# Group 2-25

COS30049 Computing Technology Innovation Project – Final Design Document for Group Set 2

Henry Le, Quang Thang Dinh, Jade Hoang
GROUP SET 2 Team 2-25

## I. Project Background

### A. Introduction

The objective of the Smart Contract Audit System project is to create a platform that audits smart contracts and detects potential vulnerabilities in the code. The website provides features such as code uploading, and audit reporting.

This report covers both the frontend and backend development phases of the Smart Contract Audit System. It outlines our teams' implementation of the website's front end, highlighting the design and functionalities of the user-friendly interface. In the second phase of the project, our primary objective is to focus on backend development. This entails integrating core functionalities, and database components, and running Slither—a static analysis tool for smart contracts. The aim is to bring together these elements into a completed system, ensuring a robust and effective smart contract auditing platform.

### B. Background

Security in software development is of utmost importance, especially when dealing with blockchain smart contract code. Smart contracts are self-executing contracts with agreement terms directly written into code, forming a crucial part of blockchain-based systems. However, due to their immutable nature once deployed, any vulnerabilities in the code can have significant consequences (Frankenfield 2023). Therefore, developers need to audit their code to uncover any potential vulnerabilities. Our system, Cyber Tech, addresses this need by offering a comprehensive auditing service. Users can upload their smart contract code for analysis. The system then generates a detailed audit report outlining identified vulnerabilities and suggested improvements using Slither, an open-source static analysis tool that aids in contract comprehension (Trail of Bits 2018).

### C. Project Scope

The project has progressed to its second stage, focusing on the integration of both frontend and backend components. This entails transitioning from designing the visual aspects and user interface of the website (frontend) to creating the internal workings and functionalities of the platform (backend). The integration process is designed to guarantee the smooth and efficient functioning of all aspects of the website. The primary objective remains enhancing security within the blockchain ecosystem as well as identifying and strengthening vulnerabilities in smart contract code.

### D. Frameworks and Technologies Used

In developing the Smart Contract Audit System, we utilised various frameworks and technologies, catering to both the frontend and the backend. The front end is developed using **React.js**. Additionally, the following frameworks or libraries are also used as part of our design. While this is not an exhaustive list, here are some key components:

- **React-icons library**: for incorporating icons into the user interface.
- **React Router DOM**: enable multi-page React application.
- **Tailwind CSS**: a CSS framework for efficient styling.

- **React-markdown**: to convert the markdown syntax to React element, facilitating dynamic content display.
- **React-hot-toast**: to provide notification messages on the website such as successful report deletion or audit completion message to the user.
- **React-spinner**: to enhance user experience by including a loading state indicator, assuring users that their requests are being processed.
- **Axios**: to make API calls to the server, ensuring data exchange between the frontend and backend.

Our back end is primarily composed of the following elements:

- **FastAPI**: a backend framework, allowing the creation of APIs using Python.
- **slither-analyzer**: This tool is integrated to execute Slither commands, enabling the analysis of Solidity files when users upload them.
- **SQLAlchemy** and **mysql-connector-python**: these libraries are used to interact with the MySQL database, ensuring efficient data storage and retrieval.

For database management, **MySQL** is selected as the relational database for efficient data storage and retrieval, ensuring smooth backend operations.

## II. Team Introduction

**Workshop time**: Tuesday 4:30 pm-6:30 pm

| Team member | Student ID | Major |
|---|---|---|
| **Quang Thang Dinh** | 103522316 | Software development |
| **Henry Le (Thanh Hoang Le)** | 103795561 | Cybersecurity |
| **Jade Hoang (Minh Phuong Hoang)** | 103795587 | Cybersecurity |

*Table 1: Team member information*

## III. Project Requirements

### *A. Requirements*

The project requires 5 core functionalities that the final product must have:

**1. Upload smart contract file:**

- To analyse the smart contract, the user needs to be able to upload their smart contract code. The website provides an input form for this purpose, specifically the file upload method. The system exclusively accepts files with a .sol extension, which are Solidity files. In the event of users attempting to upload files with incorrect extensions, the system promptly responds with an error message, stating that only files with the .sol extension are permitted.

**2. Smart contract analysis using Slither tool:**

- The system uses the Slither tool for analysing the uploaded smart contract. This tool thoroughly assesses potential vulnerabilities and risks within the provided smart contract code. The analysis integrates with the website's backend by running the necessary Slither commands.
- The output of the audit will be a report containing detailed information about the categorised security vulnerabilities discovered in the smart contract. By identifying vulnerabilities and their suggestions, we can lower the chances of security breaches or exploits that could be taken advantage of by attackers.

**3. Database for the report on analysed smart contracts:**

- After uploading the smart contract for analysis, the user will be redirected to the **Report History page** to view the analysed report.
- These reports will be stored on the **Report History page**, which is a repository of all previously analysed reports.
- The listed reports will include essential information such as the contract's name, submission date, submission time, severity level, and options to view details or delete the report.
- To access the full list of identified vulnerabilities for each report, users will have to click the "Details" link, which redirects them to a dedicated **Detail Report page**.

**4. Displaying the analysis report:**

- A dedicated **Detail Report page** will detail all the categorised vulnerabilities and security risks identified as well as corresponding suggestions to improve the smart contract code.
- **Provide suggestions for the smart contract based on the analysis.**
- Suggestions for improving each detected vulnerability category will be placed on the **Detail Report page**.

### *B. Optional Features*

These features are not strictly required but we have implemented enhancements to the website to enhance user experience further:

**1. Delete Report:**

- *Location*: integrated within the Report History page's list and Detail report page.
- *Capabilities*: enables users to delete their existing reports and manage their audit history.

**2. Search and Sort:**

- *Location*: implemented on the Report History page.
- *Capabilities*:
  - **Search Bar:** allows users to locate specific reports based on criteria such as date or severity.
  - **Sorting**: allows users to arrange reports in either ascending or descending order.

• *Default order*: Initially, reports are arranged based on their submission date in descending order. This means that the latest reports will be displayed at the top of the list, providing users with quick access to the most recent audit outcomes.

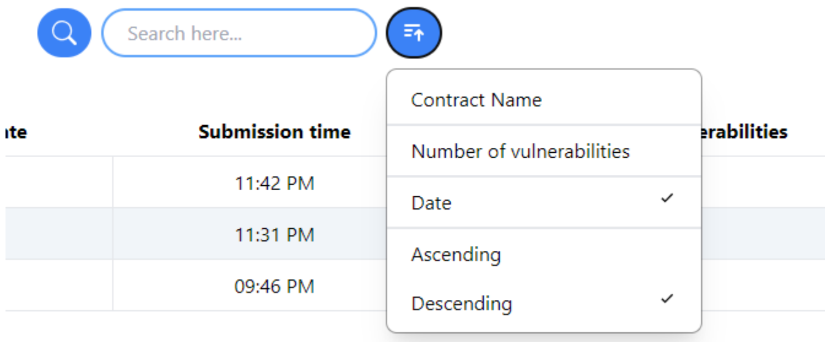


*Figure 1: Search bar and sort by feature.*

**3. Not found page:** presents a 404 error when users attempt to access a non-existing page. This feature contributes to a more informative and user-friendly experience in case of unexpected page requests.

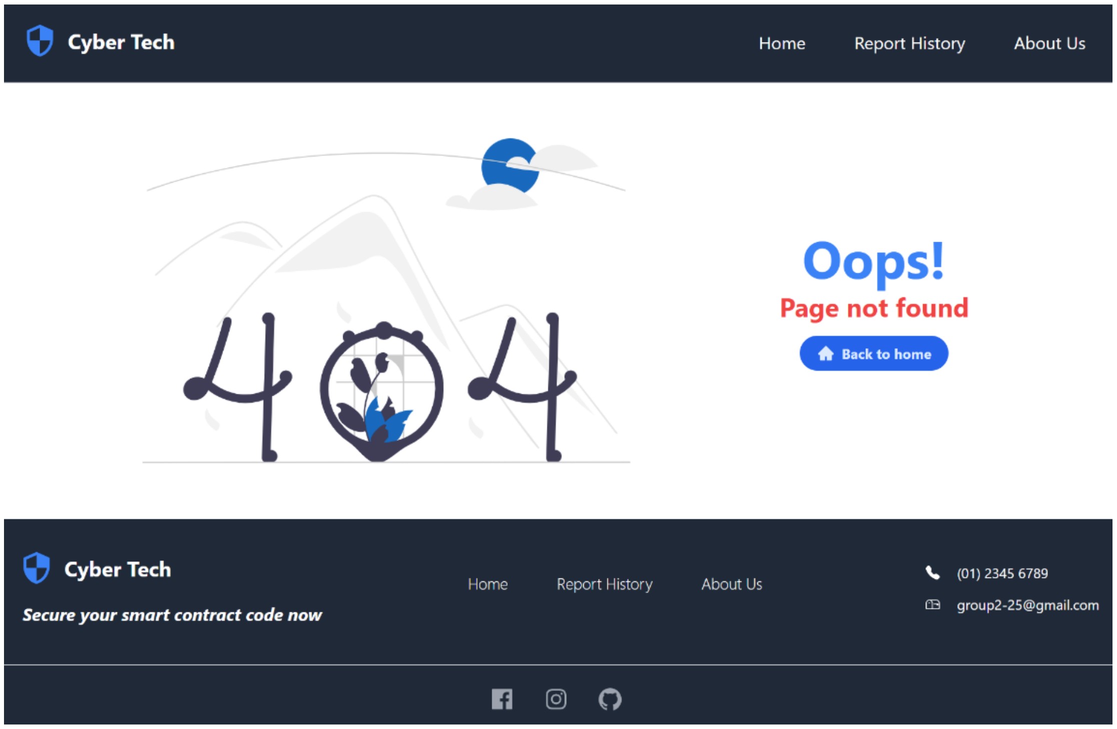


*Figure 2: Not found page.*

## IV. Project Design

### *A. Website Pages and System Requirements Alignment*

Our website consists of three main pages that are shown on the navigation bar, along with supplementary pages such as the Detail Report page:

| Page | Purpose |
|---|---|
| **Home** | Landing page for users to upload smart contract code for auditing. |
| **Report History** | Displays a list of all uploaded audit reports, allowing users to review history and retrieve previous reports. |
| **Detail Report** | This page is accessible via redirection from the Report History page. It displays in-depth analysis results (i.e., vulnerabilities, suggestions, and results of each vulnerability) for each report. |
| **About Us** | Present the project's goal and team information. |

*Table 2: Website pages and their purposes*

*For the screenshots of all pages, please refer to the Final front-end design section.*

The table below demonstrates how these pages align with the core functional requirements:

| System requirements | Implementation details |
|---|---|
| **Upload smart contract code for auditing, only accept .sol files.**<br><br>**Static analysis with Slither.** | **Home page**: file uploader that connects to FastAPI to conduct smart contract analysis. |
| **Display each audit report with identified vulnerabilities on a dedicated page.** | **Detail Report page**: displays the audit outcomes, including vulnerabilities' description and corresponding suggestions. |
| **Provide suggestions for each vulnerability category.** | **Detail Report page**: Show recommendations for each detected vulnerability. |
| **Store reports for future retrieval.** | **Report History Page**: features a report history section that displays a high-level overview of all submitted reports. |
| **User-friendly design.** | **All pages**: responsive navigation bar, footer, table, etc.<br><br>**Report History page**: search bar and sorting functionality. |

*Table 3: Alignment with project requirements*

## *B. User Workflow*

The typical user workflow, which occurs in the frontend of the application, involves the following steps:

1. The user begins by navigating to the **Home Page.**

2. On the **Home Page**, the user uploads a smart contract code file using the **file uploader**.

3. Static analysis:

- The system performs static analysis using the Slither auditing tool.
- During the analysis, a **wait message** is displayed to inform the user of the ongoing process.

4. Upon completion of the auditing process, the system automatically redirects the user to the **Detailed Report page**.

5. To access previous reports, the user navigates to the **Report History page**.

6. If the user wishes to filter, search, or sort the report list, they can use the **Search componen**t on the **Report History page**.

7. The user clicks on the **"Details"** link to view more information about a specific report.

8. From the **Detailed Report page**, the user has the option to navigate back to the **Report History page**.

9. If the user decides to delete a report, they can do so by clicking the delete link on the **Detail Report page** or navigating to the **Report History page**.

10. The **Detail Report page** provides a detailed analysis of the smart contract, including vulnerabilities, and suggestions to resolve each vulnerability, the results of each vulnerability and the location of that vulnerability in the uploaded smart contract code.

11. To learn more about the auditing system, the user can navigate to the **About Us page**.

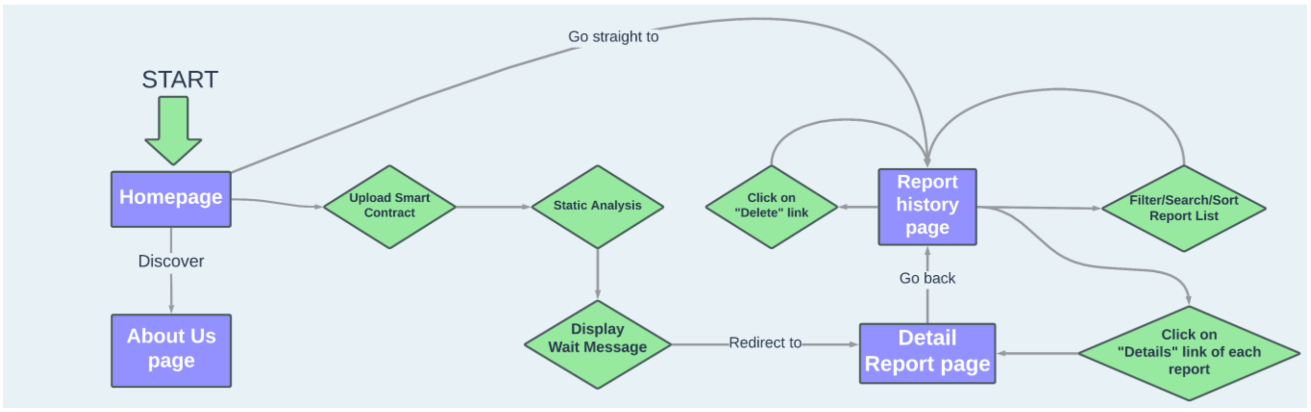


*Figure 3: User workflow and webpage interaction diagram.*

## *C. Overall System Architecture Design*

The following diagram illustrates the interaction between front-end and back-end components:

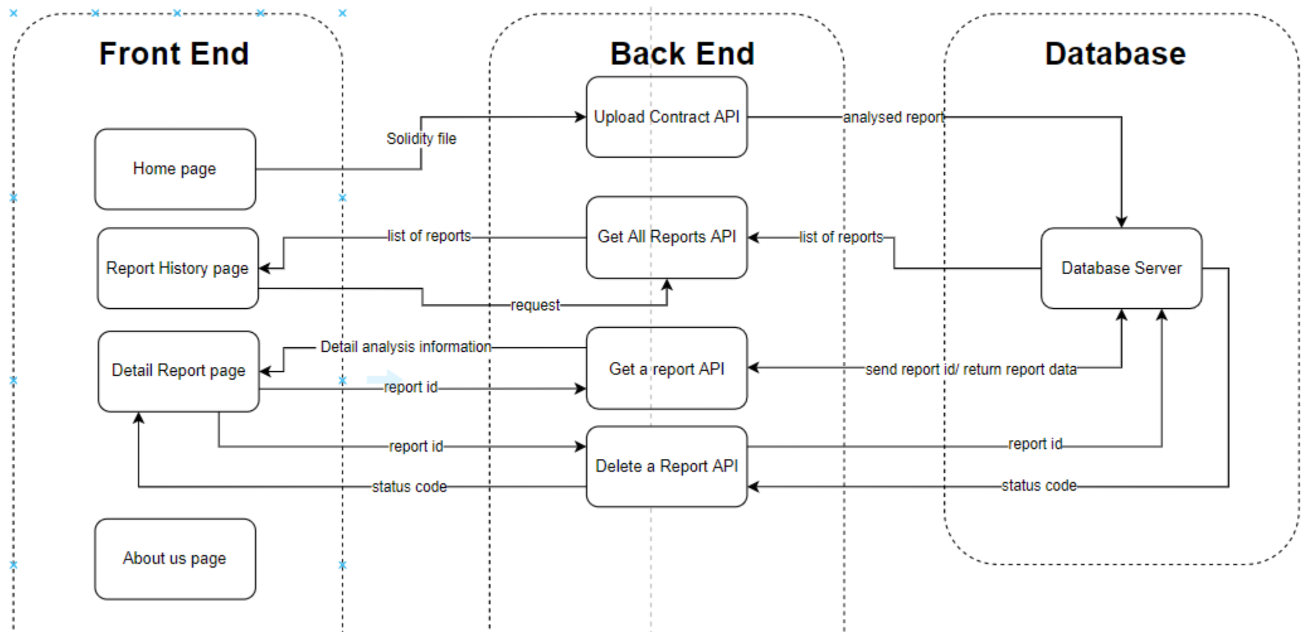


*Figure 4: Overall system architecture design.*

In the architecture above, the program is divided into 3 parts: frontend, backend, and database.

**1. Frontend:**

- The frontend component is responsible for the user interface and user interactions.
- It communicates with the backend through API endpoints to fetch or send data.
- For instance, when accessing the **Report History**, the frontend initiates a **GET** request to the backend's **upload_contract** API endpoint. This API performs the analysis of the submitted Solidity file, generates a comprehensive report, and subsequently uploads it to the database.

**2. Backend:**

- Acting as the bridge between the frontend and the database, the backend handles API requests and executes necessary logic.
- In response to requests, it interacts with the database to retrieve or store data.

**3. Database:**

- The database stores crucial information such as audit reports, vulnerabilities, and other relevant data.
- Accessed and manipulated by the backend, the database ensures persistent and organised storage of application data.

By separating the frontend, backend, and data access, we separated the concerns and structured the application in an organised method, allowing each component to function independently.

## *D. Front-end Prototype*

Below is our team's front-end low-fidelity prototype showing our web pages in different screen sizes. Please note that the prototype may differ from our actual implementation of the website:

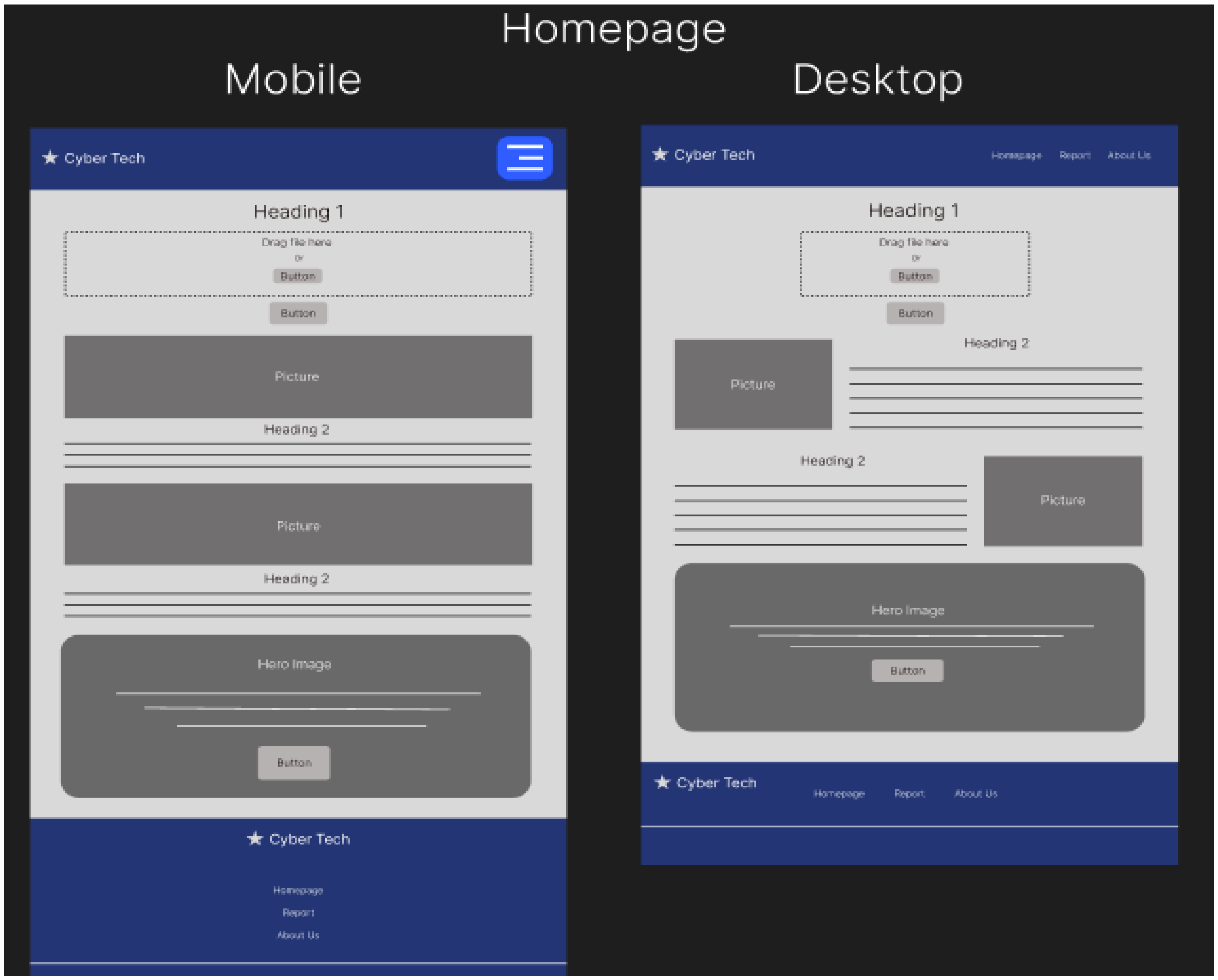


*Figure 5: Home page prototype.*

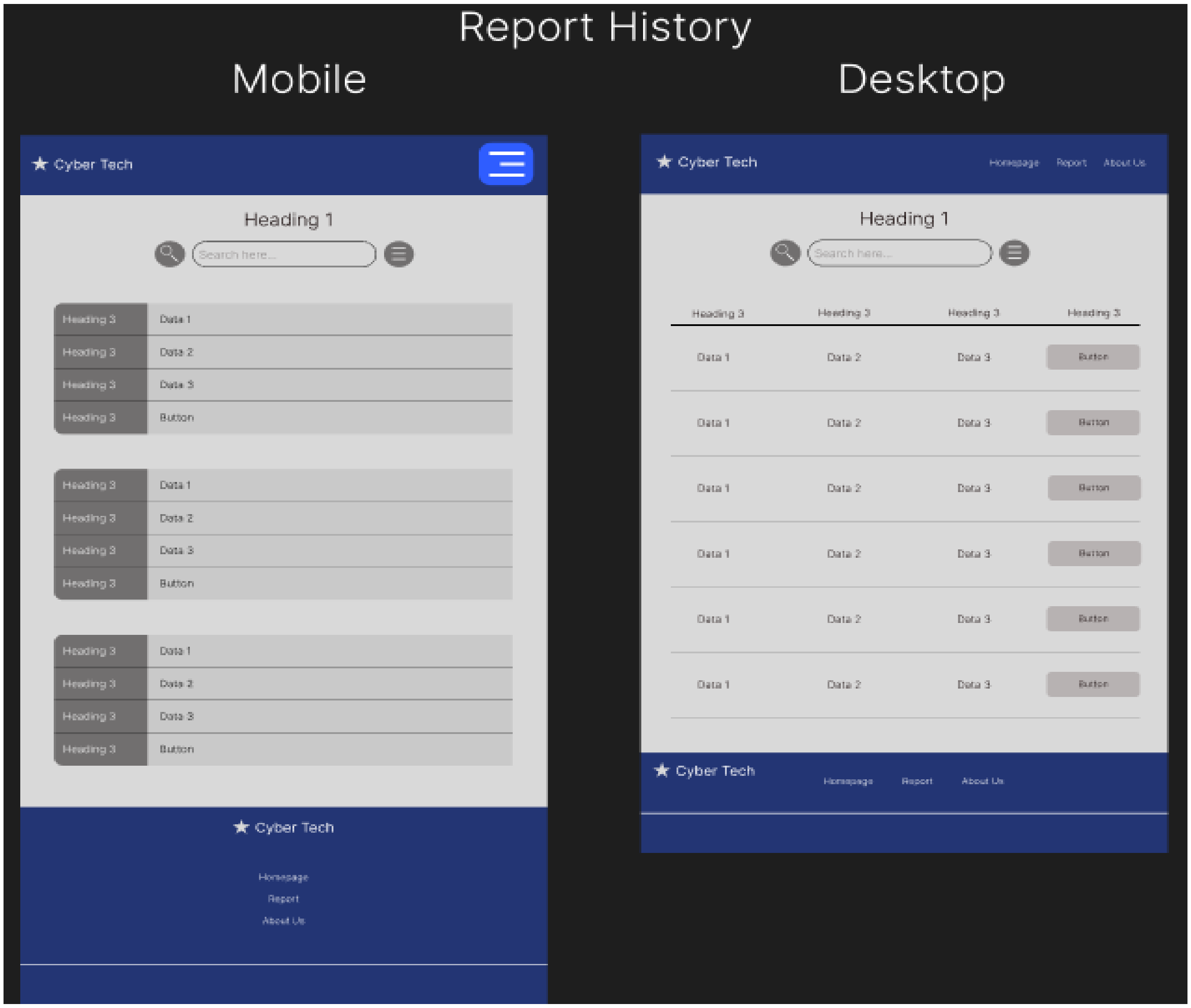


*Figure 6: Report History page prototype.*

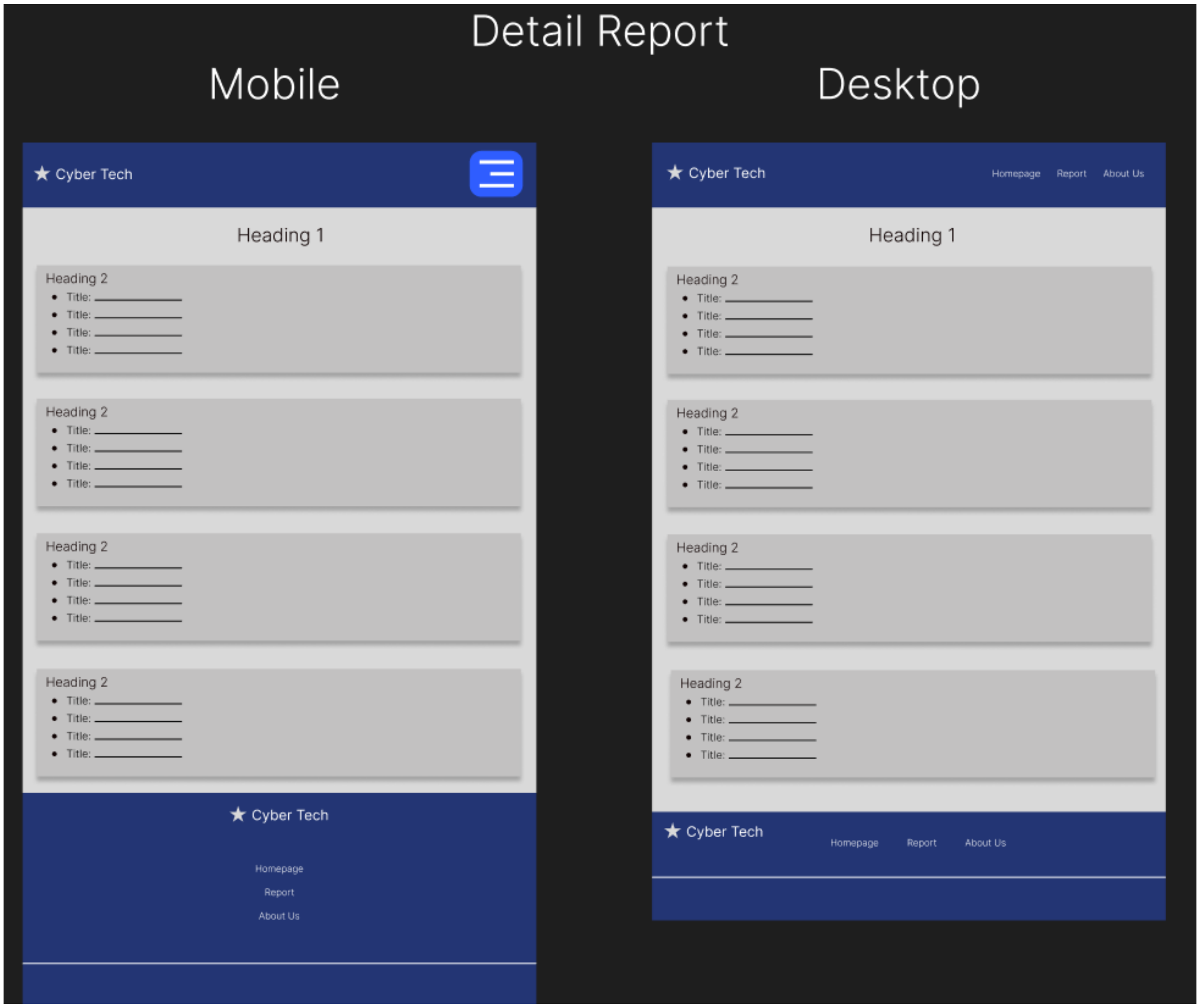


*Figure 7: Detail Report page prototype (corresponding to the Report page's link).*

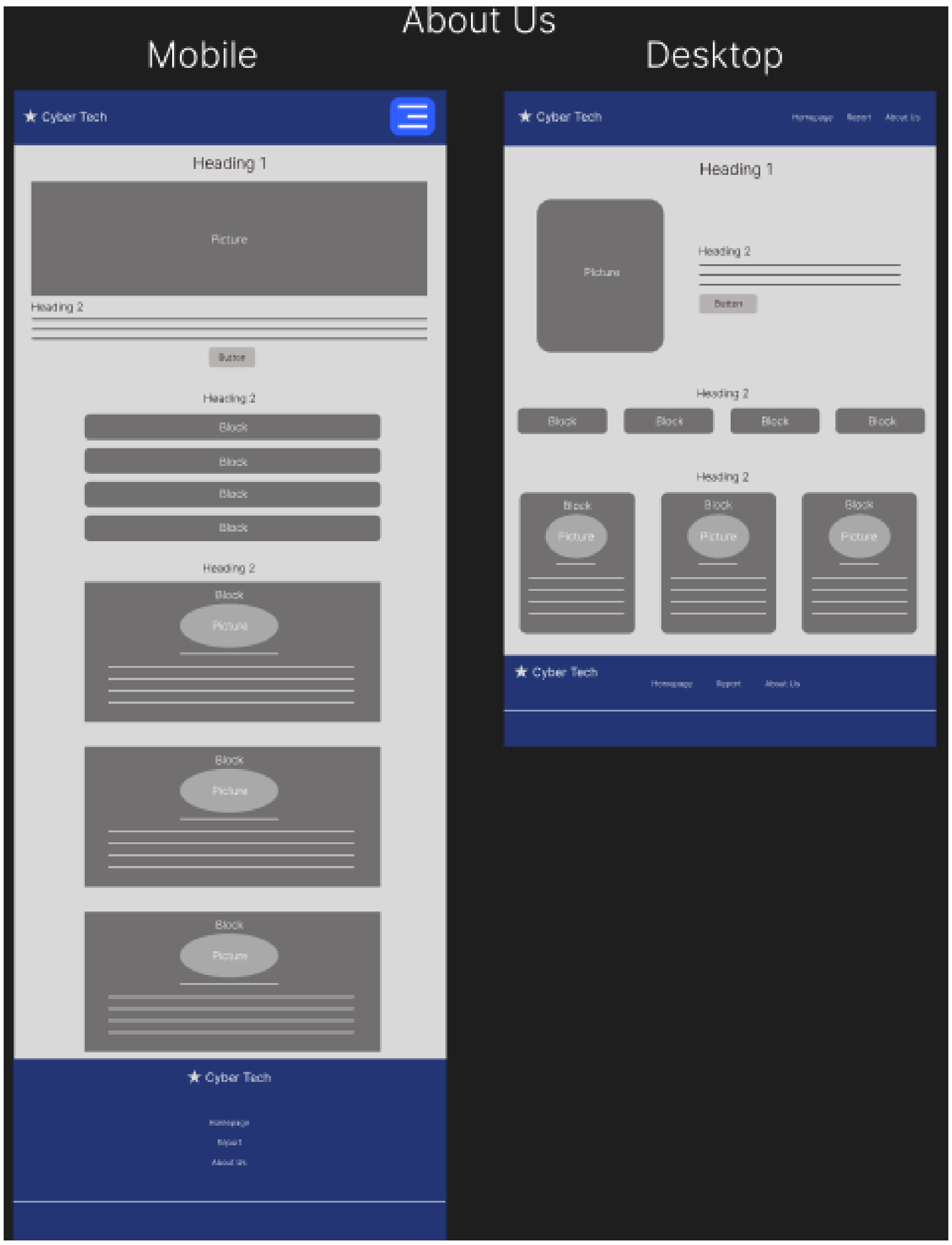
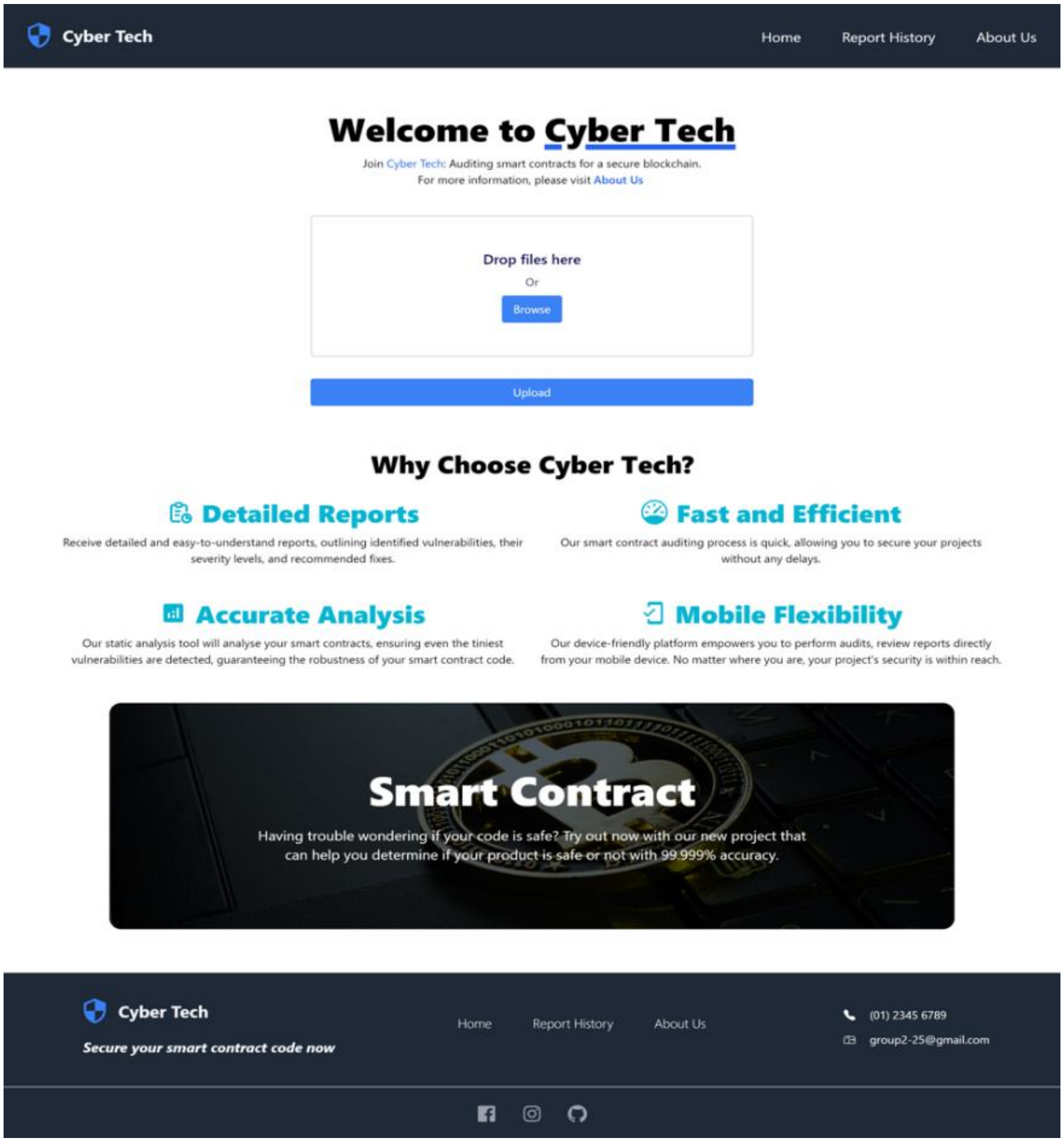


*Figure 8: About Us page prototype.*

## *E. Final Front-end Design*

Below are the screenshots of our finalised web pages:



Cyber Tech

Home Report History About Us

Welcome to Cyber Tech

Join Cyber Tech: Auditing smart contracts for a secure blockchain.
For more information, please visit About Us

Drop files here
Or
Browse

Upload

Why Choose Cyber Tech?

Detailed Reports

Receive detailed and easy-to-understand reports, outlining identified vulnerabilities, their severity levels, and recommended fixes.

Fast and Efficient

Our smart contract auditing process is quick, allowing you to secure your projects without any delays.

Accurate Analysis

Our static analysis tool will analyse your smart contracts, ensuring even the tiniest vulnerabilities are detected, guaranteeing the robustness of your smart contract code.

Mobile Flexibility

Our device-friendly platform empowers you to perform audits, review reports directly from your mobile device. No matter where you are, your project's security is within reach.

Smart Contract

Having trouble wondering if your code is safe? Try out now with our new project that can help you determine if your product is safe or not with 99.999% accuracy.

Cyber Tech

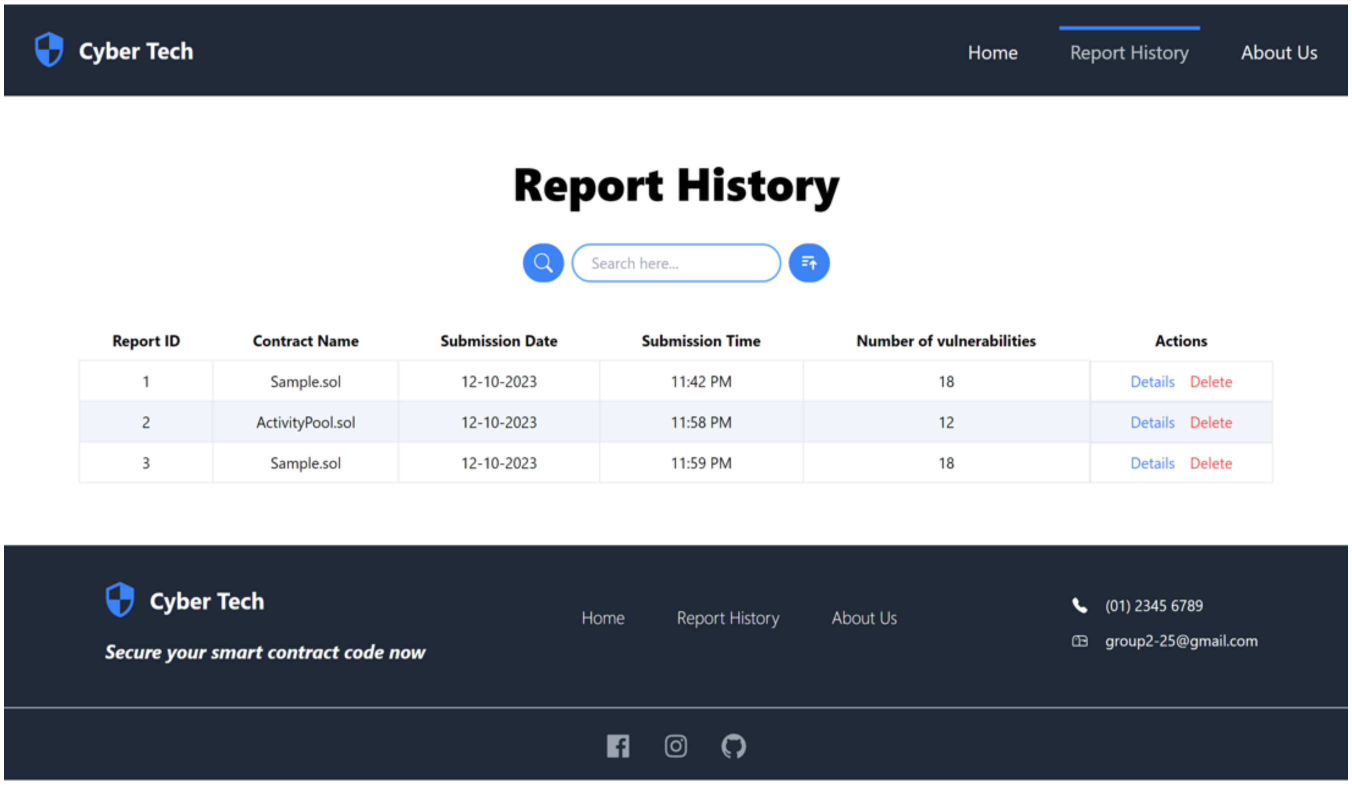
*Secure your smart contract code now*

Home Report History About Us

(01) 2345 6789

group2-25@gmail.com

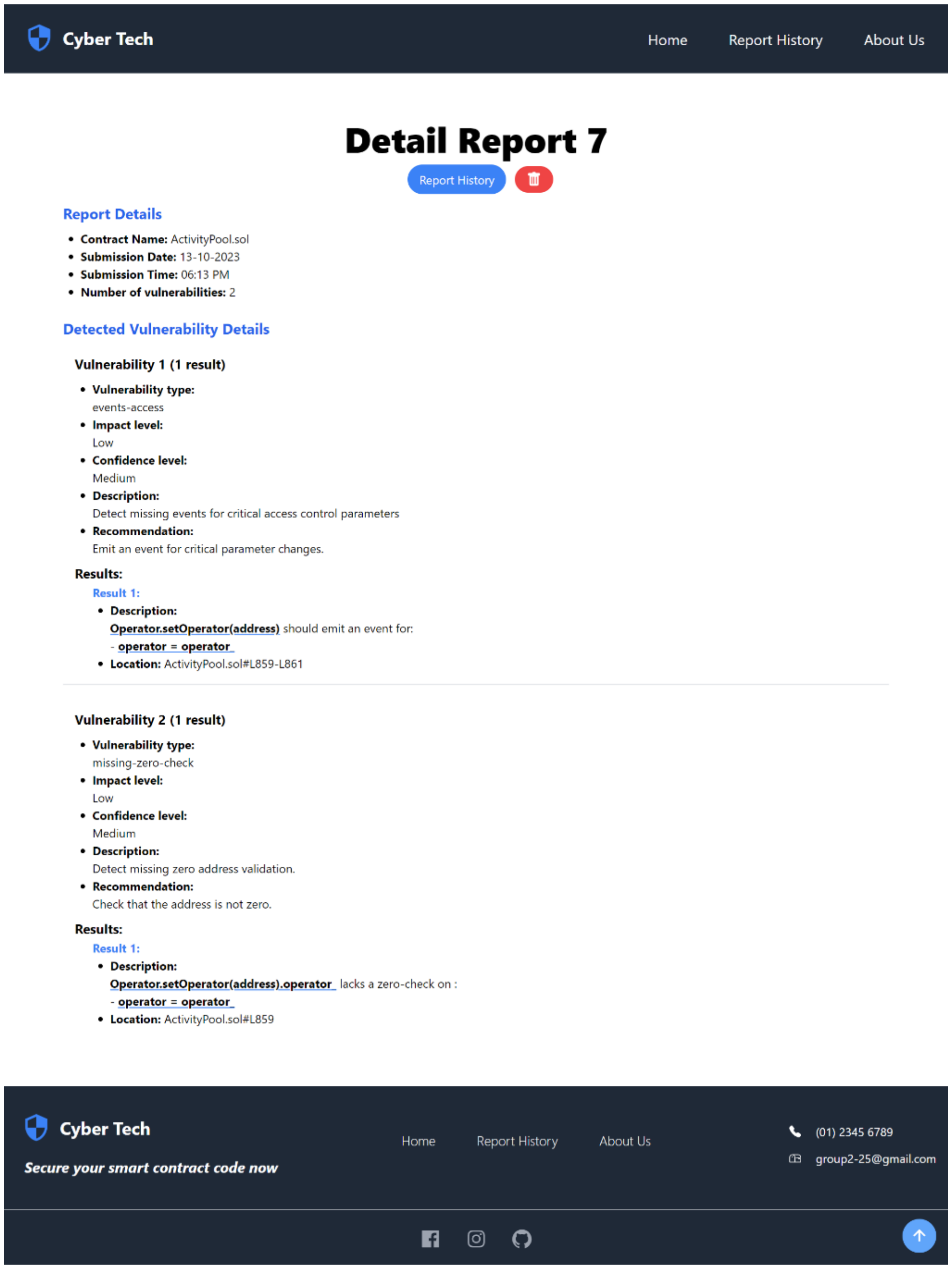
*Figure 9: Home page.*



Report History

| Report ID | Contract Name | Submission Date | Submission Time | Number of vulnerabilities | Actions |
|---|---|---|---|---|---|
| 1 | Sample.sol | 12-10-2023 | 11:42 PM | 18 | Details Delete |
| 2 | ActivityPool.sol | 12-10-2023 | 11:58 PM | 12 | Details Delete |
| 3 | Sample.sol | 12-10-2023 | 11:59 PM | 18 | Details Delete |

*Figure 10: Report History page (desktop & tablet view).*



Detail Report 7

Report Details

- **Contract Name:** ActivityPool.sol
- **Submission Date:** 13-10-2023
- **Submission Time:** 06:13 PM
- **Number of vulnerabilities:** 2

Detected Vulnerability Details

**Vulnerability 1 (1 result)**

- **Vulnerability type:**
  events-access
- **Impact level:**
  Low
- **Confidence level:**
  Medium
- **Description:**
  Detect missing events for critical access control parameters
- **Recommendation:**
  Emit an event for critical parameter changes.

**Results:**

Result 1:
- **Description:**
  Operator.setOperator(address) should emit an event for:
  - operator = operator
- **Location:** ActivityPool.sol#L859-L861

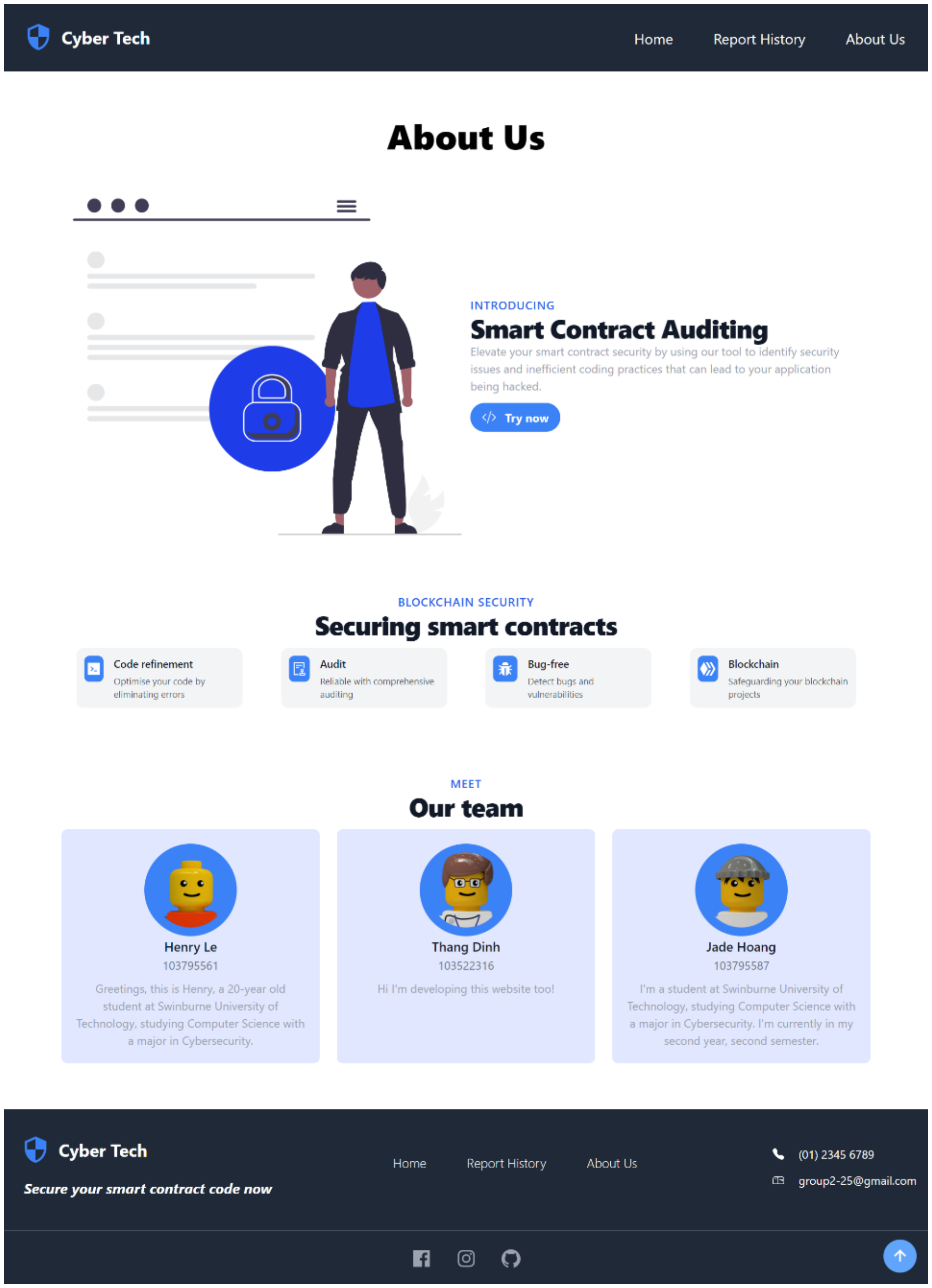
**Vulnerability 2 (1 result)**

- **Vulnerability type:**
  missing-zero-check
- **Impact level:**
  Low
- **Confidence level:**
  Medium
- **Description:**
  Detect missing zero address validation.
- **Recommendation:**
  Check that the address is not zero.

**Results:**

Result 1:
- **Description:**
  Operator.setOperator(address).operator lacks a zero-check on :
  - operator = operator
- **Location:** ActivityPool.sol#L859

*Figure 11: Detail Report page (corresponds to the Report History page's link).*



Cyber Tech

Home Report History About Us

About Us

INTRODUCING

Smart Contract Auditing

Elevate your smart contract security by using our tool to identify security issues and inefficient coding practices that can lead to your application being hacked.

Try now

BLOCKCHAIN SECURITY

Securing smart contracts

Code refinement — Optimise your code by eliminating errors

Audit — Reliable with comprehensive auditing

Bug-free — Detect bugs and vulnerabilities

Blockchain — Safeguarding your blockchain projects

MEET

Our team

Henry Le
103795561
Greetings, this is Henry, a 20-year old student at Swinburne University of Technology, studying Computer Science with a major in Cybersecurity.

Thang Dinh
103522316
Hi I'm developing this website too!

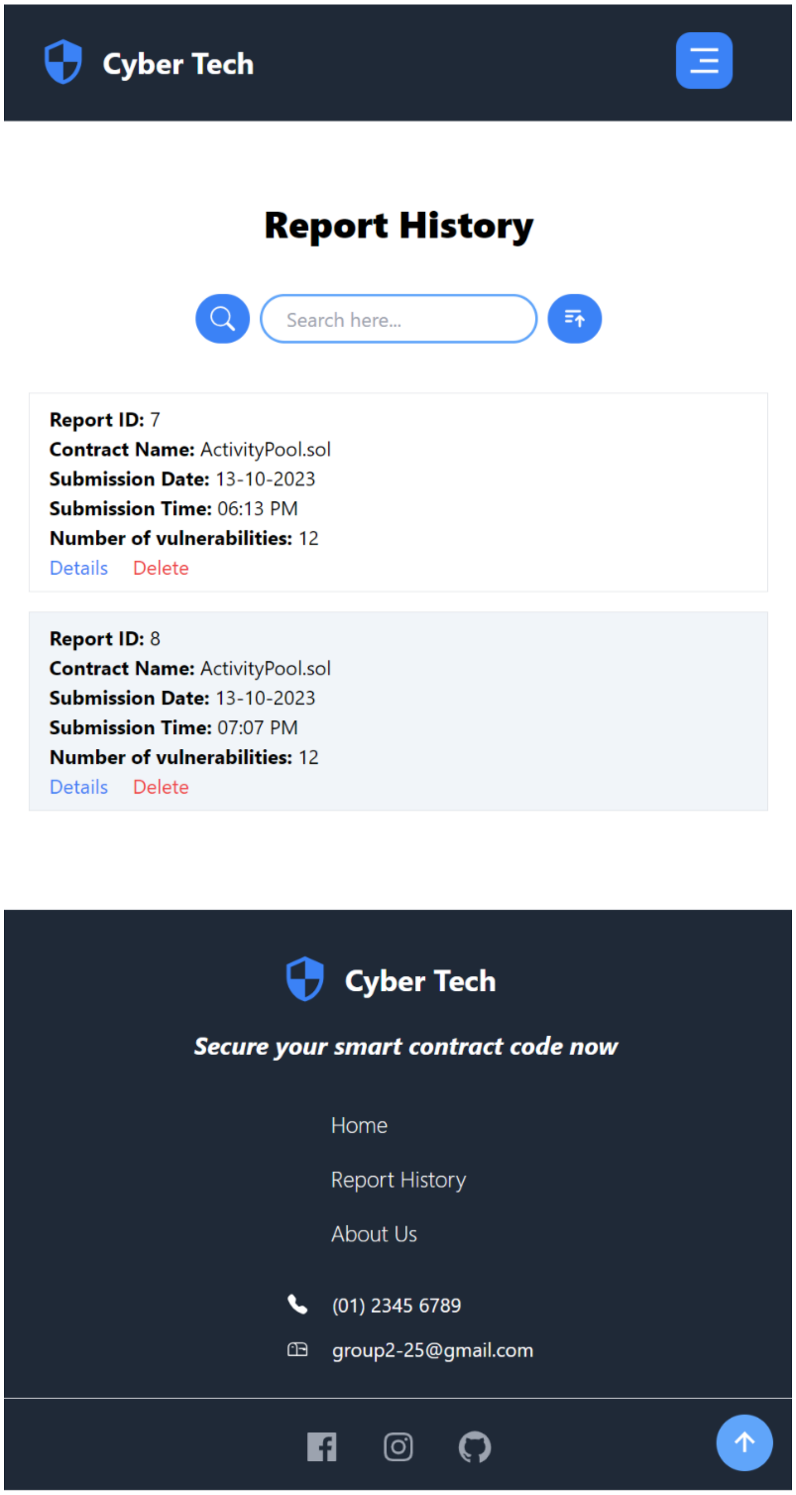
Jade Hoang
103795587
I'm a student at Swinburne University of Technology, studying Computer Science with a major in Cybersecurity. I'm currently in my second year, second semester.

Cyber Tech
*Secure your smart contract code now*

Home Report History About Us

(01) 2345 6789
group2-25@gmail.com

*Figure 12: About Us page.*



Report History

**Report ID:** 7
**Contract Name:** ActivityPool.sol
**Submission Date:** 13-10-2023
**Submission Time:** 06:13 PM
**Number of vulnerabilities:** 12
Details Delete

**Report ID:** 8
**Contract Name:** ActivityPool.sol
**Submission Date:** 13-10-2023
**Submission Time:** 07:07 PM
**Number of vulnerabilities:** 12
Details Delete

*Figure 13: Responsive mobile view (an example in the Report History page).*

## *F. Backend Database Design*

In the backend, the **Smart Contract Audit System** utilises a relational database to manage essential data related to audit reports and identified vulnerabilities. It consists of three tables: **Vulnerabilities**, **Reports**, and **Results**.

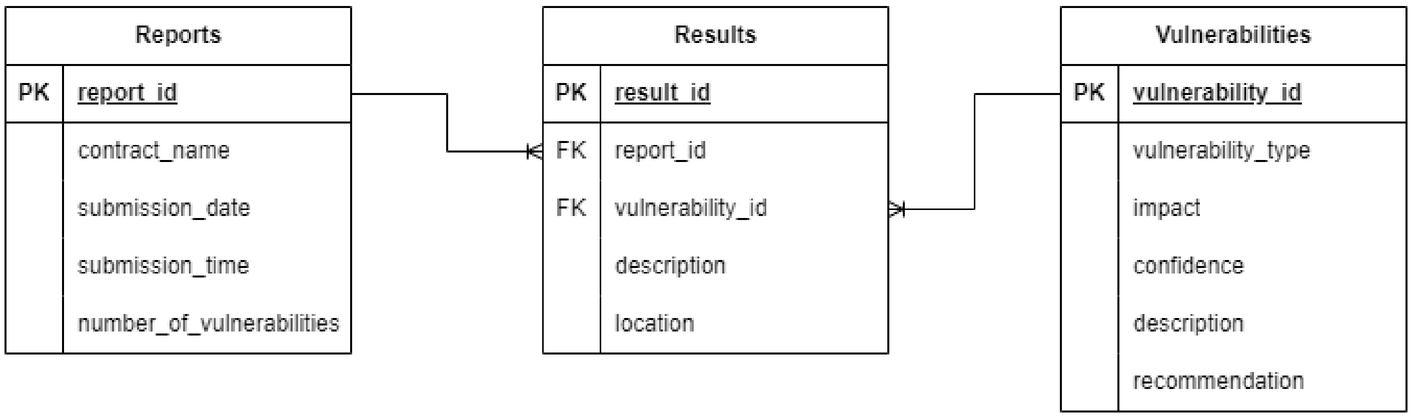


*Figure 14: Relational database diagram.*

Below are the detailed specifications for each table:

**<u>Reports table:</u>**

This table holds information about audit reports, including the audited smart contract's name, submission date, submission time and number of vulnerabilities that exist in each contract.

| Field name | Data type | Description |
|---|---|---|
| report_id (PK) | AUTO-INCREMENTED INTEGER | Unique ID for each report. |
| contract_name | STRING | The name of the audited smart contract. |
| submission_date | DATE | The date when the audit was performed. |
| submission_time | TIME | The time when the audit was performed. |
| number_of_vulnerabilities | INTEGER | Display the number of vulnerabilities found in each contract. |

*Table 4: Reports table details.*

**<u>Vulnerabilities table:</u>**

This table stores detailed information about identified vulnerabilities, including their names, impact levels, confidence levels, descriptions, and recommended solutions.

| Field name | Data type | Description |
|---|---|---|
| vulnerability_id (PK) | AUTO-INCREMENTED INTEGER | Unique ID for each vulnerability. |
| vulnerability_type | STRING | The name of the vulnerability type/category. |
| impact | STRING | The impact or severity level of the vulnerability (e.g., High, Medium, Low). |
| confidence | STRING | Level of confidence or certainty regarding the identified vulnerability (e.g., High, Medium, Low). |
| description | TEXT | Description of the vulnerability to explain what it is and why it matters. |
| recommendation | TEXT | Recommendation to resolve the vulnerability. |

*Table 5: Vulnerabilities table details.*

**Results table:**

This table establishes a link between reports and vulnerabilities, specifying the location and providing descriptions of vulnerabilities within the context of each report.

| Field name | Data type | Description |
|---|---|---|
| result_id (PK) | AUTO-INCREMENTED INTEGER | Unique ID for each result. |
| report_id (FK) | INTEGER | Foreign key linking to the Reports table, indicating which report this result belongs to. |
| vulnerability_id (FK) | INTEGER | Foreign key linking to the Vulnerabilities table, indicating which vulnerability this result belongs to. |
| description | TEXT | A specific description of the vulnerability within the code of the audited smart contract uploaded by the user. This field captures details such as method names, and parameter details of the identified vulnerability (e.g., “LuckyDraw.rewardID should be constant”) |
| location | TEXT | Location in the uploaded smart contract code where the vulnerability is found. It includes the smart contract file name and the line range where the vulnerability exists. (e.g., “ActivityPool.sol#L1350-1398”) |

*Table 6: Results table details.*

**Schema relationships:**

In our database structure, **Vulnerabilities** and **Reports** share a many-to-many relationship, with **Results** as a junction table. This design allows one report to have many vulnerabilities and one vulnerability can be in many reports. The advantage of this approach is in minimising redundancy of vulnerability details, such as impact and recommendation fields, by storing them in the **Vulnerabilities** table. This design ensures that specific details about identified vulnerabilities of a report (such as description and location) are stored in the **Results** table, providing a clear and structured representation of the relationships between reports, vulnerabilities, and their specific details.

## *G. API Design*

The **Smart Contract Audit System API** provides endpoints for interacting with the Smart Contract Audit System. It is designed to handle requests related to uploading, retrieving, and managing audit reports. The API internally interacts with a relational database to store and retrieve data. The API follows RESTful principles for clear and structured endpoints.

The table below summarises the four API endpoints along with their methods and descriptions:

| Endpoint | Method | Description |
|---|---|---|
| **/upload_contract** | POST | Upload a smart contract for analysis. |
| **/reports** | GET | Retrieves a list of all audit reports. |
| **/reports/{report_id}** | GET | Retrieves detailed information about a specific audit report by its ID. |
| **/reports/{report_id}** | DELETE | Deletes a specific audit report by its ID. |

*Table 7: API endpoints summary.*

**API specifications:**

This section provides more details for each API endpoint. Please note that while the API interacts with a database to ensure data persistence, detailed implementation aspects, such as how the database session is managed, are considered part of the backend's internal architecture, and are not exposed in this API documentation. The focus here is on specifying the required request parameters, response formats, and expected behaviours for clients interacting with the API.

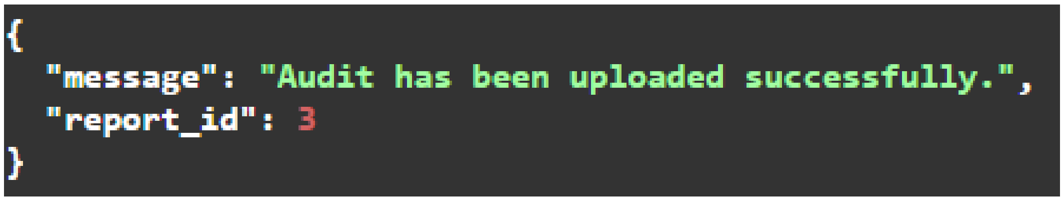
The base URL of the FastAPI server is http://127.0.0.1:8000/:

**1. Create Report:**

- *Endpoint*: **/upload_contract**
- *Method*: **POST**
- *Description*:
  - Uploads a Smart Contract file for analysis and initiates the audit report creation process.
- *Process*: Steps involved in creating a report:
  a. Validates if a file is provided and ensures it is a .sol file.
  b. Saves the uploaded file to the server.
  c. Extracts Solidity version from the uploaded file (to run **solc-select install <version>** and **solc-select use <version>** commands).

d. Analyse the contract and generate a report in Markdown **(.md)** format.
e. Filter the .md report to extract relevant information.
f. Prepares report data and uploads it to the database.

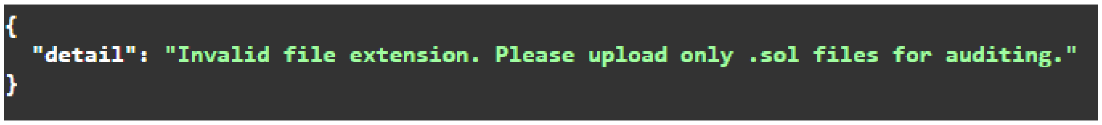
- *Request Parameters:*
    - contract (Upload File): The Smart Contract file is to be uploaded.
- *Response Format:*
    - **Status code 201 CREATED:** on successful upload. The response is formatted in JSON, providing crucial information for further actions. The response includes:
        - Message: An indicator of the success or failure of the operation. This message serves as feedback for the user or any system interacting with the API.
        - Report ID: The unique identifier assigned to the newly created report. This Report ID is essential for the React frontend to facilitate user redirection to the detailed report corresponding to this ID.



```
{
  "message": "Audit has been uploaded successfully.",
  "report_id": 3
}
```

*Figure 15: Response body for code 201 CREATED.*

- *Exceptions*:
    - HTTP Exception with status code **400 BAD REQUEST** if no file is provided.
    - HTTP Exception with status code **400 BAD REQUEST** if the file has an invalid extension.
    - HTTP Exception with status code **400 BAD REQUEST** if no Solidity version is found in the uploaded file.
    - HTTP Exception with status code **500 INTERNAL SERVER ERROR** for general server-side errors such as interruptions during the analysis.



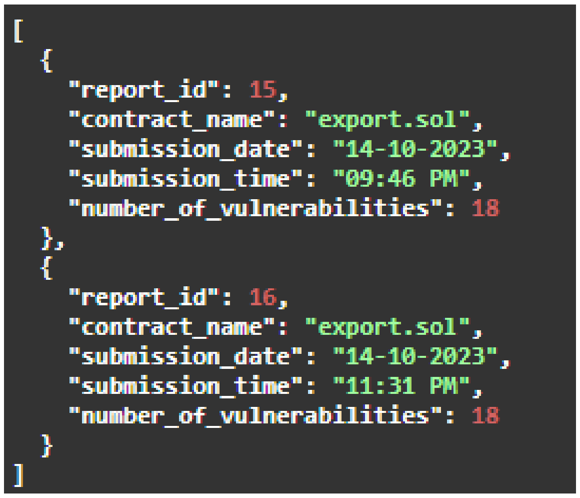
```
{
  "detail": "Invalid file extension. Please upload only .sol files for auditing."
}
```

*Figure 16: Response body for code 400 BAD REQUEST.*

**2. Get All Reports**

- *Endpoint*: **/reports**
- *Method*: **GET**
- *Description*:
    - Retrieves a list of all audit reports from the database.
- *Request Parameters:*
    - No parameters.
- *Response Format:*
    - **Status code 200 OK** on successful retrieval.
    - The response is a list of Report objects containing relevant information. Each object in the list represents a distinct audit report and includes the following details:

        - Report ID: A unique identifier for the audit report.
        - Contract name: The name of the smart contract associated with the audit.
        - Submission date: The date when the audit report was submitted, in the format DD-MM-YYYY.
        - Submission time: The time at which the audit report was submitted, in the format HH:MM AM/PM
        - Number of vulnerabilities detected: Indicates the count of vulnerabilities found in the smart contract.
    - The example response below has 2 reports returned with IDs 15 and 16, respectively.



```
[
  {
    "report_id": 15,
    "contract_name": "export.sol",
    "submission_date": "14-10-2023",
    "submission_time": "09:46 PM",
    "number_of_vulnerabilities": 18
  },
  {
    "report_id": 16,
    "contract_name": "export.sol",
    "submission_date": "14-10-2023",
    "submission_time": "11:31 PM",
    "number_of_vulnerabilities": 18
  }
]
```

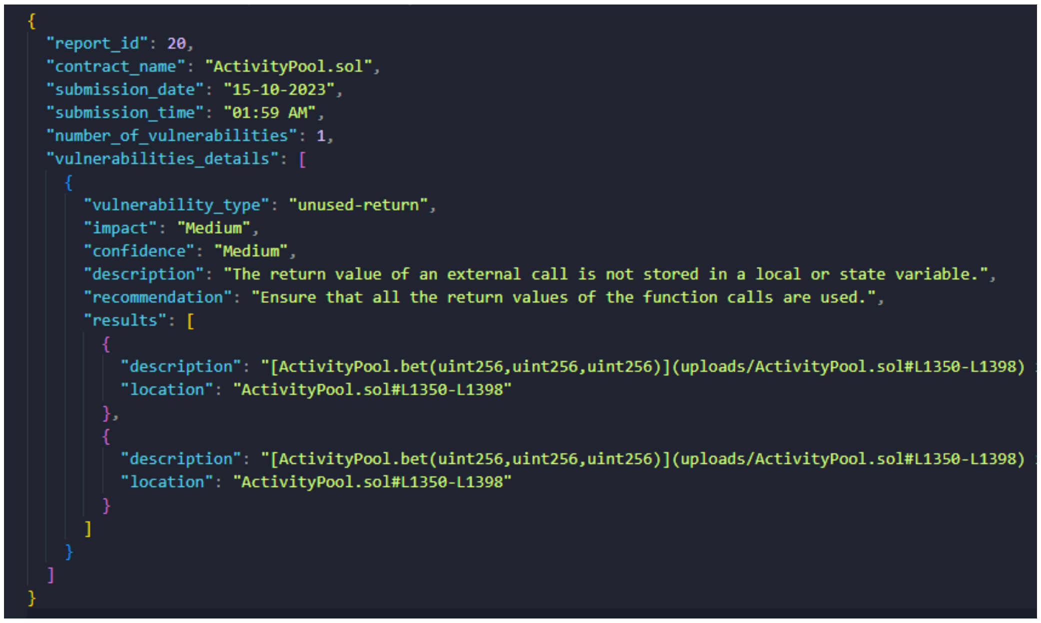
*Figure 17: Response body of Get All Reports endpoint.*

- *Exceptions*:
    - HTTP Exception with status code **500 INTERNAL SERVER ERROR** for general server-side errors.
    - HTTP Exception with status code **404 NOT FOUND** if no reports are found in the database.

**3. Get Specific Report by ID:**

- *Endpoint*: **/reports/{report_id}**
- *Method*: **GET**
- *Description*:
    - Retrieves a specific audit report based on the provided report ID.
- *Requests Parameters*:
    - report_id (int): an integer specifying the ID of the report to be retrieved.
- *Response Format*:
    - **Status code 200 OK**: on successful retrieval.
    - The response returns a JSON object containing the following fields:
        - Report ID
        - Contract name
        - Submission date
        - Submission time
        - Number of vulnerabilities

  - Vulnerabilities details (JSON list): provides information about each vulnerability, including:
    - Vulnerability type
    - Impact
    - Confidence
    - Description
    - Recommendation
    - Results (JSON list): contains Result objects within each vulnerability, each providing:
      - Description
      - Location



```
{
  "report_id": 20,
  "contract_name": "ActivityPool.sol",
  "submission_date": "15-10-2023",
  "submission_time": "01:59 AM",
  "number_of_vulnerabilities": 1,
  "vulnerabilities_details": [
    {
      "vulnerability_type": "unused-return",
      "impact": "Medium",
      "confidence": "Medium",
      "description": "The return value of an external call is not stored in a local or state variable.",
      "recommendation": "Ensure that all the return values of the function calls are used.",
      "results": [
        {
          "description": "[ActivityPool.bet(uint256,uint256,uint256)](uploads/ActivityPool.sol#L1350-L1398)
          "location": "ActivityPool.sol#L1350-L1398"
        },
        {
          "description": "[ActivityPool.bet(uint256,uint256,uint256)](uploads/ActivityPool.sol#L1350-L1398)
          "location": "ActivityPool.sol#L1350-L1398"
        }
      ]
    }
  ]
}
```

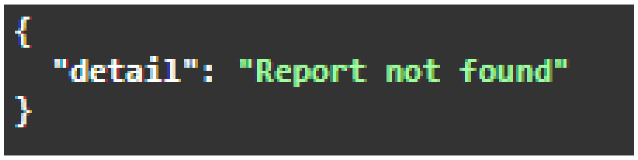
*Figure 18: Response body of code 200 OK.*

- *Exceptions*:
  - HTTP Exception with status code **404 NOT FOUND** if the specified report ID is not found.
  - HTTP Exception with status code **500 INTERNAL SERVER ERROR** for general server-side errors.

**4. Delete Specific Report by ID:**

- *Endpoint*: **/reports/{report_id}**
- *Method*: **DELETE**
- *Description*:
  - Delete a specific audit report by its ID and return a response with a status code of 204 NO_CONTENT, indicating a successful deletion.
- *Requests Parameters*:
  - Report_id: integer ID of the report to be deleted.
- *Response Format*:
  - **Status code 204 NO CONTENT**: on successful deletion, meaning the report has been deleted from the database. Returns None on successful deletion.

- *Exceptions*:
    - HTTP Exception with status code **404 NOT FOUND** if the specified report ID is not found.
    - HTTP Exception with status code **500 INTERNAL SERVER ERROR** for general server-side errors.



```
{
  "detail": "Report not found"
}
```

*Figure 19: Response body of code 404 NOT FOUND.*

**Notes:**

- Error handling is implemented to provide meaningful and user-friendly error responses.
- The API supports Cross-Origin Resource Sharing (CORS) to allow access from a React application running on http://localhost:3000.

## *H. Functionality Description*

With the core functional requirements in mind, our system has seven key functionality requirements that allow users to navigate and interact with the smart contract auditing platform. From uploading and analysing contracts to managing historical reports and exploring team information, each feature contributes to a comprehensive and user-friendly experience.

All of the use cases explained below assume that the user starts at the **Homepage** of the website.

**1. Upload and audit a smart contract**

- *Purpose*: Allow the user to upload a Solidity (.sol) file for analysis and receive a report on the vulnerabilities of the contract and recommendations to improve them. This process is user-friendly, allowing for easy contract submission through drag-and-drop or file browsing method, with automatic redirection to a detailed report page post-analysis.
- *Use cases*:
    1. Users can either drag and drop the Solidity contract into the designated input area or utilise the **"Browse"** button to select the contract file.
    2. Clicking on the **"Upload"** button starts the upload and contract analysis process.
    3. The system validates whether the uploaded file is a valid smart contract. If validation fails, error messages in red are displayed below the **"Upload"** button. Error messages may include details about invalid file formats or other issues. Additionally, the system can show a server connection error message if applicable.
    4. Upon successful audit analysis, users are automatically directed to a page providing a detailed report on the vulnerabilities found within the Solidity contract.
    5. Simultaneously, the system displays a prominent message at the top of the page, notifying the user of the successful completion of the auditing process.

**2. Browse the list of previous reports of the audited smart contract**

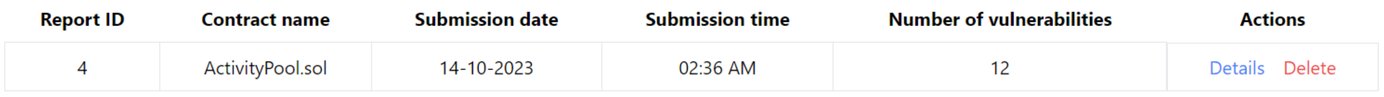
- *Purpose*: The primary objective of this functionality is to create an organised repository of all previously audited smart contract reports. Users can access this repository to retrieve information on historical contract audits. Whether on large-screen or small-screen devices, users can navigate to the

**"Report History"** section to view a comprehensive list of all submitted reports, ensuring easy access to valuable information regarding past smart contract assessments.

- *Use cases (large-screen devices)*:
    1. On the navigation bar, click on the **“Report History”** button to be directed to a page listing all the reports created from the uploaded Solidity contract files.
- *Use cases (mobile or small-screen devices)*:
    1. Open the responsive navigation bar by clicking on the **"hamburger menu" icon.**
    2. Within the pop-up menu, click on the **"Report History"** link. This action redirects the user to the list of reports page, providing them with the same comprehensive view of all submitted reports as available on larger screens.

### 3. View details of a report

- *Purpose*: This functionality is designed to provide users with in-depth insights into the analysis and recommendations associated with a specific smart contract audit report.
- *Use cases (large-screen devices)*:
    1. On the navigation bar, click on the **“Report History”** button to be directed to a page listing all the reports created from the uploaded Solidity contract files.
    2. On the list of reports page, users identify the specific report of interest. Clicking on the **"Details"** link in the **Action column**, as illustrated in Figure 20.
    3. The user gets redirected to a page that comprehensively details the analysis of the chosen report.
- *Use cases (mobile or small-screen devices)*:
    1. Open the responsive navigation bar by clicking on the **"hamburger menu" icon.**
    2. Within the pop-up menu, click on the **“Report History”** link, and the user will be redirected to the list of reports page.
    3. On the list of reports page, users locate the specific report they wish to explore in detail.
    4. Clicking on the **"Details"** link associated with the chosen report to be redirected to a dedicated page providing an in-depth analysis of the selected report.



| Report ID | Contract name | Submission date | Submission time | Number of vulnerabilities | Actions |
|---|---|---|---|---|---|
| 4 | ActivityPool.sol | 14-10-2023 | 02:36 AM | 12 | Details Delete |

*Figure 20: A report entry in the list of all reports.*

### 4. Search for a report name

- *Purpose*: This functionality enhances the user experience by offering a quick and efficient means of locating a specific smart contract audit report. Users can enter the desired report name, date, time, ID, or several vulnerabilities in the search bar, whether on large screens or mobile devices, within the **"Report History"** page. This search capability also allows users to promptly retrieve and review targeted reports without the need for manual navigation through an extensive list.
- *Use cases (large-screen devices):*
    1. On the navigation bar, click on the **“Report History”** button to be directed to a page listing all the reports created from the uploaded Solidity contract files.

2. On the list of reports page, users identify the **"Search here..." text box**. Clicking on this box allows them to input specific keywords such as report name, date, time, ID, or vulnerabilities, as shown in Figure 21. The search is case-insensitive.
3. As users type in the search criteria, the system dynamically responds by presenting search results in real time.

- *Use cases (mobile or small-screen devices):*
  1. Open the responsive navigation bar by clicking on the **"hamburger menu" icon.**
  2. Click on the **"Report History"** link, and the user will be directed to the list of reports page.
  3. Click on the **"Search here..." box** and the search results will appear instantly.

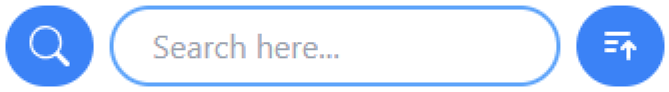


*Figure 21: A search bar on the Report History page.*

**5. Sort the list of all reports**

- *Purpose:* Allow users to organise the list of reports according to specific criteria, this functionality enhances user flexibility and efficiency. Sorting by date, contract name or the number of vulnerabilities allows users to prioritise and view the information that is most relevant to their current needs. By default, the reports are ordered by submission date and time in descending order, which means that the user can quickly access the most recent reports at the top of the list.
- *Use cases (large screen devices):*
  1. On the top menu bar, click on the **"Report History"** button to be directed to a page listing all the reports created from the uploaded solidity contract files.
  2. Click on the right icon next to the **"Search here..." box**, it will display the sorting options so that the users can customise the order of the reports based on their preferences, as shown in Figure 22.
- *Use cases (mobile or small screen devices):*
  1. Open the responsive navigation bar by clicking on the **"hamburger menu" icon.**
  2. Click on the **"Report History"** link, and the user will be redirected to the list of reports page.
  3. Click on the sorting icon located next to the search box field.
  4. Choose specific criteria for sorting, such as date, contract name, or the number of vulnerabilities. Additionally, users can specify whether the order should be ascending or descending.
  5. As users make their sorting selections, the ordered list results appear instantly.

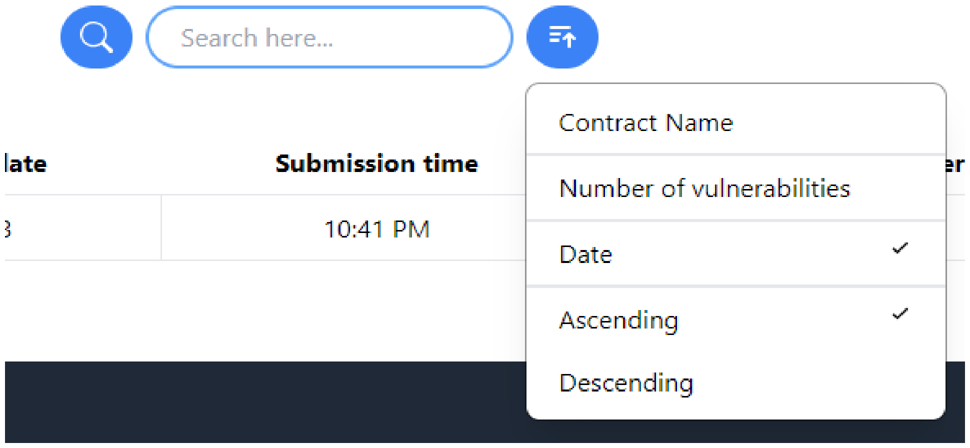


*Figure 22: Sorting option in the Report History page.*

**6. Delete a report**

- *Purpose*: This functionality grants users control over their data by allowing them to remove specific smart contract audit reports when they are no longer needed.
- *Use cases (large-screen devices):*
    1. On the top menu bar, click on the **"Report History"** button to be directed to a page listing all the reports created from the uploaded Solidity contract files.
    2. On the list of reports page, users can identify the specific report they wish to remove. Clicking on the **"Delete"** option associated with that report, as shown in Figure 23.

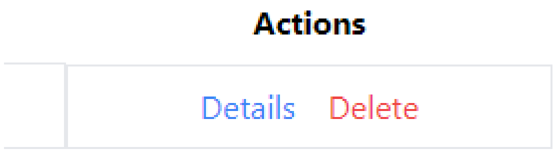


*Figure 23: Delete option in the Report History page (desktop and tablet view).*

3. Alternatively, users can access the Detail Report page by clicking on the **"Details"** link. Once on the **Detail Report page**, users can click on the trash icon (Figure 24) to delete the report from there.

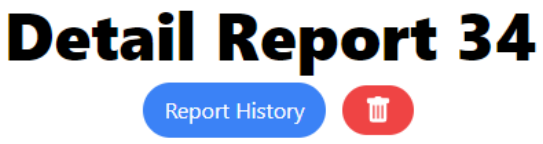
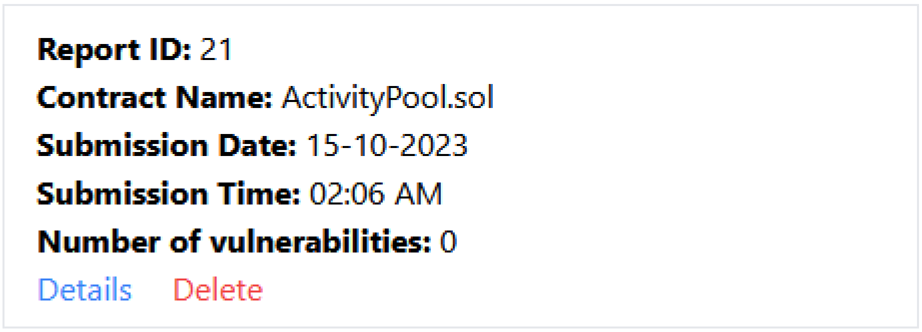


*Figure 24: Delete option on the Detail Report page (all screen sizes view).*

- *Use cases (mobile or small-screen devices):*
    1. Open the responsive navigation bar by clicking on the **"hamburger menu" icon**.
    2. Click on the **"Report History**" link, and the user will be redirected to the list of reports page.

3. On the list of reports page, click on the **“Delete” option** of a particular report, as illustrated in Figure 25.



**Report ID:** 21
**Contract Name:** ActivityPool.sol
**Submission Date:** 15-10-2023
**Submission Time:** 02:06 AM
**Number of vulnerabilities:** 0

*Figure 25: Delete option of a specific report on mobile view.*

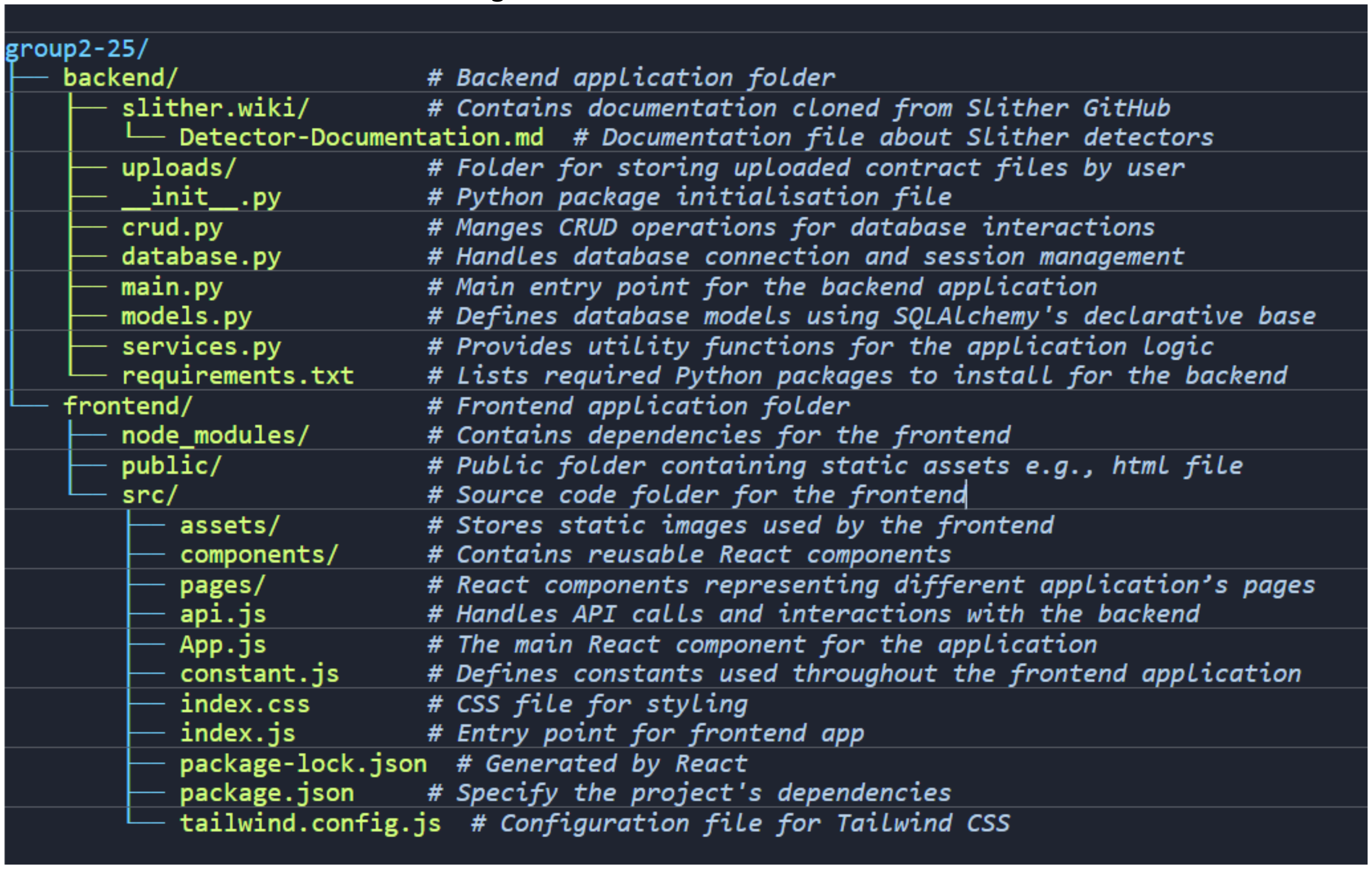
4. Alternatively, click on **“Details”** to be redirected to the **Detail Report page** and then click on the **trash icon**, as shown in Figure 24.

**7. View the team’s info**

- *Purpose:* This functionality allows users to access information about the team responsible for the smart contract audit platform, providing transparency and building user trust.
- *Use cases (large screen devices):*
  **1.** On the top menu bar, click on the **“About us” link.**
  2. The user gets directed to the **“About Us” page** of the team.
- *Use cases (mobile or small screen devices):*
  1. Open the responsive navigation bar by clicking on the **"hamburger menu" icon**.
  2. Within the navigation bar, click on the **”About Us”** link
  3. The user gets directed to the **“About us” page** of the team. Here, they gain insights into the team's background, and the values of the audit platform.

## I. Directory Structure Overview

Below is an overview of our team project's directory structure, outlining the purpose of each file and folder in both the backend and frontend components. This project is a full-stack web application, utilising FastAPI for the backend and React for the front end. The backend manages database interactions, while the frontend communicates with the backend through API calls.



```
group2-25/
├── backend/                  # Backend application folder
│   ├── slither.wiki/         # Contains documentation cloned from Slither GitHub
│   │   └── Detector-Documentation.md  # Documentation file about Slither detectors
│   ├── uploads/              # Folder for storing uploaded contract files by user
│   ├── __init__.py           # Python package initialisation file
│   ├── crud.py               # Manges CRUD operations for database interactions
│   ├── database.py           # Handles database connection and session management
│   ├── main.py               # Main entry point for the backend application
│   ├── models.py             # Defines database models using SQLAlchemy's declarative base
│   ├── services.py           # Provides utility functions for the application logic
│   └── requirements.txt      # Lists required Python packages to install for the backend
└── frontend/                 # Frontend application folder
    ├── node_modules/         # Contains dependencies for the frontend
    ├── public/               # Public folder containing static assets e.g., html file
    └── src/                  # Source code folder for the frontend
        ├── assets/           # Stores static images used by the frontend
        ├── components/       # Contains reusable React components
        ├── pages/            # React components representing different application's pages
        ├── api.js            # Handles API calls and interactions with the backend
        ├── App.js            # The main React component for the application
        ├── constant.js       # Defines constants used throughout the frontend application
        ├── index.css         # CSS file for styling
        ├── index.js          # Entry point for frontend app
        ├── package-lock.json  # Generated by React
        ├── package.json      # Specify the project's dependencies
        └── tailwind.config.js  # Configuration file for Tailwind CSS
```

*Figure 26: Project directory structure overview.*

## *J. Project Deployment Instructions*

**Prerequisites:**

Before deploying the Smart Contract Audit System, ensure that the following prerequisites are met:

**1. Frontend (React):**

- **Node.js (at least version 14.0.0):** Required for managing frontend dependencies using npm and running the React application.
- **npm (Node Package Manager):** Included with Node.js.

**2. Backend (FastAPI):**

- **Python 3 (preferably version 3.11.5)**: Required for setting up and running the FastAPI backend using pip.
- **pip (Python Package Installer)**: Included with Python 3.
- **venv module (optional but recommended):** Virtual environment to create an isolated environment for Python projects. This module is built into Python so no further installation is required.

3**. Relational Database:** You need a **MySQL server** installed on your machine **(preferably version 8.0.34).**

Note:

- The latest version of Python 3 (i.e., 3.12.0 version) does not support the installation of **the slither-analyzer** tool, which is needed to run the Slither static analysis tool.
- You can use other virtual environments like **virtualenv** or **anaconda**, but in this guide, we will instruct you on how to use **venv**.
- For more information about the dependencies needed for both frontend and backend, please refer to the **"frontend/package.json"** file for React dependencies and the **"backend/requirements.txt"** file for FastAPI dependencies.

**Installation and Setup:**

**1. Download and extract source code:**

- Download the source code folder **"group2-25.zip"** and unzip it.

**2. Open terminals:**

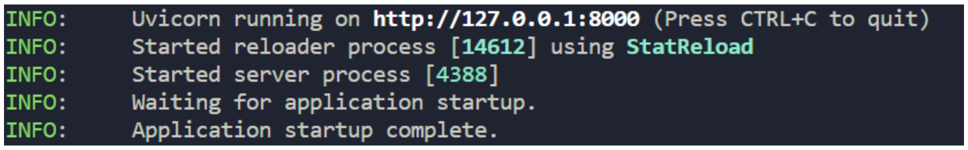
- Open two terminals: one for running the FastAPI backend server and another for running the React frontend.
- For Windows, use command prompt or PowerShell, and for MacOS/Linux, use the Terminal.

**3. Backend deployment (FastAPI):**

In the first terminal/command prompt:

- Navigate to the backend directory/folder: **cd backend**
- Set up a virtual environment to isolate project dependencies. In this guide, we use a Python built-in **venv** module to create a virtual environment named 'venv': **python -m venv venv**
- Activate the **venv**:
    - For Windows user: **venv\Scripts\activate**

    - For MacOS users: **source venv/bin/activate**
- Once you are in the virtual environment ‘venv’: the prompt should change to have **‘(venv)’** at the beginning of each line of the prompt. Additionally, you will notice a new folder named 'venv' in the project directory 'backend.'
- Install project dependencies: **pip install -r requirements.txt**
- Run the FastAPI backend server: **uvicorn main:app --reload**
- Wait for the backend to lit and it should show "Application startup complete." in the terminal, as shown in Figure 27.



```
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [14612] using StatReload
INFO:     Started server process [4388]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

*Figure 27: FastAPI successfully loading messages.*

**Command summary for Windows users:**

```
cd backend
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
uvicorn main:app --reload
```

*Figure 28: Commands needed to run the backend for Windows users.*

**Command summary for Linux or MacOS users:**

```
cd backend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
uvicorn main:app --reload
```

*Figure 29: Commands needed to run the backend for Linux/MacOS users.*

The FastAPI backend, which uses the Uvicorn server, is now up and running at http://127.0.0.1:8000. The backend API is documented at http://127.0.0.1:8000/docs.

**Note: special or hidden characters**

- When copying commands directly from this document, be cautious of special or hidden characters that may not be visible. Manually typing the commands is recommended to ensure the accuracy of the entered commands.

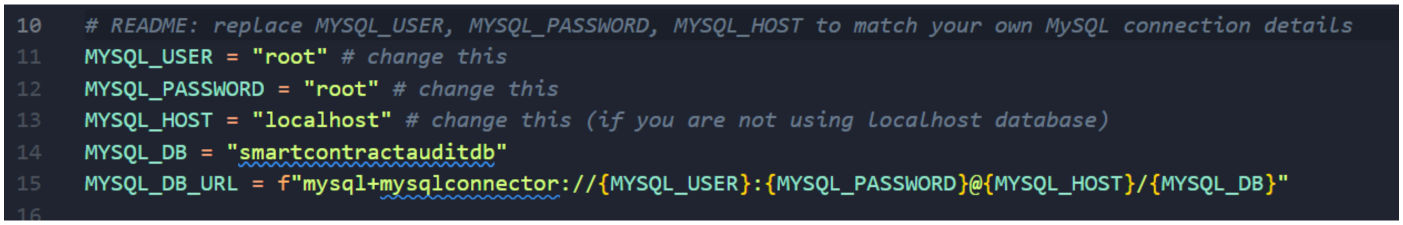
**4. MySQL database setup:**

While the FastAPI backend is still up:

- Go to the ‘**database.py’** file within the ‘**backend’** folder and change the **MYSQL_USER, MYSQL_PASSWORD,** and **MYSQL_HOST** variables to match your own MySQL connection.
- Save the **database.py** file.
- Once FastAPI shows "Application startup complete," the "**smartcontractauditdb**" database and its tables should be automatically created for you. If not, and you have configured the MySQL user to have limited privileges, you can manually create a new database in MySQL with this command:

**CREATE DATABASE smartcontractauditdb;**

- Make sure that the database name matches the value of the **MYSQL_DB** variable. Then, the three tables ("reports," "vulnerabilities," and "results") should be created automatically in MySQL for you.



```python
# README: replace MYSQL_USER, MYSQL_PASSWORD, MYSQL_HOST to match your own MySQL connection details
MYSQL_USER = "root" # change this
MYSQL_PASSWORD = "root" # change this
MYSQL_HOST = "localhost" # change this (if you are not using localhost database)
MYSQL_DB = "smartcontractauditdb"
MYSQL_DB_URL = f"mysql+mysqlconnector://{MYSQL_USER}:{MYSQL_PASSWORD}@{MYSQL_HOST}/{MYSQL_DB}"
```

*Figure 30: MySQL credentials on backend/database.py file.*

**5. Frontend deployment (React):**

In the second terminal/command prompt:

- Navigate to the frontend directory: **cd frontend**
- Install project dependencies: **npm install**
- This command reads the **"package.json"** file in the **"frontend"** directory and installs the required dependencies listed there. Wait for the dependencies to be installed, and a new ‘**node_modules’** folder will be created within the ‘**frontend’** directory.
- Start the React project: **npm start**
- React will take some time to build and load the application. Once it is ready, your default web browser will open, and you should be redirected to our frontend interface website, specifically the home page where you can find the file uploader and start the auditing process.

**Command summary (for both Windows and MacOS/Linux users):**

```
cd frontend
npm install
npm start
```

*Figure 31: Commands needed to run frontend.*

The frontend will be available at http://localhost:3000. The React project uses port 3000 by default unless you change it.

## V. Conclusion

This report covers the progress made by Team 2-25 in developing a user-friendly Smart Contract Audit System, including both the frontend and backend aspects. Our focus was on following defined specifications and implementing suitable frameworks to create an accessible platform for blockchain developers. The report covers technical specifications and project background, team introduction, requirements, design, system architecture, frontend prototype, backend database design, API design, and functionality description with various use cases. By following the defined specifications, and using appropriate frameworks, the goal is to provide blockchain developers with an intuitive platform for assessing the security posture of their smart contracts.

## References

Frankenfield, J 2023, ‘Smart Contracts’, *Investopedia*, viewed 22 August 2023, <https://www.investopedia.com/terms/s/smart-contracts.asp>.

Trail of Bits 2018, ‘Slither – a Solidity static analysis framework’, *Trail of Bits Blog*, viewed 1 September 2023, <https://blog.trailofbits.com/2018/10/19/slither-a-solidity-static-analysis-framework/>.