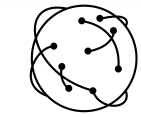
Consolidated Vulnerability Assessment and Reporting

Team:



ASTRO

Client:

bitm

Making Cyber Security Practical

COS 301
Capstone Project
University of Pretoria

Name	Student Number
Alec Watson	u22491351
Aren Repko	u04515791
Daniel Geerdink	u22556860
Edward Harvey	u22517783
Jacques Malan	u22554107

Introduction

The COVAR project by BlueVision ITM aims to develop a comprehensive vulnerability assessment and reporting tool designed to consolidate and analyze data from various cybersecurity assessment tools. By integrating findings from multiple vulnerability scanners, COVAR will provide a unified view of an organization's security posture, enabling clients to identify and mitigate potential risks more efficiently.

Background & Motivation

In the current cybersecurity landscape, organizations rely on a multitude of vulnerability scanning and assessment tools, each generating reports with unique signatures, detection techniques, and formats. This diversity often leads to inconsistencies and requires security engineers to spend significant time correlating and validating findings, which delays the reporting process and diminishes the value provided to clients. The COVAR project seeks to streamline this process by consolidating results from various tools, thereby enhancing the efficiency of vulnerability assessments and increasing the actionable insights available to clients.

Objectives

The primary objectives of the COVAR project include:

1. **Development of a Consolidated Reporting Tool:** Create a robust and scalable tool that integrates findings from various vulnerability assessment tools, providing a comprehensive view of an organization's security risks.
2. **Enhanced Client Reporting:** Enable clients to receive detailed, customizable reports that highlight their risk exposure, with options to exclude false positives and add false negatives.

3. **Visualization of Vulnerability Trends:** Present data in a user-friendly, graphical format, showing vulnerability status over time to help clients track improvements and emerging threats.
4. **Periodic Risk Exposure Reports:** Provide clients with regular updates on their security posture, facilitating proactive risk management.
5. **Role-Based Access Control:** Implement a centralized web application with defined roles and access levels, ensuring secure and appropriate access to sensitive data.
6. **Automation and Integration:** Allow for automated fetching of reports via API, integration with threat platforms, and the ability to report critical findings through ticketing systems or email alerts.
7. **Risk Quantification and Industry Comparison:** Develop a risk model to quantify potential risks based on observed vulnerabilities, client industry, and active threat actors, enabling clients to benchmark their security posture against similar organizations.

Scope of the Project

The scope of the COVAR project is extensive and includes several critical components necessary for its successful implementation:

- **Data Consolidation:** Importing and standardizing data from popular vulnerability assessment tools such as GreenBone, OpenVAS/GVM and Nessus, and establishing a common data model for reporting.
- **Report Customization and Visualization:** Designing a flexible report generation engine that allows users to customize report layouts and visualize vulnerability data effectively.
- **Role-Based Access Management:** Ensuring complete segmentation of client data and implementing strict access controls based on user roles and responsibilities.
- **API Development and Integration:** Using technologies such as NestJS for the main API
- **Database Management:** Utilizing PostgreSQL for user management and Redis for caching, ensuring efficient data handling and retrieval.
- **Deployment and Documentation:** Providing comprehensive documentation for deploying the final product on Linux-based servers and offering a user manual for key features and functionalities.

- **The COVAR project** will be deployed within BlueVision ITM's cloud infrastructure, with development support provided through local or cloud-hosted servers. The project team will ensure detailed documentation and support to facilitate the deployment and usage of the final product.

Use Cases

User Characteristics:

1. **Admin**:
    - **Role**: Admins are special users who can appoint VAs (Vulnerability Assessors) and assign them to clients.
    - **Responsibilities**: They manage penetration testing allocation, performing appointments or dismissals of VAs.
    - **Interactions**: Admins oversee the coordination between VAs and clients, ensuring that testing is carried out efficiently across multiple clients within an organization.
2. **Vulnerability Assessors (VAs)**:
    - **Role**: VAs are users who perform penetration testing.
    - **Responsibilities**: They inspect clients' systems for vulnerabilities, generate reports, and provide recommendations.
    - **Needs**: VAs require tools for quick and easy report generation, tailored to different clients' needs. They may need to handle multiple clients within the same organization, conducting comprehensive assessments and generating consolidated reports.
    - **Appointment**: This role is open only to those appointed by the Admin.
3. **Client**:
    - **Role**: Clients are users who are interested in the reports generated about their system by the CoVAR system and the VA.
    - **Types**:
        - **Business Executives**: Interested in high-level summaries of vulnerabilities and overall security posture.
        - **Technical Individuals**: Responsible for fixing the issues highlighted in the reports, requiring detailed technical information.
    - **Needs**: Clients may be part of larger organizations, and thus, require reports formatted to meet both executive and technical needs. They need

the capability to view consolidated reports for the entire organization and detailed reports for individual systems or departments.
    - **Interactions**: Clients interact with both the CoVAR system and the VAs to receive tailored reports, track vulnerability remediation, and ensure ongoing security improvements.

By incorporating these characteristics, the system can better accommodate the complex needs of organizations with multiple clients, ensuring that all stakeholders receive the information they need in the most appropriate format.

.

User Stories

User Story 1: As a VA (Vulnerability Assessor)

1. **Log in**
    - **Story**: I want to log into the system securely.
    - **Goal**: So that I can access and manage client data and generate reports.
2. **Fetch client data**
    - **Story**: I want to feed client data results from different security tools into the system.
    - **Goal**: So that I can consolidate all the data into a single report.
3. **Consolidate Data**
    - **Story**: I want to have the system automatically consolidate data from different security tools.
    - **Goal**: So that I can have a unified view of the client's security status.
4. **Validate and Modify Data**
    - **Story**: I want to manually validate and change outputs in the consolidated data.
    - **Goal**: So that I can ensure the accuracy and reliability of the final report.
5. **Resolve Contradictions**
    - **Story**: I want to resolve contradictions in the consolidated data.
    - **Goal**: So that the final report accurately reflects the client's security status.
6. **Generate a Report**
    - **Story**: I want to generate the final report in a format of my choosing.
    - **Goal**: So that it meets the specific requirements of the client or stakeholders.

User Story 2: As a Client

1. **View report**
    - **Story**: I want to view my security report.
    - **Goal**: So that I can understand the status and findings related to my security.
2. **Download report**
    - **Story**: I want to download my security report.
    - **Goal**: So that I can keep a local copy for my records.
3. **Modify report format**
    - **Story**: I want to change the format of my security report.
    - **Goal**: So that I can view or download it in a format that suits my needs.
4. **Create Organization**
    - **Story**: I want to create an organization within the system.
    - **Goal**: So that I can manage and organize my company's users and security reports.
5. **Join Organization**
    - **Story**: I want to join an organization when invited by another client.
    - **Goal**: So that I can collaborate and access shared resources and reports.

User Story 3: As an Admin

1. **Assign and Unassign VAs**
    - **Story**: I want to assign VAs to clients and unassign them.
    - **Goal**: So that VAs can generate reports for the clients.
2. **Promote Users to VAs**
    - **Story**: I want to promote users to VAs.
    - **Goal**: So that I can increase the number of VAs available for penetration testing.

Functional Requirements

1. User Login

1.1 The system shall provide a login page accessible via a web browser. 1.2 The login page shall require a username and password. 1.3 The system shall support multi-factor authentication (MFA) for added security. 1.4 The system shall validate user credentials against a secure database. 1.5 The system shall provide appropriate error messages for invalid login attempts. 1.6 The system shall allow users to reset their passwords securely if they forget them. 1.7 Upon successful login, the system shall redirect the user to their dashboard or home page.

2. Generating Reports

2.1 The system shall accept data inputs from various security tools.

2.1.1 The system shall support multiple file formats (e.g., CSV, JSON, XML).

2.2 The system shall provide a user-friendly interface for uploading and importing data.

2.3 The system shall merge data from various sources into a single report.

2.3.1 The system shall identify and highlight contradictions in the data.

2.3.2 The system shall allow the VA to manually resolve contradictions and update the report.

2.4 The system shall allow customization of the report layout and content.

2.5 The system shall generate the report in the selected format and make it available for download or sharing.

3. Client View/Modify Report

3.1 The system shall display a dashboard with a list of available reports for the client.
3.2 The system shall provide a download option for the client to download their report in the selected format.

3.3 Clients shall be able to interact with the report, such as clicking on specific vulnerabilities to get more detailed information.

3.4 The system shall ensure that clients can only view and modify reports related to their own organization.

3.5 Role-based access control (RBAC) shall be implemented to restrict access to certain features based on user roles.

3.6 The system shall notify clients via email or in-app notifications when a new report is available or when there are updates to existing reports.

4. Admin Assignment and User Management

4.1 The system shall allow an admin user to assign VAs to clients.

4.2 The system shall allow an admin user to unassign VAs from clients.

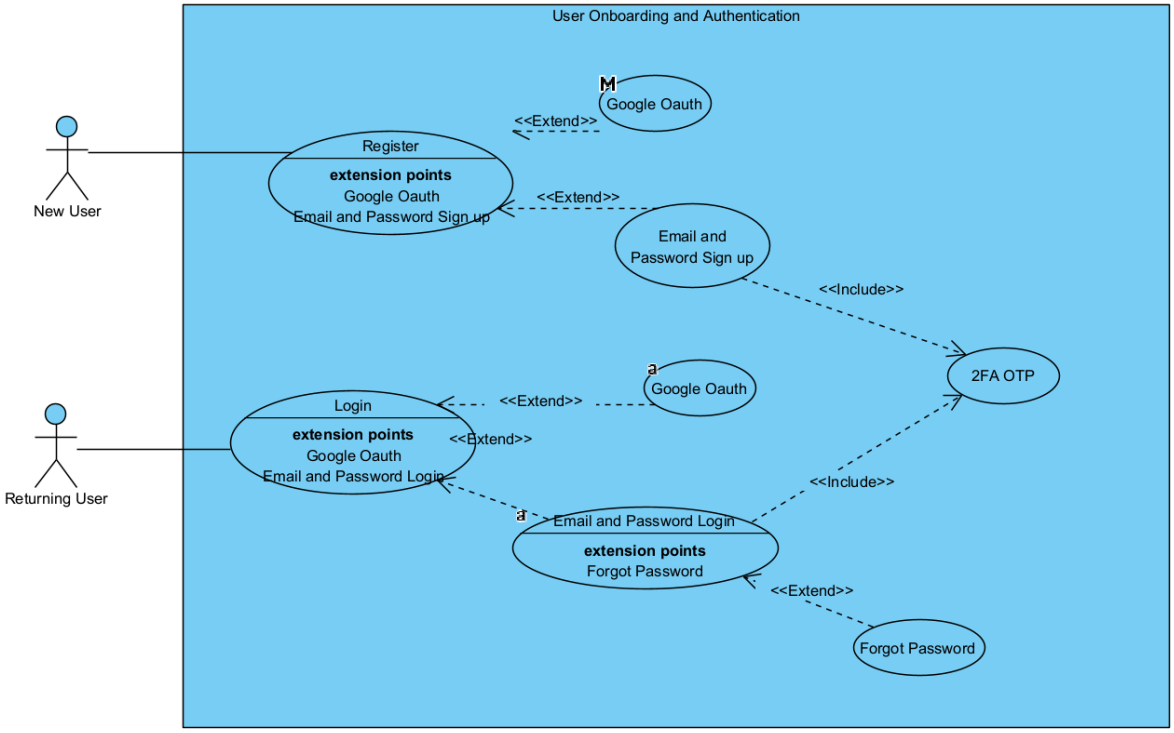
4.3 The system shall allow an admin user to promote regular users to VAs.

5. Organization Management by Clients

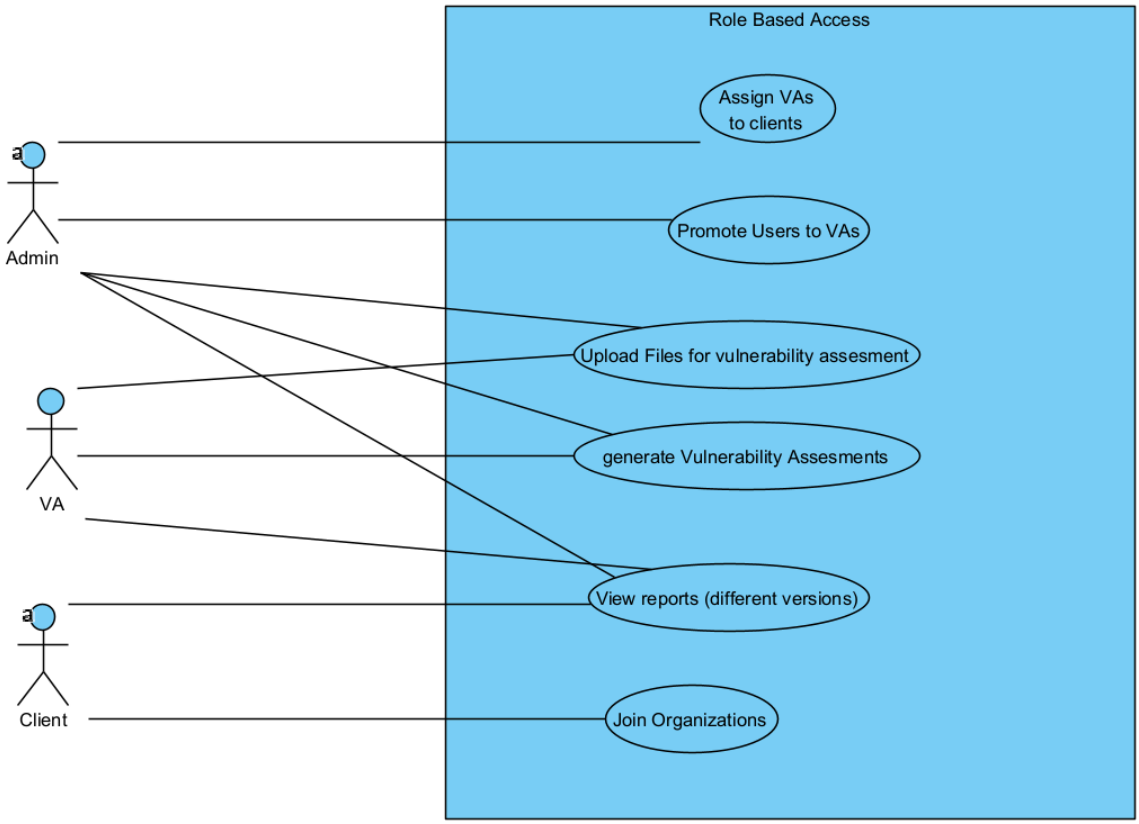
5.1 The system shall allow clients to create new organizations within the system. 5.2 The system shall allow clients to invite other users to join their organization. 5.3 The system shall provide an interface for clients to manage their organization's member

Subsystems

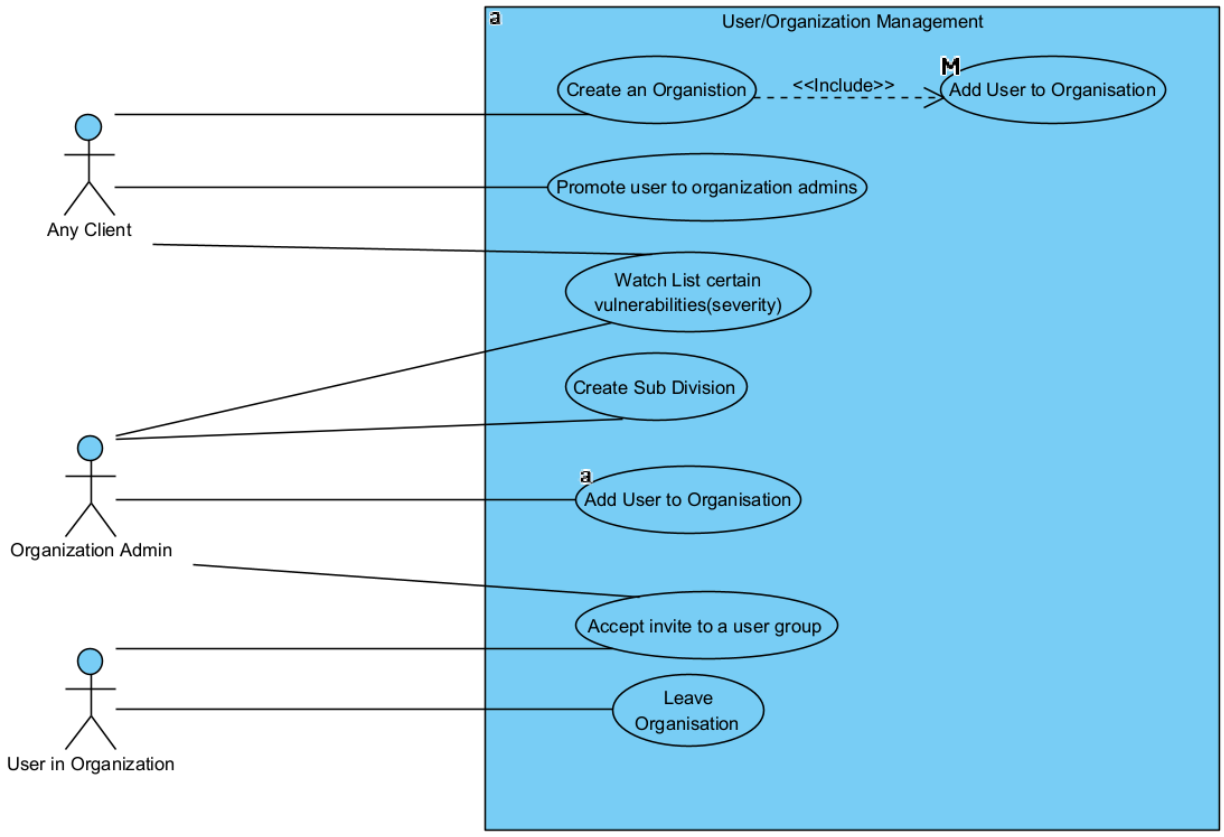
User Onboarding and Authentication Subsystem

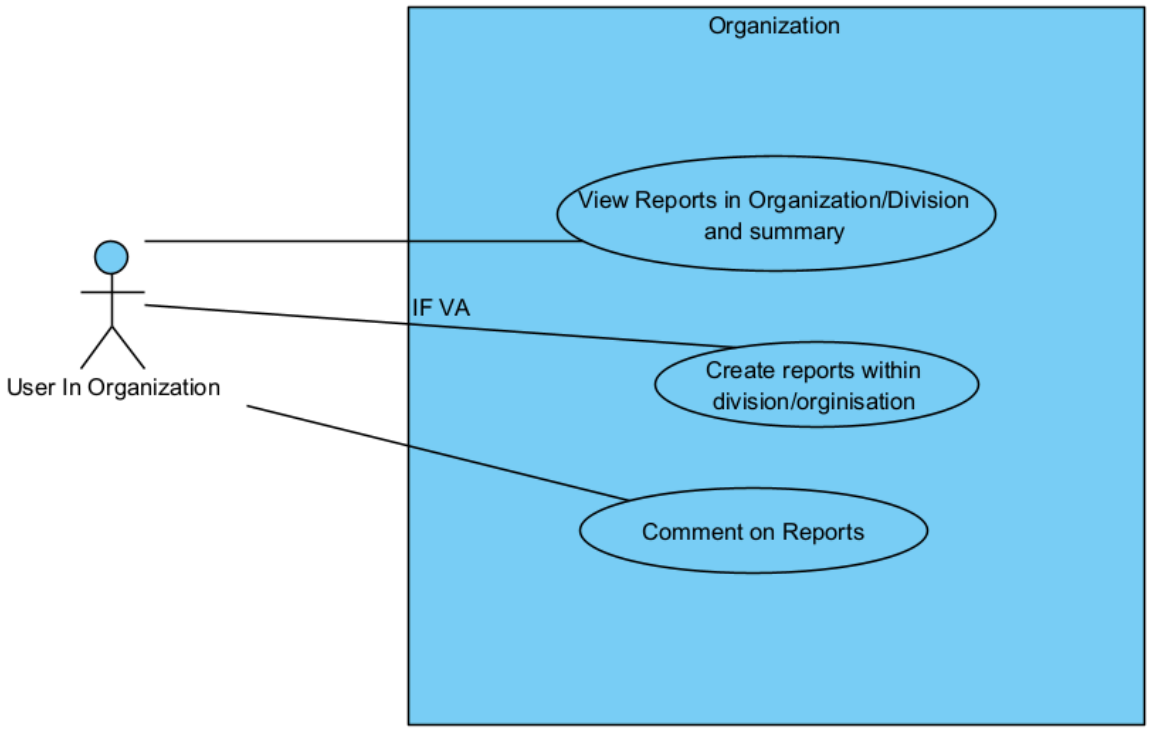


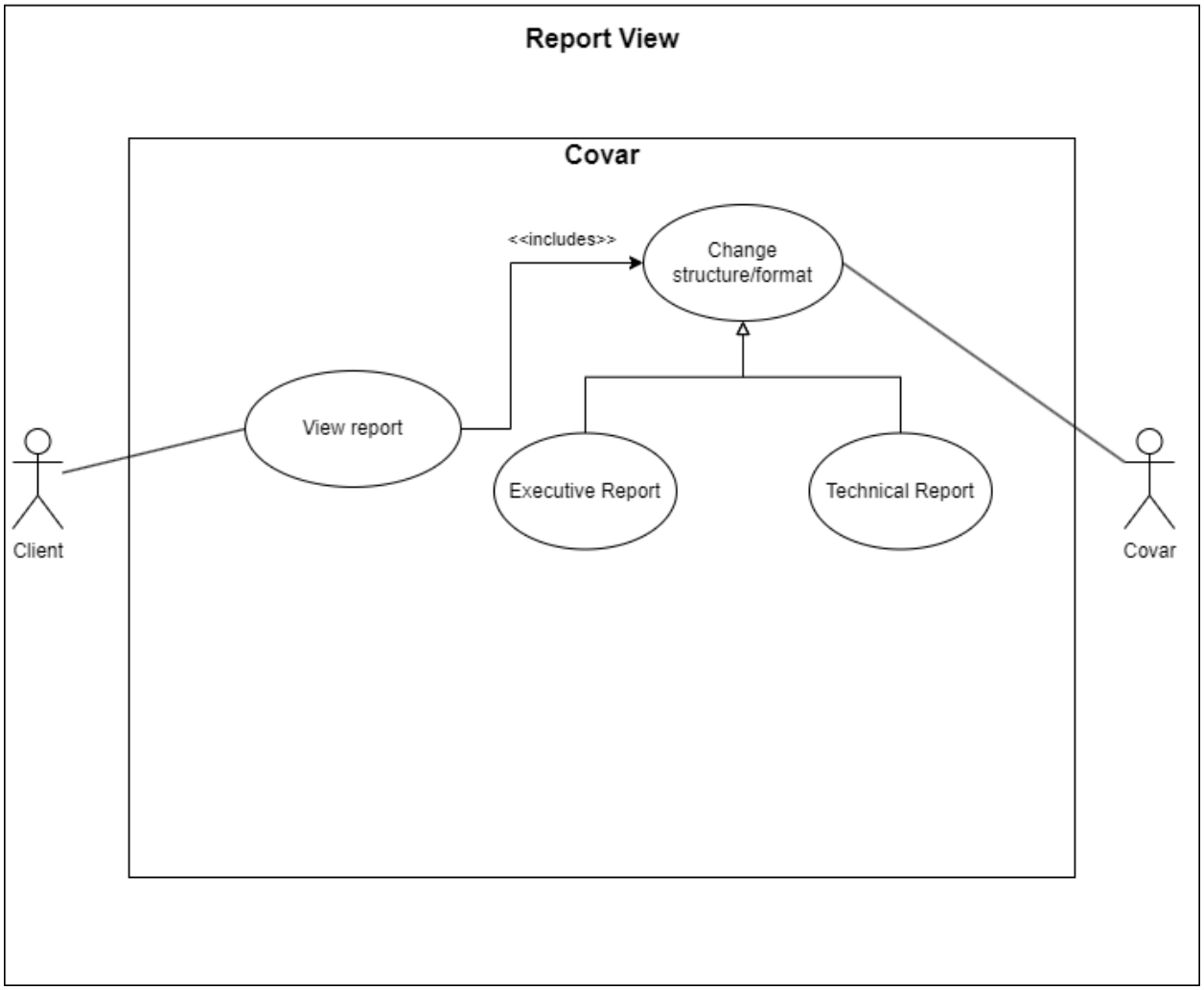
Role Based access of client


Role Based Access
Assign VAs to clients
Promote Users to VAs
Upload Files for vulnerability assesment
generate Vulnerability Assesments
View reports (different versions)
Join Organizations
Admin
VA
Client

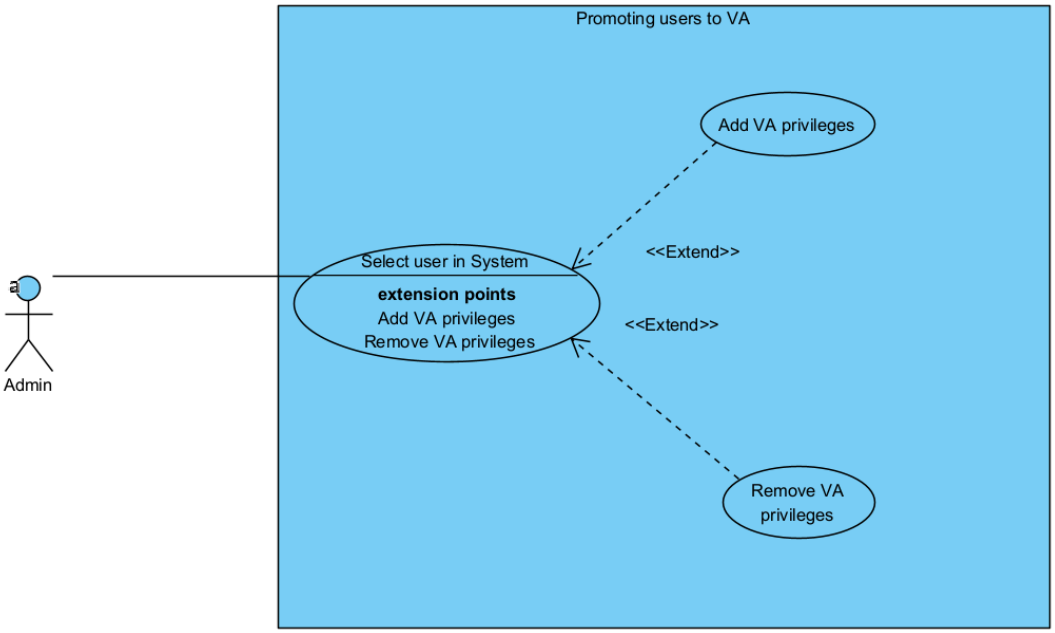
User Organization Subsystem

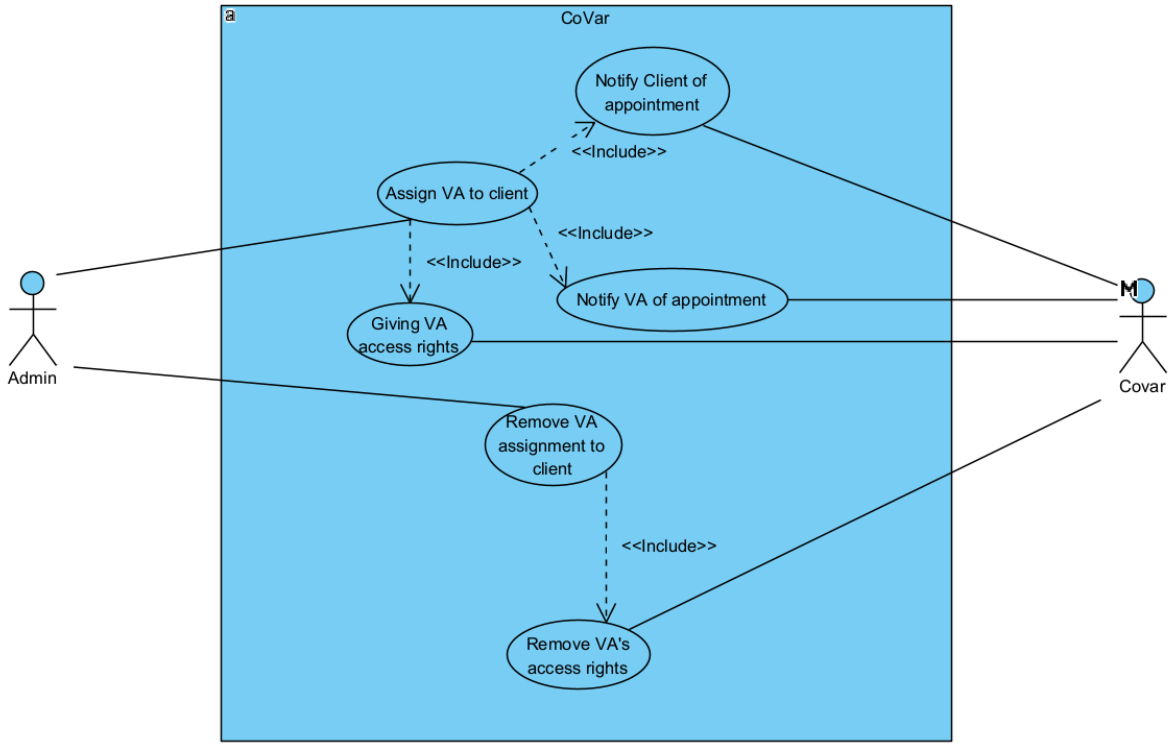



Organization
View Reports in Organization/Division and summary
IF VA
Create reports within division/orginisation
Comment on Reports
User In Organization

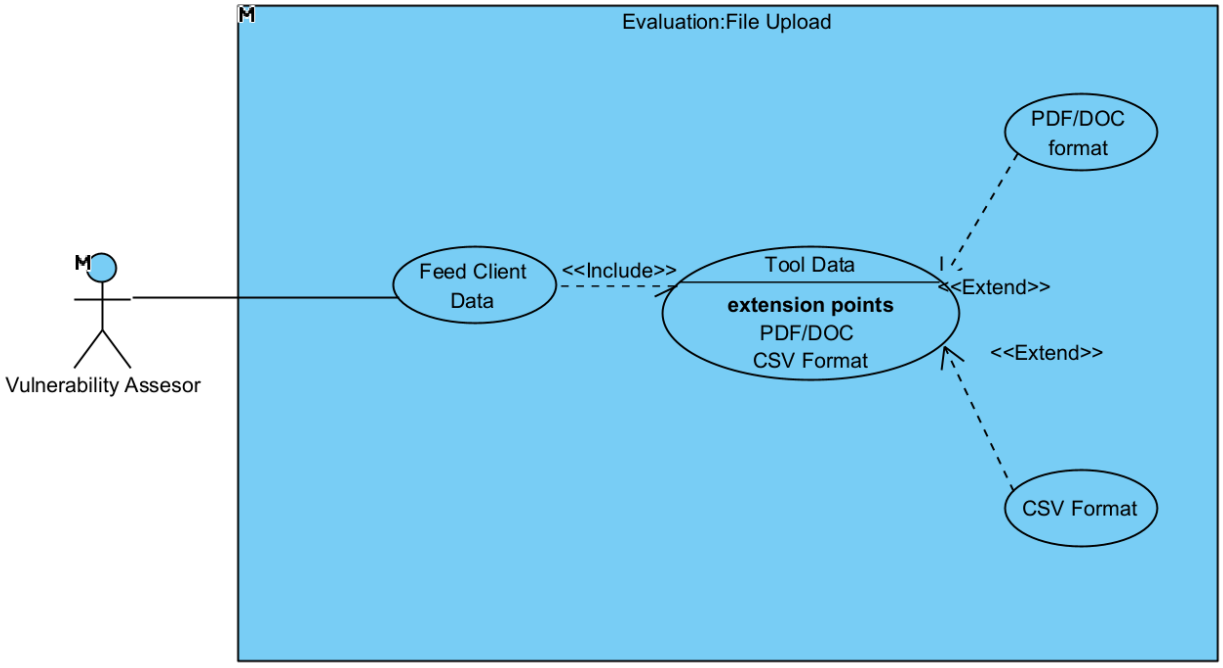

Report View
Covar
<<includes>>
Change structure/format
View report
Executive Report
Technical Report
Client
Covar

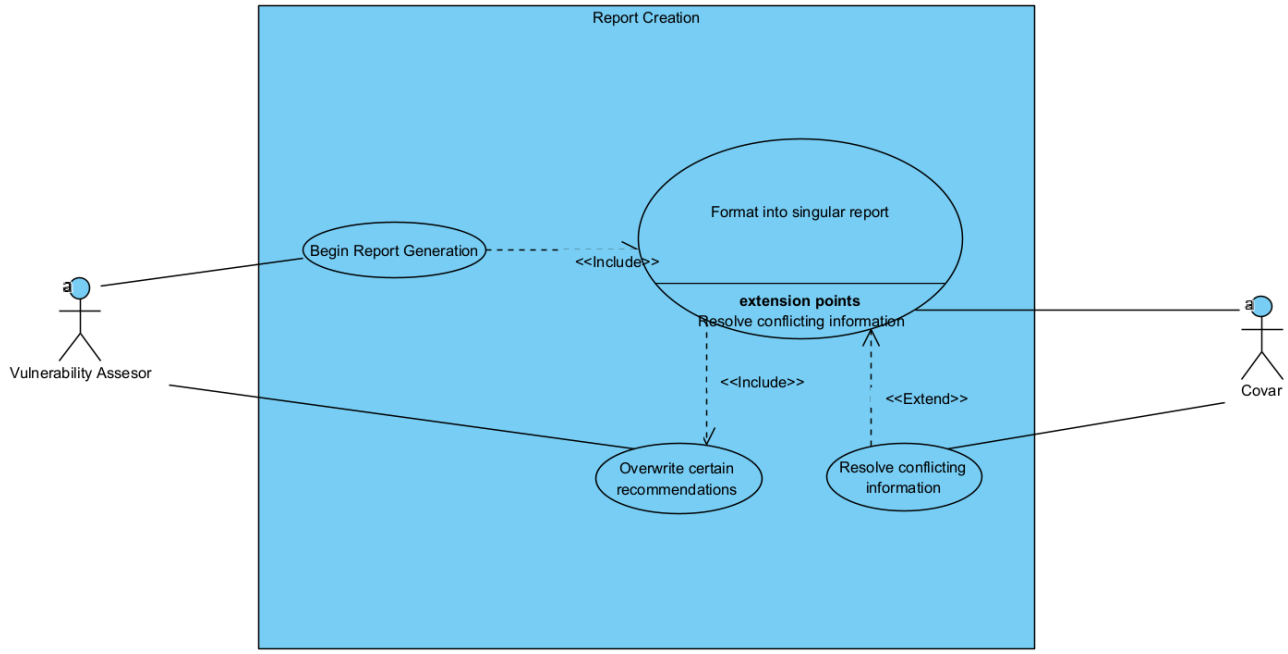
Admin Tools Subsystem





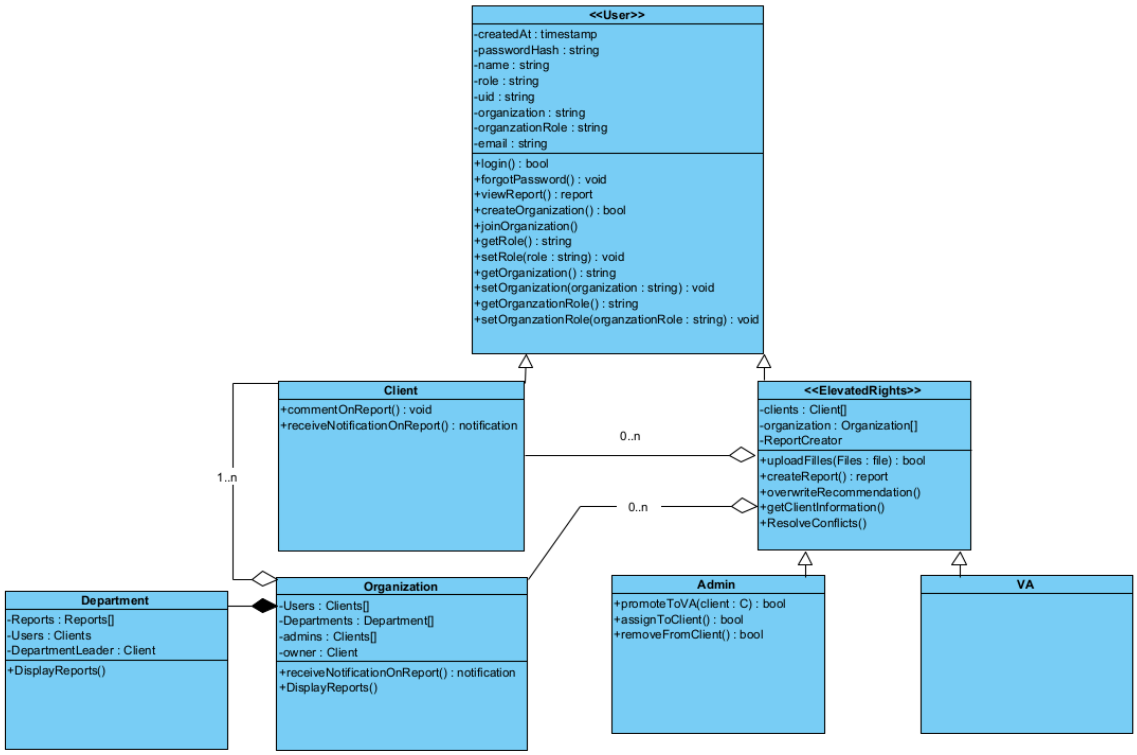
VA Subsystem



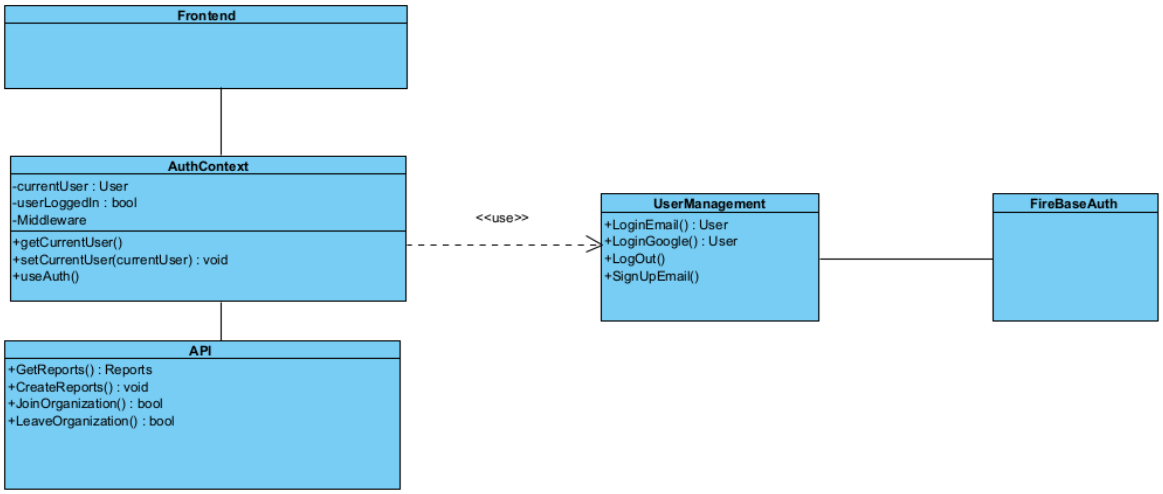


Class Diagram

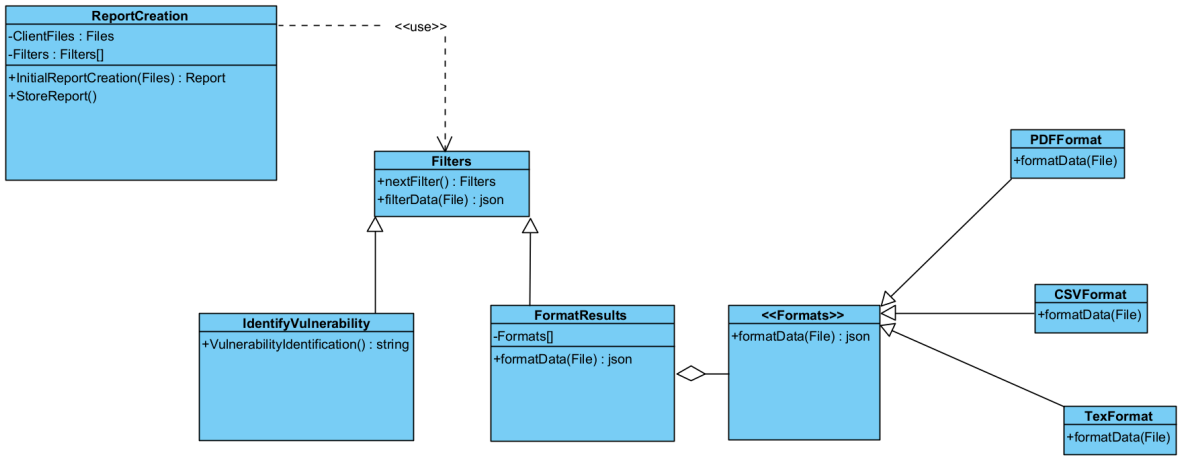
Users/Organization Interactions



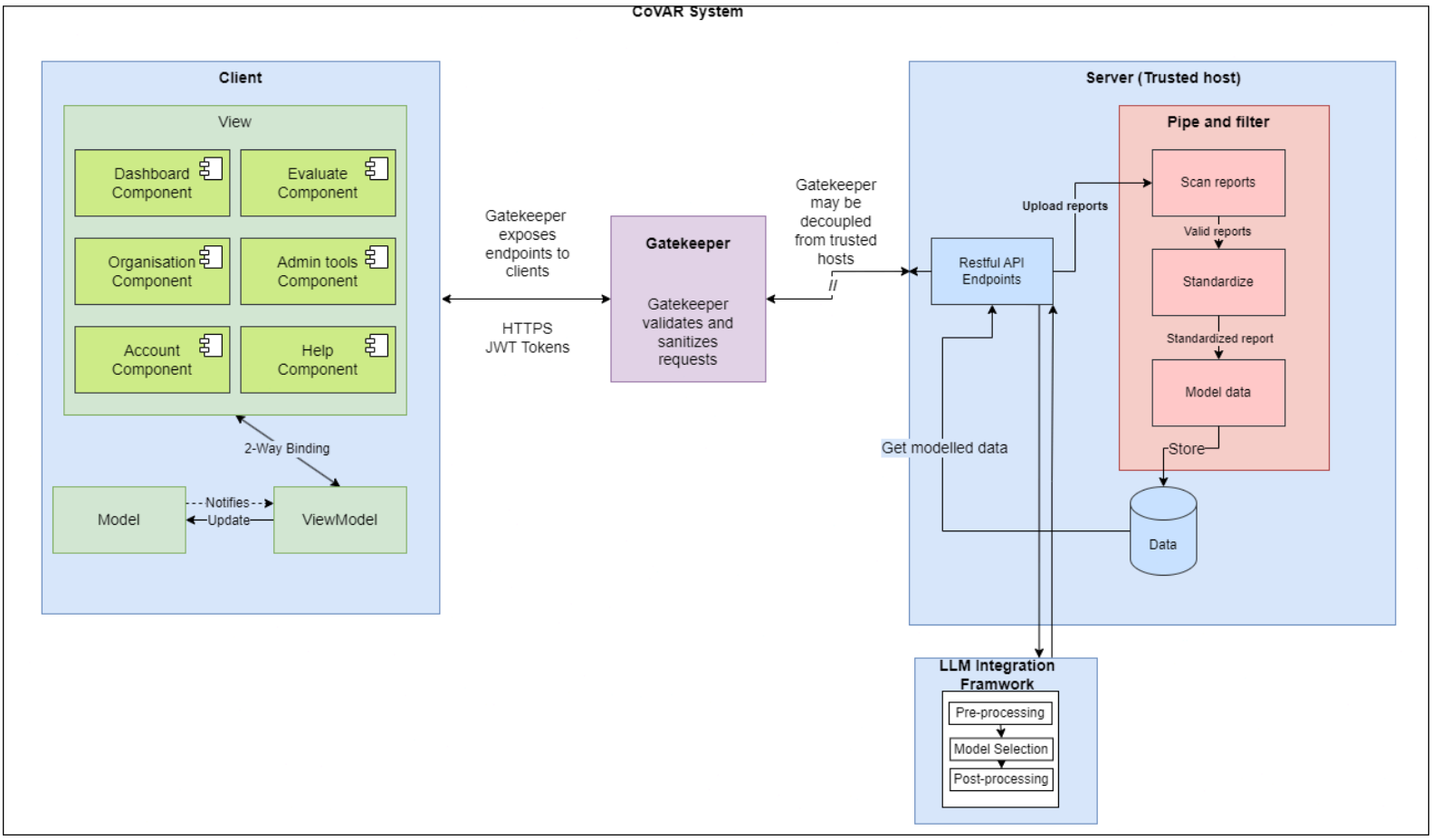
App FrontEnd Interaction



Unification into common Data Model



Architecture



Reasoning:

Client Server:

- **Performance:**
  - Our system is not catering for a large amount of users so client-server will have enough performance

- **Security:**
  - Single point of access makes it easier to enable a secure system because there is only one point to secure
  - Centralized control allowing client to have control over all security measures in one place

    - Log access to resources/information will be in one place
- **Availability:**
    - Allows for easy logging of issues/errors
- **Maintainability**
    - Easy to maintain (1 thing to maintain) and the client has control of said resources
- **Other:**
    - Our client (BITM) allocated a Virtual Private server to which suits the client server architecture.
    - Client has singular server resources allocated for the project.

Gatekeeper:

- **Security:**
    - All access to the server is sanitised and controlled by the gatekeeper.
    - There is limited risk and exposure as the gatekeeper has no access to credentials or keys
    - Runs in limited privilege mode, if the gatekeeper is compromised, the +attacker does not get access to any information.
- **Reliability:**
    - Since the gatekeeper is decoupled from the server, a compromised gatekeeper does not compromise the system.
    - DDOS protection provided by the gatekeeper improves reliability and lessens downtime.

Pipes:

- **Maintenance:**
    - It is easy to understand each filter that our PDF reports will go through.
    - Changes to one part of the chain of filtering does not affect any other part, because of modularity.
- **Performance:**
    - Multiple reports can be in the pipe simultaneously, thus increasing the throughput of the report processing.
- **Security:**
    - At each step, or before entering the pipe, we can validate that the report to be processed is valid and non-malicious.

Quality Requirements

1. Performance

1.1 The system shall load the report viewing page within 3 seconds under normal conditions.

1.2 The system shall support simultaneous access by at least 100 users without degradation in performance.

Architectural Strategies:

- **Caching:** Implement caching strategies to store frequently accessed data in memory to reduce database load and improve response times.
- **Spreading load over time through the use of concurrency:** Concurrently processing certain parts of the server operations to increase performance.
- **Optimized Queries:** Optimize database queries and use indexing to speed up data retrieval.
- **Data Partitioning:** Use partitioning techniques to divide large tables into smaller, more manageable pieces, improving query performance.
- **Elastic Scaling:** Use of container orchestration systems to automate scaling and management.

2. Security

2.1 The system shall use encryption (e.g., TLS) for all data transmissions to protect sensitive client data.

2.2 The system shall implement role-based access control (RBAC) to ensure only authorized users can view or download reports.

2.3 The system shall log all access and download activities for audit purposes.

2.4 Validating data, vulnerability reports should contain valid data and accurate data depending on the client and VA.

Architectural Strategies:

- **Encryption:** Utilize TLS for data in transit and AES for data at rest to ensure all data is protected.
- **Authentication and Authorization:** Implement robust multi-factor authentication (MFA) and role-based access control (RBAC) with constant authorization for requests
- **Least Privilege Principle:** Apply the principle of least privilege to ensure that users and services only have the minimum access necessary to perform their functions.

- **Single Access Point:** Design the system to have a single entry point for all users to reduce the number of access channels that need to be secured.
- **Logging and Monitoring:** Implement logging of all access and download activities using centralized logging solutions and monitor logs for suspicious activities.
- **Secure Development Practices:** Follow OWASP secure coding practices and use static analysis tools like SonarQube to identify security issues during development.

3. Compatibility

3.1 The system shall be compatible with major web browsers (e.g., Chrome, Firefox, Safari, Edge).

3.2 The system shall support integration with various security tools (Nessus, GreenBone) for data input.

Architectural Strategies:

- **Cross-Browser Testing:** Regularly test the system on major web browsers (using tools like Selenium or BrowserStack to ensure compatibility.
- **Standardized Web Technologies:** Use standardized web technologies and frameworks to ensure cross-browser compatibility.
- **Containerization:** Use containerization to package and deploy microservices consistently across different environments, ensuring compatibility and simplifying deployment.
- **API Standards:** Use RESTful APIs with standardized formats to ensure easy integration with various security tools.
- **Documentation and Testing:** Provide comprehensive documentation for the integration process and use automated testing to ensure ongoing compatibility.

4. Reliability and Availability

4.1 The system should maintain a high level of reliability, ensuring consistent performance and minimal downtime.

4.2 The system should achieve at least 99.9% availability, ensuring it is accessible and operational for users almost all the time.

Architectural Strategies:

- **Contracts-Based Architecture:** Defining clear contracts or interfaces between different components or services in the system.
- **Error/Exception Communication:** se standardized error messages, and log errors with relevant details.
- **Proper Logging:** Recording events, errors, and other significant activities in the system through the use of logging frameworks or other methods to monitor system health and possible issues.
- **Maintaining Backups:**Ensures data can be recovered in the event of corruption, accidental deletion, or disaster, thereby maintaining data integrity and availability.

5. Maintainability

5.1 The system shall be designed to facilitate easy maintenance and extension, allowing for rapid updates and the addition of new features without significant rework.

Architectural Strategies:

- **Modular Design:** Design the system in a modular way, with well-defined interfaces between components to facilitate easy updates and extensions.
- **Code Quality:** Adopt coding standards and best practices to ensure code quality and readability. Use linters and conduct code reviews.
- **Documentation:** Maintain comprehensive documentation for both the system architecture and the codebase to assist new developers and streamline maintenance.
- **Automated Testing:** Implement automated unit, integration, and end-to-end tests using tools like Jenkins or GitHub Actions to catch issues early and ensure that new changes do not break existing functionality.

Constraints

Technology Requirements

Frontend

ReactJS: A declarative JavaScript library for building user interfaces, providing a component-based architecture and efficient rendering.

Backend

NestJS: A progressive Node.js framework for building efficient and scalable server-side applications. It will be used for developing the main RESTful API.

Firebase Authentication: For secure and scalable user authentication, including support for multi-factor authentication.

PostgreSQL: A powerful, open-source object-relational database system for user management and report data storage.

Redis: An in-memory data structure store, used for caching to improve system performance.

File Storage and Management

Firebase Storage: For handling file uploads and storage, providing robust and secure storage capabilities.

Deployment

Docker: To containerize the application, ensuring consistent and reliable deployment across different environments.

Kubernetes: For orchestrating the deployment, scaling, and management of containerized applications.

Nginx: As a web server and reverse proxy to manage API requests efficiently.

Development and Documentation

Swagger: For API documentation, enabling easy interaction with the API endpoints during development and testing.

GitHub: For version control and collaboration, ensuring a streamlined development workflow.

Cypress : For unit and e2e testing ensuring the reliability and stability of the application.

Service Contracts

API Endpoints

User Authentication

Sign Up

URL: /api/auth/signup

Method: POST

Request Body:

json

```
{
  "email": "string",
  "password": "string",
  "name": "string"
}
```

Response:

json

```
{
  "userId": "string",
  "email": "string",
  "name": "string",
```

"token": "string"

}

Sign In

URL: /api/auth/signin

Method: POST

Request Body:

json

{

"email": "string",

"password": "string"

}

Response:

json

{

"userId": "string",

"email": "string",

"name": "string",

"token": "string"

}

Client Management

Add Client

URL: /api/clients

Method: POST

Request Body:

json

```
{
  "name": "string",
  "email": "string",
  "organization": "string"
}
```

Response:

json

```
{
  "clientId": "string",
  "name": "string",
  "email": "string",
```

```
  "organization": "string",
  "createdAt": "timestamp"
}
```

Get Clients

URL: /api/clients

Method: GET

Response:

json

```
[
  {
    "clientId": "string",
    "name": "string",
    "email": "string",
    "organization": "string",
    "createdAt": "timestamp"
  }
]
```

Report Management

Upload Report

URL: /api/reports

Method: POST

Request Body:

json

```
{
  "clientId": "string",
  "reportData": "object",
  "fileType": "string"
}
```

Response:

json

```
{
  "reportId": "string",
  "clientId": "string",
  "reportData": "object",
  "fileType": "string",
  "createdAt": "timestamp"
}
```

Get Reports

URL: /api/clients/:clientId/reports

Method: GET

Response:

json[

  {

    "reportId": "string",

    "clientId": "string",

    "reportData": "object",

    "fileType": "string",

    "createdAt": "timestamp"

  }

]

Download Report

URL: /api/reports/:reportId/download

Method: GET

Response: File download (binary data)