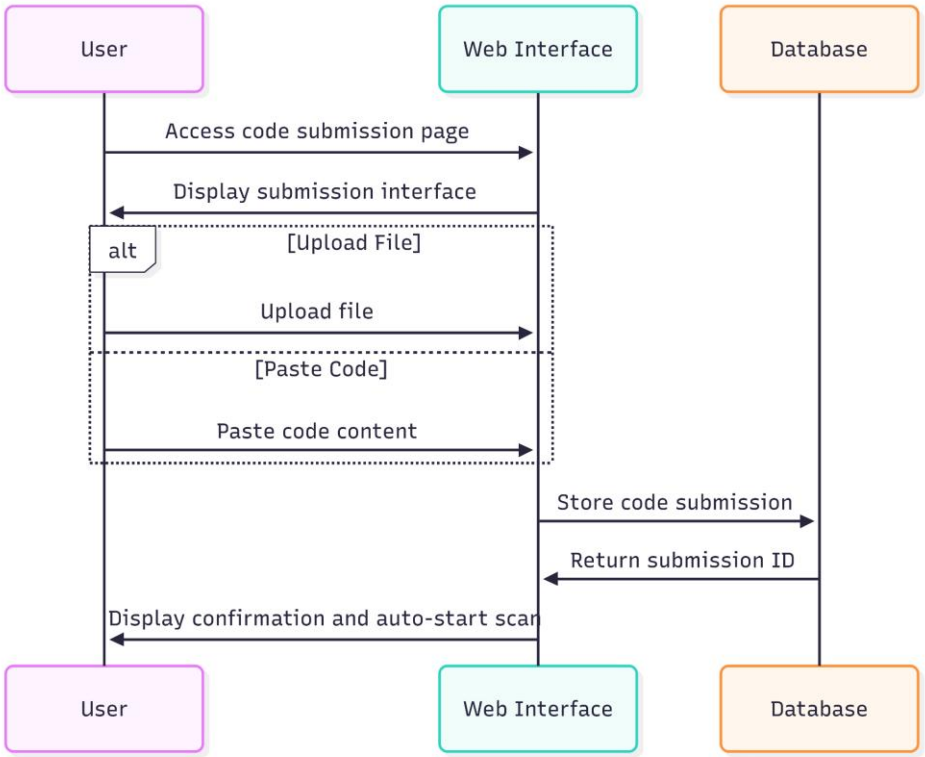


AI Model:

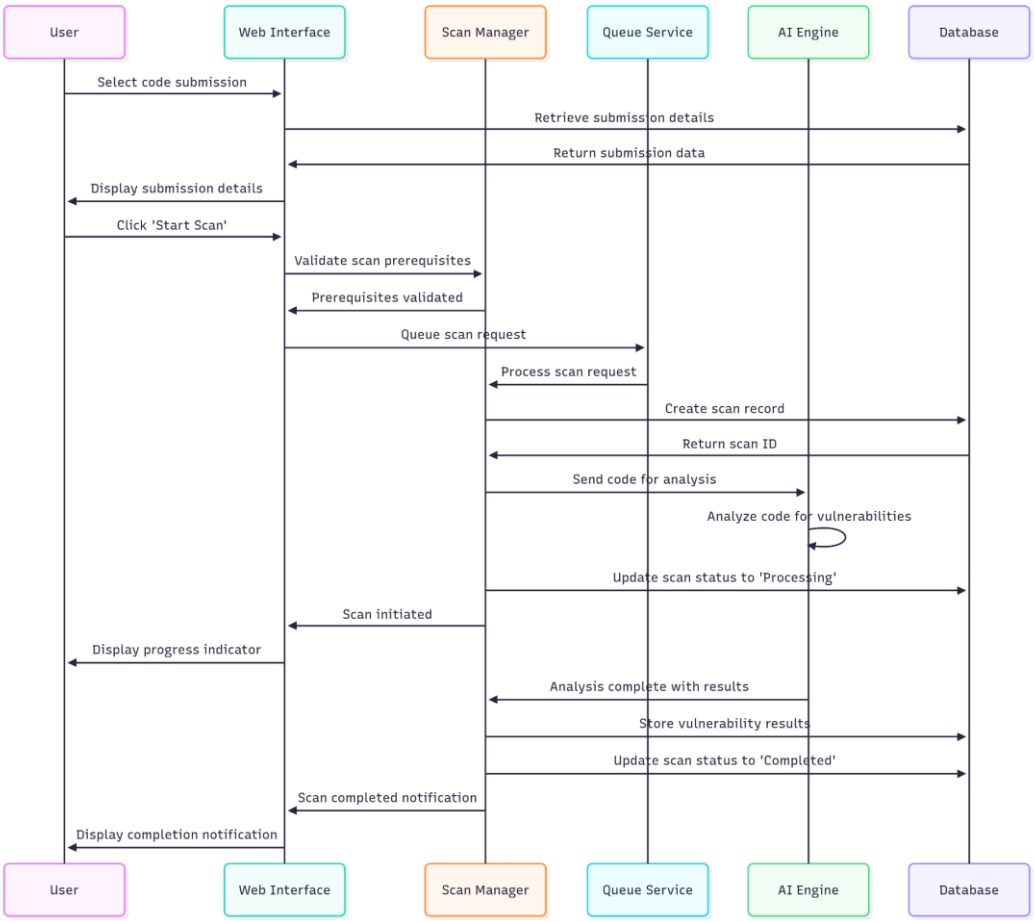


2.7 Sequence Diagram

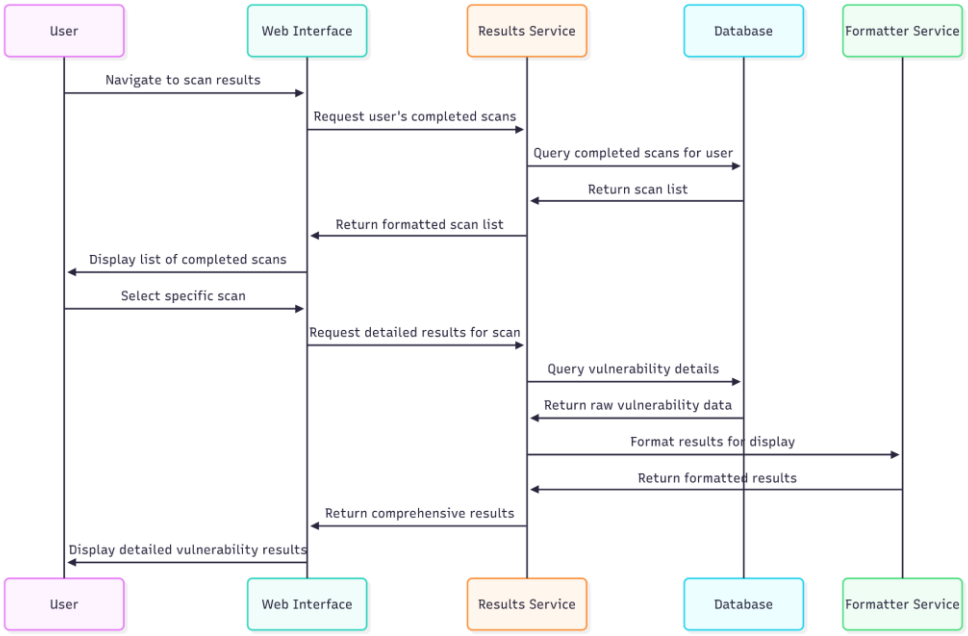
Submit Code:



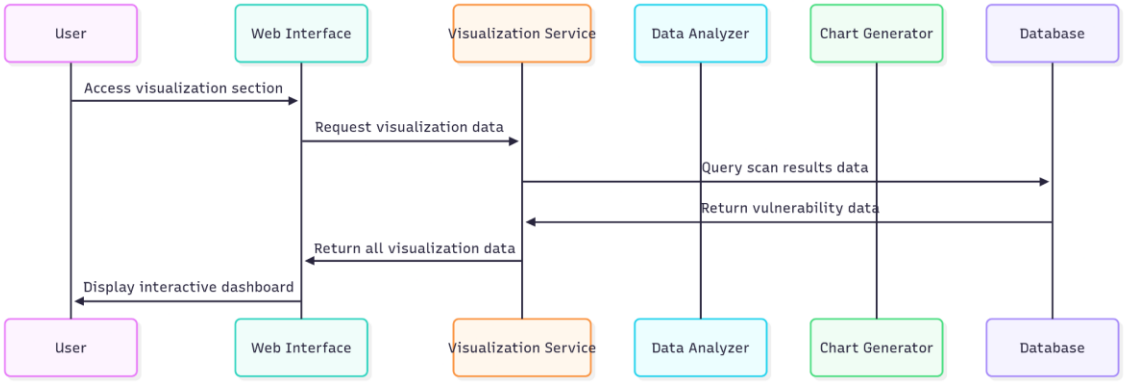
Initiate Code Scan:



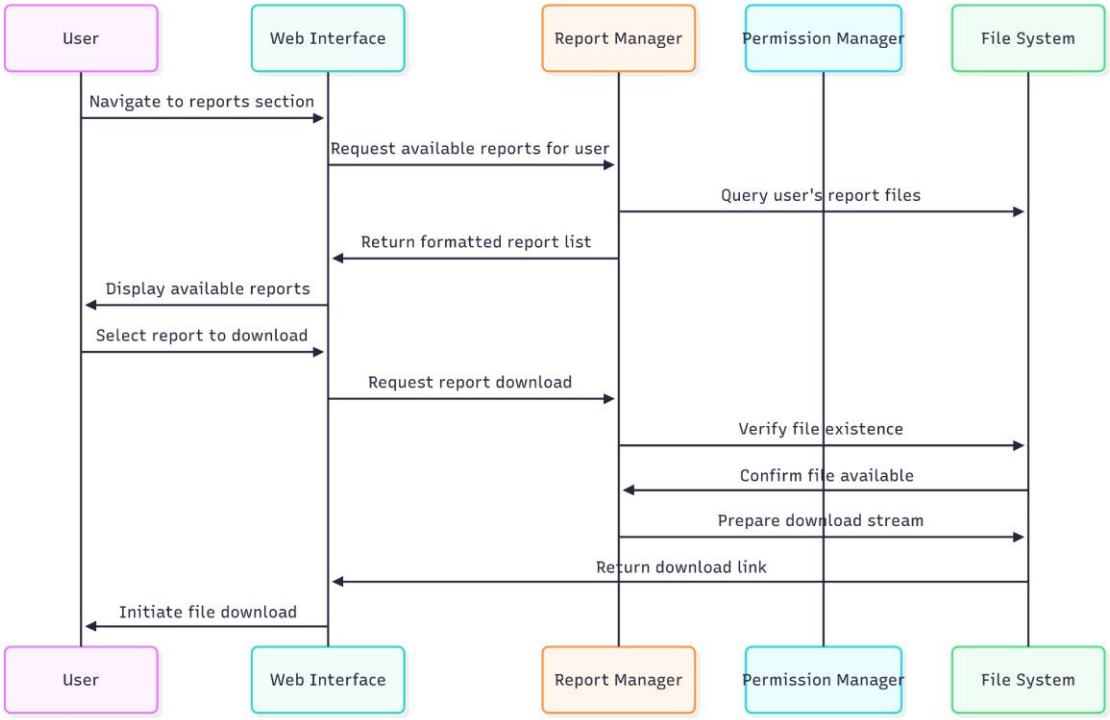
View Scan Result:



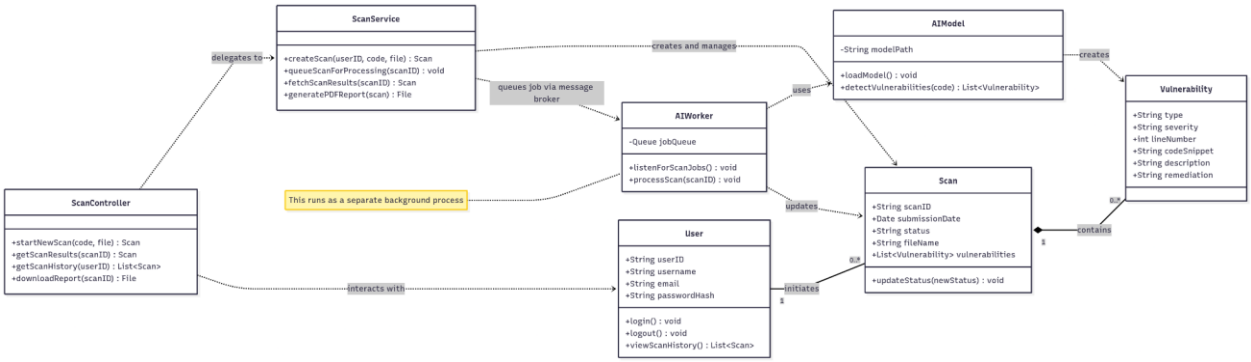
Visualize Scan Result



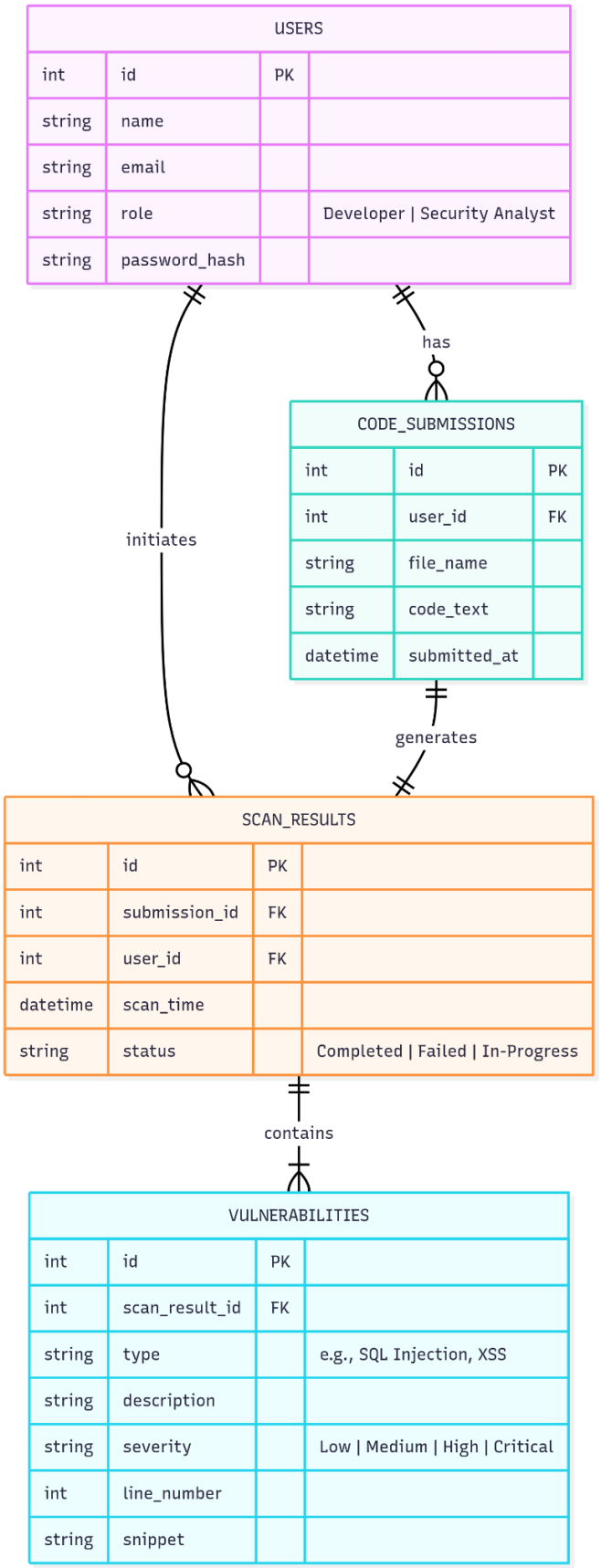
Download Reports



2.8 Class Diagram



2.9 Entity Relationship Diagram



CHAPTER 3 -
SOFTWARE TESTING

3.1 Feature Testing

In the software development lifecycle, feature testing is an essential part, particularly for systems that prioritize user engagement for most of the activities and to ensure critical security of a product. I used feature testing in this project to confirm the effectiveness and accuracy of results using the web-based user interface and the AI-powered vulnerability detection system that is specialized for coding related tasks and instructions. To verify the correctness of the AI model findings, ease of using the application, dependability for developers, and source code security findings, each essential element was tested separately and thoroughly.

When a new features are added or existing ones are changed inside the application, the significance of testing is further highlighted to ensure proper operation. This procedure of testing ensures that the system remains robust, effective for vulnerability detection, and provides a seamless and user friendly experience for all types of users, including the main target group such as security analysts and developers.

3.1.1 Features to be tested

Feature	Priority	Description
Code Submission	1	Users can upload or paste source code to be scanned
Scan Initiation	1	Initiates the vulnerability scan using the trained deep learning model
View Scan Results	2	Displays detailed vulnerability report with code references and severity levels
Graphical Visualization	2	Generates visual charts to represent scan history and vulnerability trends
Download Report	3	Allows users to download scan reports for documentation

3.2 Test Strategies

3.2.1 Test approach

To ensure system quality, two different types of tests are used black box testing and white box testing.

- Black Box Testing: Black box testing is a testing where the internal functions are not known to the tester and focus is only on the output. It is also called functional and behavioral testing.

- White Box Testing: White Box Testing is called clear bo x testing because internal structure of software's design, and coding are tested to verify input-output flow and improve design, usability, and security.

3.2.2 Test Schedule

Test Phase	Time
Create Test Plan	1 Week
Unit Testing	During Development
Component Testing	During Development
User Interface Testing	1 Week
Performance Testing	2 Week
Accessibility Test	2 Week

3.3 Test Cases

3.3.1 Test Case for Code Submission

Test Case 01	
Case Name:	Verifying Code Submission via Paste and Upload
Designed By:	Ashraful Alim
Design Date:	04.06.25
Executed By:	Ashraful Alim
Execute Date:	04.06.25
Short Description:	This case tests the functionality for submitting source code by pasting it into a textarea and by uploading a file.
Precondition:	User has navigated to the new scan page: https://ai-vulnerability-app.vercel.app/dashboard/scan

Test Table

Steps	Action	Action Result	Expected System Response	Pass/Fail
1	Paste a valid snippet of Python code into the text area.	Text appears in the text area.	The system accepts the pasted code. The "Start Scan" button becomes enabled.	Pass
2	With pasted code still in the text area, attempt to upload a .py file.	Error Message or UI prompt.	A message appears, stating "You can either paste code or upload a file, not both. Please clear one to proceed."	Pass

3	Clear the text area and attempt to upload an invalid file type (e.g., image.png).	Error Message.	An error toast/message appears: "Invalid file type. Please upload a source code file (e.g., .py, .java, .js)."	Pass
4	Clear the text area and upload a valid source code file (e.g., main.java).	File name appears on the UI.	The system accepts the file. The UI displays "main.java" as the selected file, and the "Start Scan" button is enabled.	Pass

3.3.2 Test Case for Scan Initiation

Test Case 02	
Case Name:	Verifying the Scan Initiation Process
Designed By:	Ashraful Alim
Design Date:	04.06.25
Executed By:	Ashraful Alim
Execute Date:	04.06.25
Short Description:	This case tests the system's behavior when a user initiates a code vulnerability scan.

Test Table

Steps	Action	Action Result	Expected System Response	Pass/Fail
1	Click the "Start Scan" button without pasting or uploading any code.	Button is disabled or shows an error.	The "Start Scan" button is disabled. If enabled, clicking it shows a message: "Please paste code or upload a file to scan."	Pass
2	Upload a valid .js file and select "JavaScript" from a language dropdown (if available).	File is accepted.	The system correctly identifies the file and prepares for scanning.	Pass
3	Click the "Start Scan" button.	UI shows a loading state.	The button changes to a "Scanning..." state with a spinner. The user is prevented from submitting another scan. A progress bar may appear.	Pass
4	Wait for the scan to complete.	Redirect to the results page.	After the AI model finishes processing, the user is automatically redirected to the results page for the new scan	Pass

5	Clear the text area and upload a valid source code file (e.g., main.java).	File name appears on the UI.	The system accepts the file. The UI displays "main.java" as the selected file, and the "Start Scan" button is enabled.	Pass

3.3.3 Test Case for View Scan Results

Test Case 03	
Case Name:	Verifying the Scan Results Display
Designed By:	Ashraful Alim
Design Date:	07.06.25
Executed By:	Ashraful Alim
Execute Date:	07.06.25
Short Description:	This case tests the detailed view of a completed scan, ensuring all information is accurate and well-presented.

Test Table

Steps	Action	Action Result	Expected System Response	Pass/Fail
1	Observe the summary section at the top of the page.	Summary data is displayed.	The page correctly displays the Scan ID, file name (user_code.py), timestamp, and a summary (e.g., "5 Vulnerabilities Found: 1 High, 2 Medium, 2 Low").	Pass
2	Scroll down to the list of detected vulnerabilities.	A list of vulnerabilities is shown.	Each item in the list clearly shows the vulnerability type (e.g., "SQL Injection"), severity level (e.g., "High"), and the line number where it was found.	Pass
3	Click on a specific vulnerability from the list (e.g., a "Cross-Site Scripting" vulnerability).	The item expands to show details.	An expanded view appears, showing the vulnerable code snippet (with the specific line highlighted), a detailed description of the risk, and a recommended solution/remediation advice.	Pass
4	Navigate to the results page of a scan that found zero vulnerabilities.	"No Vulnerabilities" message.	The page displays a clear message like "Scan Complete. No vulnerabilities were found in your code." The summary shows "0 Vulnerabilities Found."	Pass

3.3.4 Test Case for Graphical Visualization

Test Case 04	
Case Name:	Verifying Graphical Representation of Scan Results
Designed By:	Ashraful Alim
Design Date:	07.06.25
Executed By:	Ashraful Alim
Execute Date:	07.06.25
Short Description:	This case tests the graphical charts used to visualize the distribution of vulnerabilities.

Test Table

Steps	Action	Action Result	Expected System Response	Pass/Fail
1	Locate the graphical visualization component on the results page.	A chart is displayed.	The chart is rendered correctly without any visual glitches.	Pass
2	Compare the chart segments with the text summary.	Chart data matches text data.	The chart accurately reflects the summary (e.g., if the text says 2 High, 3 Medium, 1 Low, the chart segments are proportionally sized and labeled accordingly).	Pass
3	Hover the mouse cursor over the "High" severity segment of the chart.	A tooltip appears.	A tooltip pops up displaying the exact count for that severity, e.g., "High: 2".	Pass
4	Hover over the other segments ("Medium" and "Low").	Tooltips appear for each segment.	Each segment's tooltip correctly displays its corresponding vulnerability count.	Pass

3.3.5 Test Case for Download Report

Test Case 05	
Case Name:	Verifying the Report Download Functionality
Designed By:	Ashraful Alim
Design Date:	08.06.25
Executed By:	Ashraful Alim
Execute Date:	08.06.25

Short Description:	This case tests the ability to download the report

Test Table

Steps	Action	Action Result	Expected System Response	Pass/Fail
1	Click the "Download Report" button on the results page.	A file download is initiated in the browser.	The browser prompts to save a file named something like Vulnerability_Report_scan_id_789.pdf.	Pass
2	Save the file and open it using a PDF reader.	The PDF file opens without errors.	The file is not corrupted and is readable.	Pass
3	Verify the content of the PDF.	The PDF lists all vulnerabilities.	The report contains a detailed breakdown of each vulnerability, including its type, severity, description, line number, code snippet, and remediation advice, mirroring the information on the web page.	Pass

CHAPTER 4 -
DEPLOYMENT AND MAINTENANCE

4.1 Agile Methodology Adoption

The project is developed using an Agile SDLC methodology from the very beginning of the development phase. The development was not only focused on Agile but also with a particular focus on the Scrum framework. This effective management approach made it possible to provide feature additions incrementally during the development and offered frequent chances to adjust to user feedback to key ares to improve and with the technical difficulties they are facing.

- Product Backlog: Priority is set for this project based on various different aspects of the outcomes. A prioritized Product Backlog will contain all user stories related to the product, technical tasks that are need to be done, necessary bug fixes for the application, and requested features by the users. For all work to be done, product backlog record will be very reliable source of necessary information to focus on.
- Sprints: Sprint is another important concept from Agile methodology. It says that, the project will be divided into brief, time-boxed iterations known as Sprints. For this project, each sprint can last anywhere from one to two weeks of activity. High-priority items from the Product Backlog are selected at first to be finished in each Sprint. During the development, the goal of each sprint is to produce potentially shippable increment of software that has less bugs and more of the completed features that are important.
- Sprint Review: At the end of each Sprint, the completed work is demonstrated for the project. This type of periodic review system provides a regular feedback loop with the development team to ensure the project is developing in the right direction and going to meet the proper goal. For example, after an early Sprint, a basic UI design for uploading the code could be demonstrated to the users to gather necessary feedback from them about the usability and visibility of the web interface.

4.2 The Software Release Life Cycle (SRLC) in an Agile Context

The SRLC is a necessary concept for the development because this concept provides a structured pipeline during the development of a project through which each software increment passes to next phase or steps. This approach and management actually ensures the quality of work and stability of the software from the beginning of development to last stage of production.

Phase 1: Development (Alpha Stage)
This is the active development phase that occurs during each Sprint.

- Environment: All development occurs on local machines of the developers. Developers use Docker and Docker Compose to run a complete, containerized replica of the application stack (frontend, backend API, database, AI worker). This ensures the local environment mirrors the production setup as closely as possible, preventing "it works on my machine" issues and bugs related to this.
- Version Control: Git is mostly used for version control, with a central repository hosted on GitHub. All work is done on feature branches (e.g., feature/user-login). Code is only merged into the main development branch after it is complete and has been reviewed.

Phase 2: Testing & Integration (CI Pipeline)

This phase is automated using a Continuous Integration (CI) server.

- Trigger: As soon as a developer pushes code to a feature branch or creates a pull request to merge into the main branch, the CI pipeline is automatically triggered.
- Process:
    - Build: The CI server checks out the code and builds the application components (e.g., builds the Docker containers).
- Automated Testing: A suite of automated tests is run against the build. This includes:
    - Unit Tests: To verify individual functions and classes work as expected.
    - Integration Tests: To ensure that different components (e.g., API and database) interact correctly.
- Outcome: If the build or any test fails, the merge is blocked, and the developer is notified immediately. This ensures that the main branch always remains stable and functional.

Phase 3: Staging (Pre-Production / Beta Stage)

Once code is merged into the main branch and passes all CI checks, it is automatically deployed to a Staging environment.

- Purpose: The Staging server is a complete, publicly inaccessible replica of the Production environment. It uses the same infrastructure and is configured with its own separate database. Its purpose is for final validation before a public release.
- Activities:
    - User Acceptance Testing (UAT): The developer (or a peer) performs manual testing to confirm that the new features work as expected in a real-world environment. For example, they would manually upload a code file containing a known SQL injection flaw and verify that the AI model correctly identifies it.
    - Performance & Load Testing: Basic checks can be run to ensure the new changes have not negatively impacted system performance.

Phase 4: Production Release (Stable Release)

This is the process of making the new features available to end-users. This is managed via Continuous Deployment (CD).

- Trigger: The deployment to the Production server is a manually triggered action from the main branch, ensuring a human gatekeeper for the final release.
- Strategy: A Blue-Green Deployment strategy will be used to ensure zero downtime.
    - A new, identical "Green" environment is provisioned with the new version of the code.
    - The traffic router continues to send all live user traffic to the existing "Blue" environment.
    - Once the Green environment is fully deployed and tested, the router is switched to send all new traffic to the Green environment.
    - The old Blue environment is kept on standby and can be switched back to instantly in case of any issues.

Phase 5: Post-Release Maintenance & Monitoring

The lifecycle does not end at deployment. Continuous maintenance is a core part of the Agile process.

- Monitoring: The production environment is continuously monitored for:
    - Health & Uptime: Automated checks (e.g., UptimeRobot) ping the application endpoints to ensure the service is available.

    - Performance Metrics: Server CPU, RAM usage, and API response times are monitored to identify performance bottlenecks.
    - Error Tracking: An automated error tracking service (e.g., Sentry) captures and reports any runtime exceptions in the backend or frontend, providing immediate alerts for new bugs.
- Bug Fixing: When bugs are discovered in production, they are formally reported, prioritized, and added to the Product Backlog. They are then addressed in an upcoming Sprint, following the same SRLC process as any new feature.
- System Updates: Maintenance also includes regular updates to the underlying software stack (OS, runtimes, libraries) to apply security patches. This includes periodically re-training and deploying updated versions of the AI model as new vulnerability patterns are identified.

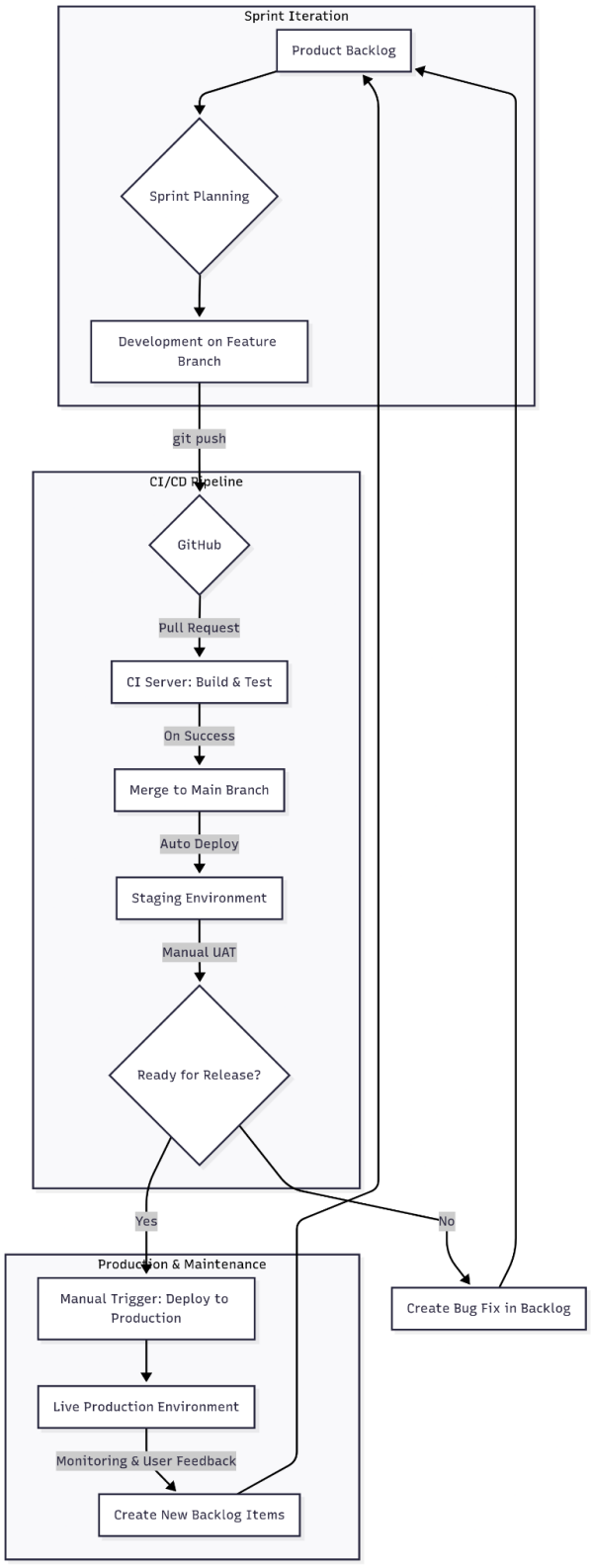
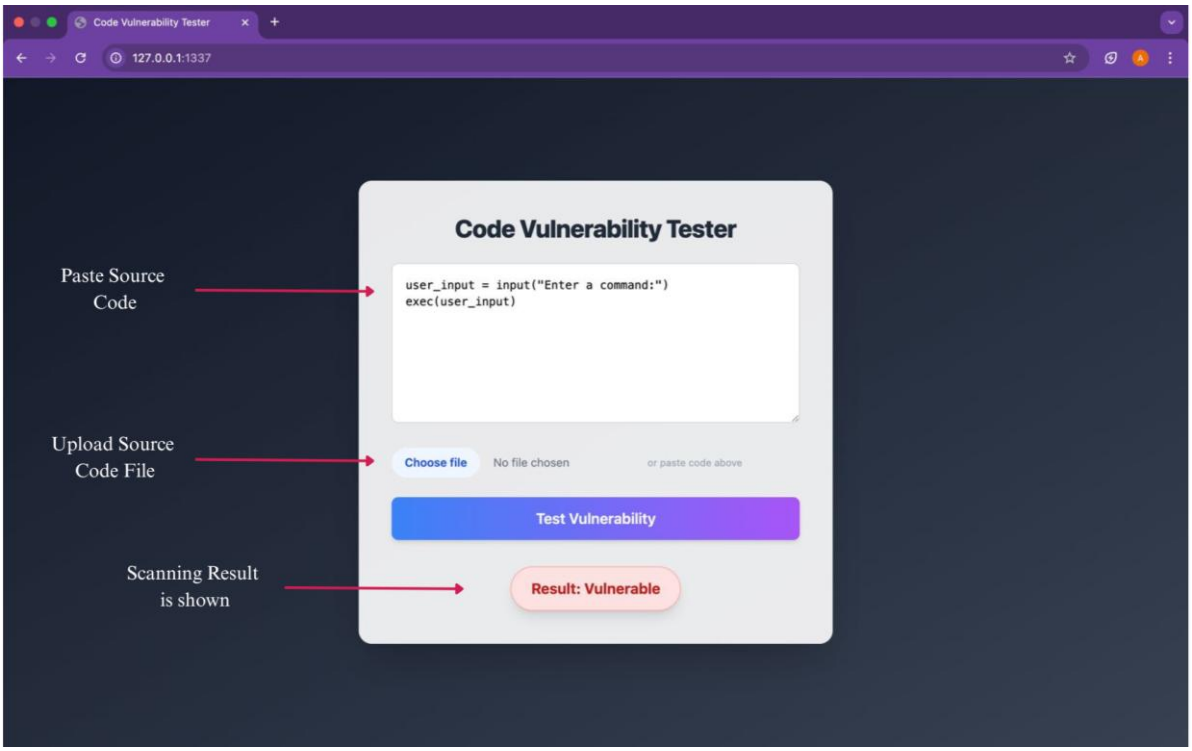


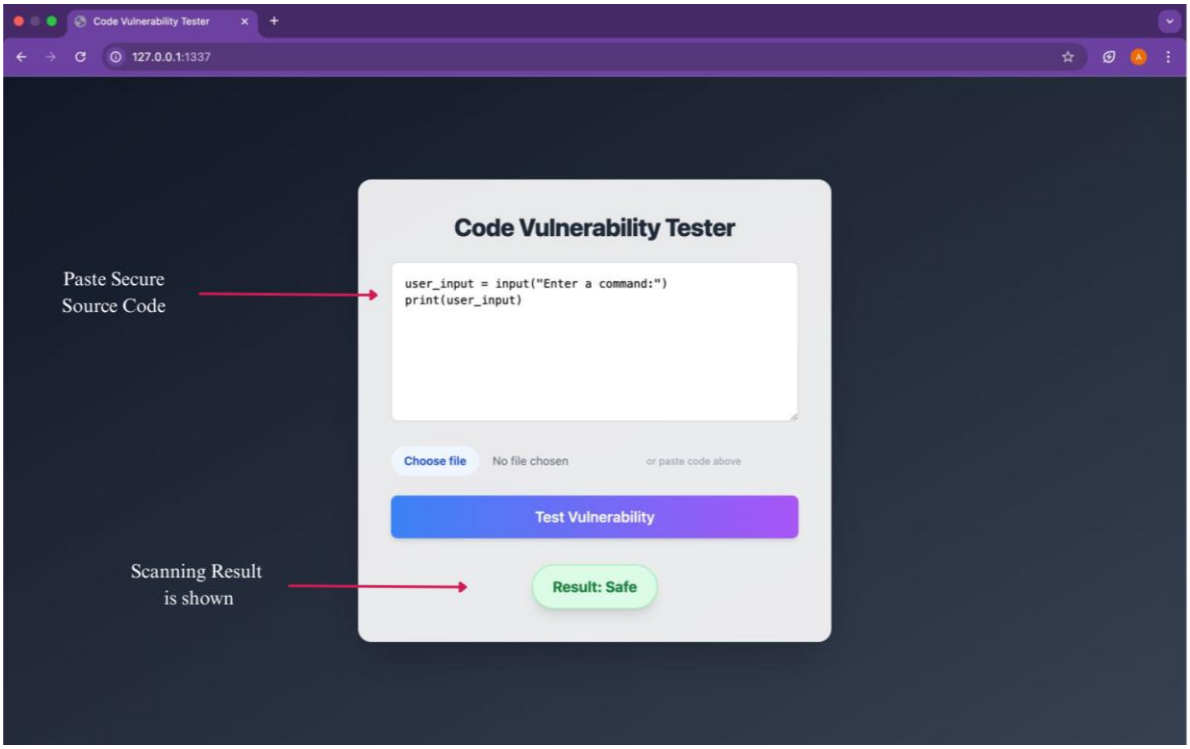
Figure 4.1: Agile SRLC Flow

CHAPTER 5 -
USER MANUAL

5.1 Scanning for Vulnerability

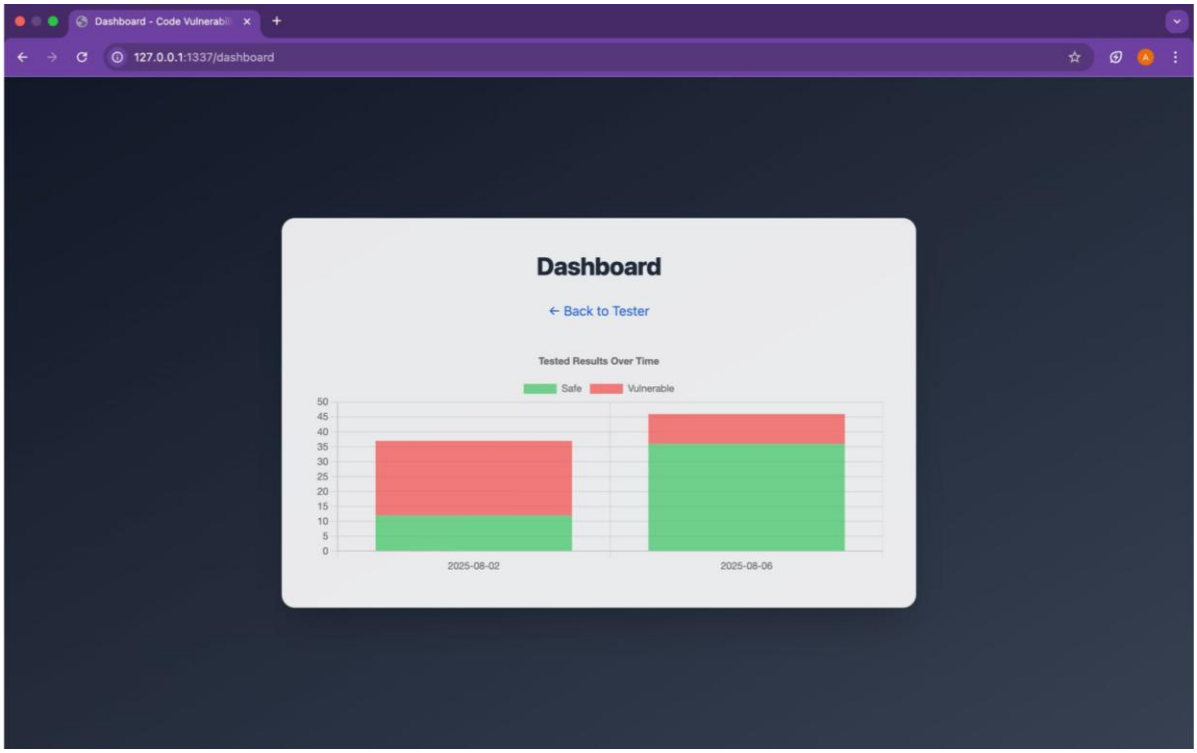


Different output will be shown for secure code.



5.2 Viewing Output

Vulnerability history will be shown in the dashboard.



CHAPTER 6 -
PROJECT SUMMARY

6.1 Project Summary

The web app "Vulnerability Detection in Source Code Using AI" uses a cutting-edge deep learning model to look for holes in source code. There are mainly two types of users for this application. The application system can support: developers and security analysts.

Users can upload files or copy code to add to the source code, and the site lets users start using an AI model to find security holes. The system gives a full report after the scan is done. This security report lists the types of vulnerability, severity of the issue, and exact line number or location of any vulnerabilities that were found. The application will show the findings in a graph so that users can better understand vulnerability trends and security risks associated with source code.

The Agile technique was used to plan, create, and manage my project in several stages of development. After successfully deploying the application, it is now prepared for testing in the real world with real user interaction and find out ways for further improvements of the features . In order to improve its effectiveness for user need and make the usage more easily understandable, the project has undergone a number of modifications and improvements over time during the development stage.

6.2 Future Scope

To make my project more practical and usable for developers, future research and iterations could incorporate the following features and enhancements:

- Programming Support: Expand the system's functionality to enable it to scan code written in various programming languages that are not regularly used in day to day development.
- Allow developers to scan their source code directly from their IDEs or browser-based editors with real-time code editor integration while developing the application. This will enable them for better detection efficiency and vulnerability detection in between the development workflow.
- Advanced Threat Classification: Use an improved AI model that is pre-trained with coding related task along with bigger labeled dataset. This process will enable the model to effectively categorize and understand source code vulnerabilities for users. Then analyse and provide feedback for those issues according to risk levels and real-world updated vulnerability data.
- Secure REST API Access: Create REST APIs that will enable users with application for secure connections between third-party systems and automated vulnerability scanning solution for better integration with multiple use case.
- User Feedback Loop: To enhance the overall system performance to detect security vulnerabilities accurately during the scan, allow users to deliver their input on overlooked vulnerabilities by the solution or false positives in the result.

By including these capabilities above, my AI powered vulnerability detection system will become more intelligent in vulnerability detection, root cause analysis of the issues and then deliver useful guidance for remediation for the developers and cybersecurity analysts. This will enable my targeted user groups to improve their software security and protection during the various development stage.

6.3 References

[1] Samiha Shimmi, Hamed Okhravi, & Mona Rahimi. (2025). "AI-based software vulnerability detection: A systematic literature review," *arXiv preprint arXiv:2506.10280*.

[2] Nima Shiri Harzevili, Alvine Boaye Belle, Junjie Wang, Song Wang, Zhen Ming Jiang, & Nachiappan Nagappan. (2023). "A survey on automated software vulnerability detection using machine learning and deep learning," *arXiv preprint arXiv:2306.11673*.

[3] Guan-Yan Yang, Yi-Heng Ko, Farn Wang, Kuo-Hui Yeh, Haw-Shiang Chang, & Hsueh-Yi Chen (2024). "Automated vulnerability detection using deep learning techniques," *arXiv preprint arXiv:2410.21968*.

[4] Zhen Huang, & Amy Aumpansub. (2024). "Vulnerability detection with deep learning," *arXiv preprint arXiv:2405.12384*.

[5] Fazli Subhan, Xiaoxue Wu, Lili Bo , Xiaobing Sun, & Muhammad Rahman. (2022). "A deep learning-based approach for software vulnerability detection," *IET Software, 16*(3), 245-258.

[6] Xin Zhou, Jianmin Pang, Feng Yue, Fudong Liu, Jiayu Guo, Wenfu Liu, Zhihui Song, Guoqiang Shu, Bing Xia & Zheng Shan. (2022). A new method of software vulnerability detection based on quantum neural network. *Scientific Reports, 12*, 7982.

[7] Yege Yang, & Guiping Li. (2025). "Code vulnerability detection based on graph neural network. *International Journal of Advanced Network, Monitoring and Controls," 10*(1), 123-135.

[8] Zhaoyang Chu, Yao Wan, Qian Li, Yang Wu, Hongyu Zhang, Yulei Sui, Guandong Xu, & Hai Jin. (2024). "Graph neural networks for vulnerability detection: A counterfactual explanation," *arXiv preprint arXiv:2404.15687*.

[9] Ruitong Liu, Yanbin Wang, Haitao Xu, Jianguo Sun, Fan Zhang, Peiyue Li, & Zhenhao Guo. (2024). "Vul-LMGNNs: Fusing language models and online-distilled graph neural networks for code vulnerability detection," *arXiv preprint arXiv:2404.14719*.

[10] Soolin Kim, Jusop Choi, Muhammad Ejaz Ahmed, Surya Nepal, & Hyoungshick Kim. (2022). "VulDeBERT: A vulnerability detection system using BERT. *Security Lab Technical Report*," Sungkyunkwan University.

[11] Junze Hu, Xiangyu Jin, Yizhe Zeng, Yuling Liu, Yunpeng Li, Dan Du, Kaiyu Xie, & Hongsong Zhu. (2025). "QLPro: Automated code vulnerability discovery via LLM and static code analysis integration," *arXiv preprint arXiv:2506.23644*.

[12] Neophytos Christou, Di Jin, Vaggelis Atlidakis, & Baishakhi Ray. (2023). "Automated vulnerability discovery in deep learning frameworks," In *32nd USENIX Security Symposium* (pp. 2245-2262). USENIX Association.

[13] Niklas Risse, & Marcel Böhme. (2024). "Uncovering the limits of machine learning for automatic vulnerability detection," In *33rd USENIX Security Symposium* (pp. 1823-1840). USENIX Association.

