
# Detailed Design Report

CleaverWall

Arda Barış Örtlek

Ali Emre Aydoğmuş

Onur Korkmaz

Selahattin Cem Öztürk

Yekta Seçkin Satır

**Supervisor:** Özcan Öztürk

**Jury Members:** Erhan Dolak, Tağmaç Topal

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

# 1. Introduction

CleaverWall is an open source anti-malware mechanism designed to detect whether a portable executable is malicious and classify the malware type. To achieve this, CleaverWall uses a set of classifiers that are trained using various machine learning and deep learning techniques. Static and dynamic analysis techniques are used to create different feature vectors for the classifiers, which work collaboratively to decrease false positive occurrences.

## 1.1 Purpose of the system

The purpose of the system is to provide users with a reliable and efficient malware scanning service that can detect and handle malicious portable executable files. The system aims to provide an easy-to-use interface for users to scan their files, view detailed scan results, and schedule auto-scans for specific file paths.

## 1.2 Design goals

Usability: CleaverWall's graphical user interface is designed to be clear and intuitive, making it easy to use. The graphical user interface styles of the desktop application and the web server are similar to each other to enhance the user experience when switching between the two services.

Security and Privacy: CleaverWall ensures that uploaded files are safe from cyber attacks. The classification model is protected from outside forces to ensure that it is not disturbed. Newly obtained and saved data to improve the machine learning model should not be disturbed by outside forces.

Reliability: CleaverWall's classification model can classify mainstream malware families. The model's accuracy can increase by using the new saved data.

Scalability: The server can analyze multiple files uploaded by different users concurrently.

Performance: Our design goal is to provide a system that can perform static and dynamic analysis on portable executable files in a reasonable time frame. We aim to provide a service that can handle multiple file uploads and requests concurrently.

Maintainability: The mean time to restore the system following a system failure on the server side must not be greater than 10 minutes. The mean time to restore the system includes all corrective maintenance time and delay time.

## 1.3 Definitions, acronyms, and abbreviations

- PE: Portable Executable
- API: Application Programming Interface
- NN: Neural Network
- CNN: Convolutional Neural Network
- ML: Machine Learning
- DL: Deep Learning

## 1.4 Overview

CleaverWall uses three different approaches to classify malware. In the first approach, the portable executable is disassembled, and information such as operation codes, registers, symbols, sections, miscellaneous, and Windows API calls are analyzed to create features. In the second approach, the portable executable is represented as a grayscale image, and Convolutional Neural Networks are used for classification. The third approach is to combine the first two approaches to create a multimodal malware classifier.

Dynamic analysis is conducted using a sandbox, and information regarding the API call sequence is collected. The dataset for training the models is taken from VirusShare, and the academic API of VirusTotal is used to label the dataset.

CleaverWall is an open source project that aims to surpass other anti-malware mechanisms that use only signature-based detection methods. The innovation type is product performance, and it is incremental.

# 2. Current software architecture

## 2.1 VirusTotal [1]

VirusTotal is an online service that provides users with the ability to scan files and URLs for potential threats. The service uses a range of antivirus engines and other security tools to analyze files and URLs and provide users with a comprehensive view of any potential security risks.

Users can upload files or enter URLs into the VirusTotal website, and the service will automatically scan the file or URL using a range of antivirus engines, as well as other security tools, such as behavior-based analysis and sandboxing. The results of the scan are then displayed to the user, along with detailed information on any potential threats detected.
One of the key features of VirusTotal is its ability to aggregate data from multiple antivirus engines and other security tools. This allows users to get a more comprehensive view of the potential threats associated with a file or URL, as well as identify false positives or potential inaccuracies in individual scans. VirusTotal also allows users to view the scan results from

individual antivirus engines and security tools, providing more detailed information on each individual scan.

VirusTotal also provides a range of additional features and tools designed to help users analyze and investigate potential threats. For example, the service includes a community-based platform where users can discuss and share information about potential threats, as well as a set of specialized tools for analyzing and investigating specific types of malware and other security threats. These tools include a behavior-based analysis tool, which allows users to analyze the behavior of potential threats in a virtual environment, as well as a sandboxing tool, which allows users to execute files in a controlled environment to identify potential malicious behavior.

VirusTotal also provides APIs and other integration tools that allow developers to integrate the service into their own applications and workflows. This can be particularly useful for organizations that need to scan large volumes of files or URLs as part of their security operations.

## 2.2 SentinelOne [2]

SentinelOne's endpoint protection platform is designed to protect organizations from a wide range of cyber threats, including malware, ransomware, and other advanced threats. The platform uses artificial intelligence and machine learning to detect and respond to threats in real-time, without relying on signatures or rules. This means that the platform can detect and prevent new and emerging threats, as well as known threats, with high accuracy.

One of the key features of SentinelOne's platform is behavioral detection, which analyzes the behavior of processes and applications to detect malicious activity. The platform also offers fileless attack prevention, which can detect and prevent attacks that use fileless techniques, such as PowerShell attacks.

SentinelOne's platform also includes automated remediation capabilities, which can automatically contain and remediate threats in real-time, without requiring manual intervention. This can help organizations respond to threats quickly and efficiently, minimizing the impact of cyber attacks.

In addition to its endpoint protection solutions, SentinelOne also offers a threat intelligence service called "SentinelOne Intelligence". This service provides real-time threat intelligence and analysis to help organizations stay ahead of emerging threats. The service includes a team of security experts who analyze threats and provide recommendations for remediation and prevention.

SentinelOne's solutions are designed to be easy to deploy and manage, with a cloud-based management console that provides real-time visibility into endpoint security. The platform

also integrates with other security solutions, such as SIEM and SOAR platforms, to provide a comprehensive security solution.

## 2.3 Sophos [3]

Sophos is a global cybersecurity company that provides a range of security solutions for businesses of all sizes. The company's solutions are designed to protect against a wide range of cyber threats, including malware, ransomware, phishing, and other advanced threats.

Sophos offers a variety of endpoint protection solutions, including Sophos Intercept X, which is designed to protect against known and unknown malware threats using a combination of signature-based and behavioral-based detection. The endpoint protection solutions also include features such as web filtering, device control, and application control, which can help organizations enforce security policies and protect against data loss.

Sophos' network security solutions include firewalls, wireless access points, and VPN solutions. The company's firewall solutions are designed to provide advanced threat protection and secure remote access for businesses of all sizes. The wireless access points are designed to provide secure and reliable Wi-Fi connectivity, while the VPN solutions enable secure remote access for employees and partners.

Sophos' cloud security solutions include products for securing cloud infrastructure, applications, and data. The company's encryption solutions provide secure data protection for organizations, with features such as full-disk encryption, file and folder encryption, and email encryption.

Sophos also offers a range of professional services, including security assessments, incident response, and training, to help organizations maximize the value of their security investments. The company's professional services are designed to help organizations identify vulnerabilities, respond to incidents, and improve overall security posture.

Sophos uses machine learning in its endpoint protection solutions, such as Sophos Intercept X, to provide advanced threat detection capabilities. The software uses deep learning neural networks to analyze files and identify potentially malicious behavior, such as attempts to modify system files or access sensitive data. Sophos also uses machine learning to improve its web filtering capabilities. The software analyzes web traffic and identifies potentially malicious URLs, blocking access to these sites before they can cause harm.

# 3. Proposed software architecture

## 3.1 Overview

In this section, the software architecture used for the project and subsystems of the architecture is explained in detail. Firstly, subsystem decomposition and functions of each subsystem are given. Then, a hardware/software mapping diagram is given to illustrate which hardware will use which software. Then, how the project is going to store the required data is explained. Finally, precautions taken for security and access control are explained.

## 3.2 Subsystem decomposition

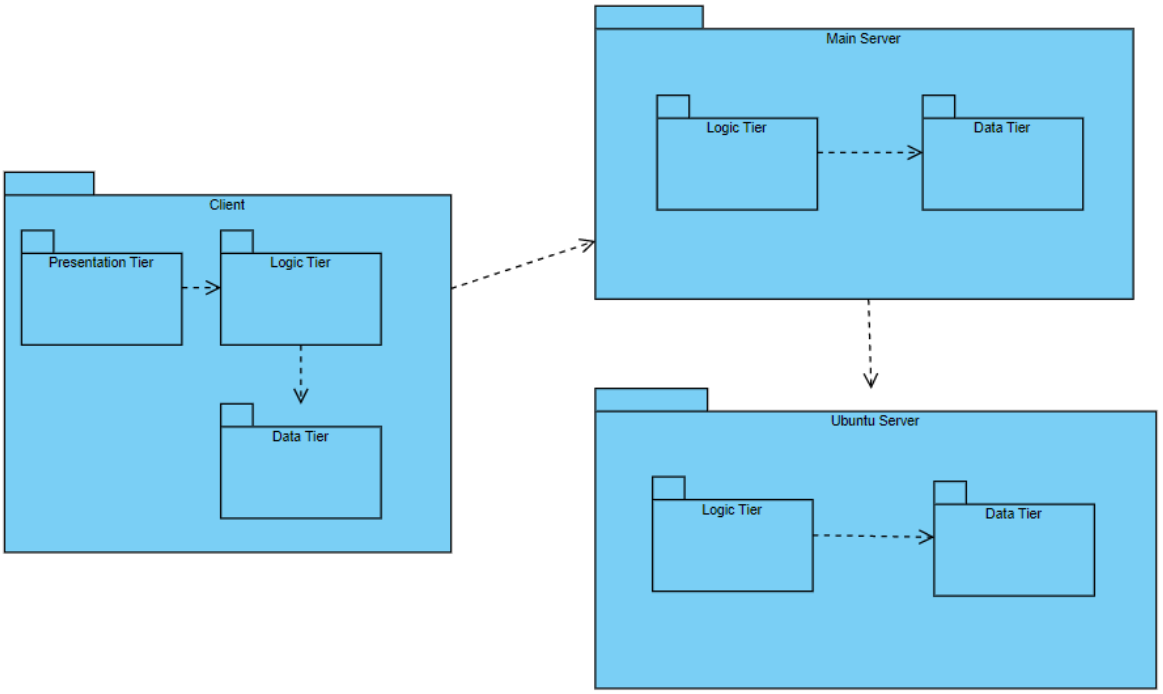


*Figure 1: Subsystem decomposition of the whole system.*

CleaverWall is based on server-client architecture. There are two types of applications on the client side, the web application and the desktop application. Both sides are being developed using Flutter. There are three subsystems on the client side, Presentation Tier and Logic Tier, and Data Tier. Presentation Tier will handle basic user interface functionalities. Specifically, it is the layer of widgets and states environment in the Flutter framework. Logic Tier will handle processing the data sent by the Main Server, logging in, and switching between pages. Data Tier will handle repositories and data structures that are used to illustrate in the view. After developing the web client, the code will be transferred to the desktop application using

Flutter's service. Then, Logic Tier of the desktop application will be extended for more functionalities such as offline static analysis.

There are two servers that will be used in the project. A main server will be used to communicate with the client, maintain crud operations in the database, do static analysis, and communicate with the side server when the dynamic analysis is needed. Main Server is decomposed into two subsystems, Logic Tier and Data Tier. Logic Tier contains the core logic of the application. Data Tier handles the storage of information of users and submissions. The second server handles the operations of dynamic analysis. Because Cuckoo Sandbox needs an Ubuntu environment to run, the side server will run with Ubuntu. The side server is also decomposed into two subsystems, Logic Tier and Data Tier. Logic Tier of the Ubuntu server actuates the Cuckoo Sandbox and operates on it by requests from the Main Server. After getting results, it runs the dynamic analysis model to do classification, then sends the outputs to the main server. Data Tier of the Ubuntu server stores requests from the Main server temporarily.

## 3.3 Hardware/software mapping

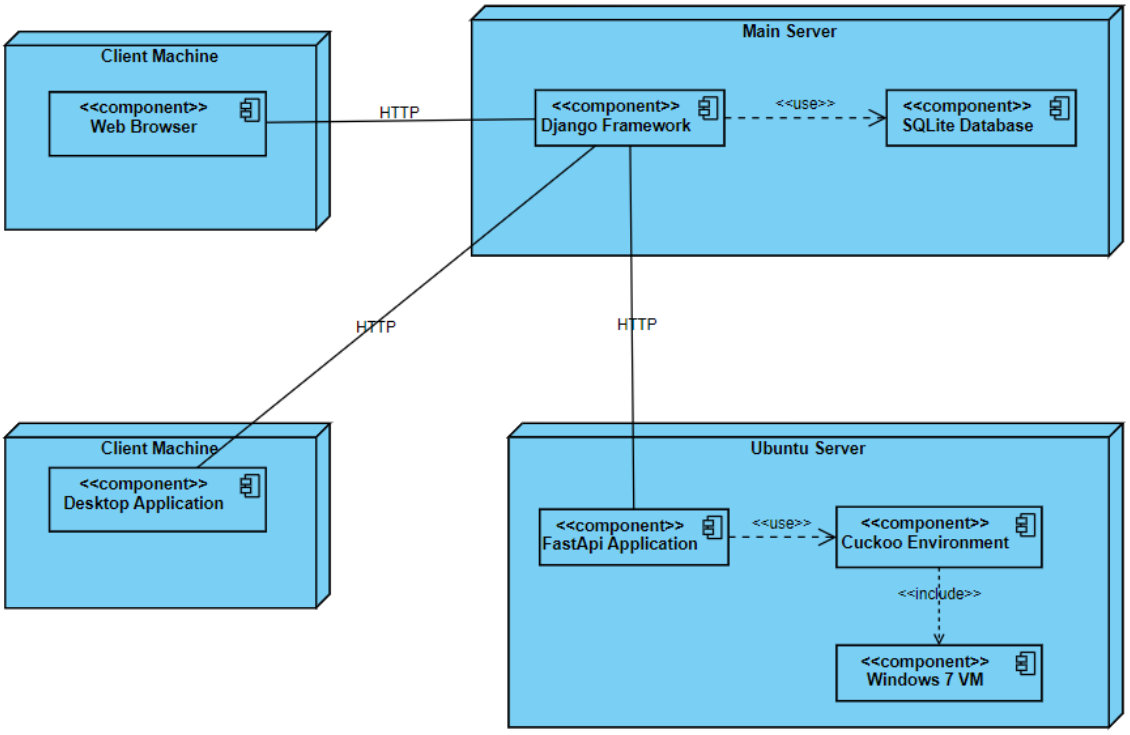


*Figure 2: Hardware and software mapping of the system.*

The communication between the main server and the client machine is done via the web browser or the desktop application on the client side. On the main server, a backend server

has been implemented using Django Framework. The results and data are stored in an SQLite database. Also, because the static analysis is handled in this server, the static analysis model resides here.

If the dynamic analysis is needed the main server sends the file to the Ubuntu server. In this server, a back-end application will handle these requests by the FastApi framework. A Cuckoo Sandbox environment will run permanently on this server. The sandbox uses a Windows 7 virtual machine on the QEMU emulator to run portable executables and get the features. Also, the dynamic analysis model resides here to get the classification outputs.

## 3.4 Persistent data management

We need to store User and Submission information. For the User, the information about logging in such as user name and password will be stored. For Submission, information about the metadata of the file such as MD5 hash, static analysis features, dynamic analysis features, the outputs of classification, and intermediate values like the result of the greyscale image model, the date of submission, and user information will be stored. All of this data must be operated properly and stored persistently. We are using SQLite database for managing the data.
The ubuntu server will also have a database running on SQLAlchemy, only for internal logging purposes. This database will not have an effect on the application logic.

## 3.5 Access control and security

In CleaverWall, there is only one type of user. A user can submit a file to be analyzed, get the result of the submission and related features of the submitted file, and access the submission history and information about his/her account. However, the user should not be able to access other people's submissions, accounts, and submission history. Because the communication between the main server and clients is done via HTTP and REST API, the backend application will be adjusted to prevent the above cases. To prevent any cross-site forgery attacks, CsrfViewMiddleware's token mechanism of Django Framework will be used.

An important issue about security is uploaded content. PEs will be run only inside the Cuckoo Sandbox environment. This will protect any harmful actions to the operating system and FastApi application in the Ubuntu Server. Also, Only communication of the Ubuntu Server will be done between the Main Server. Hence, any unwanted requests from clients will be obstructed.

# 4. Subsystem services

## 4.1 Client

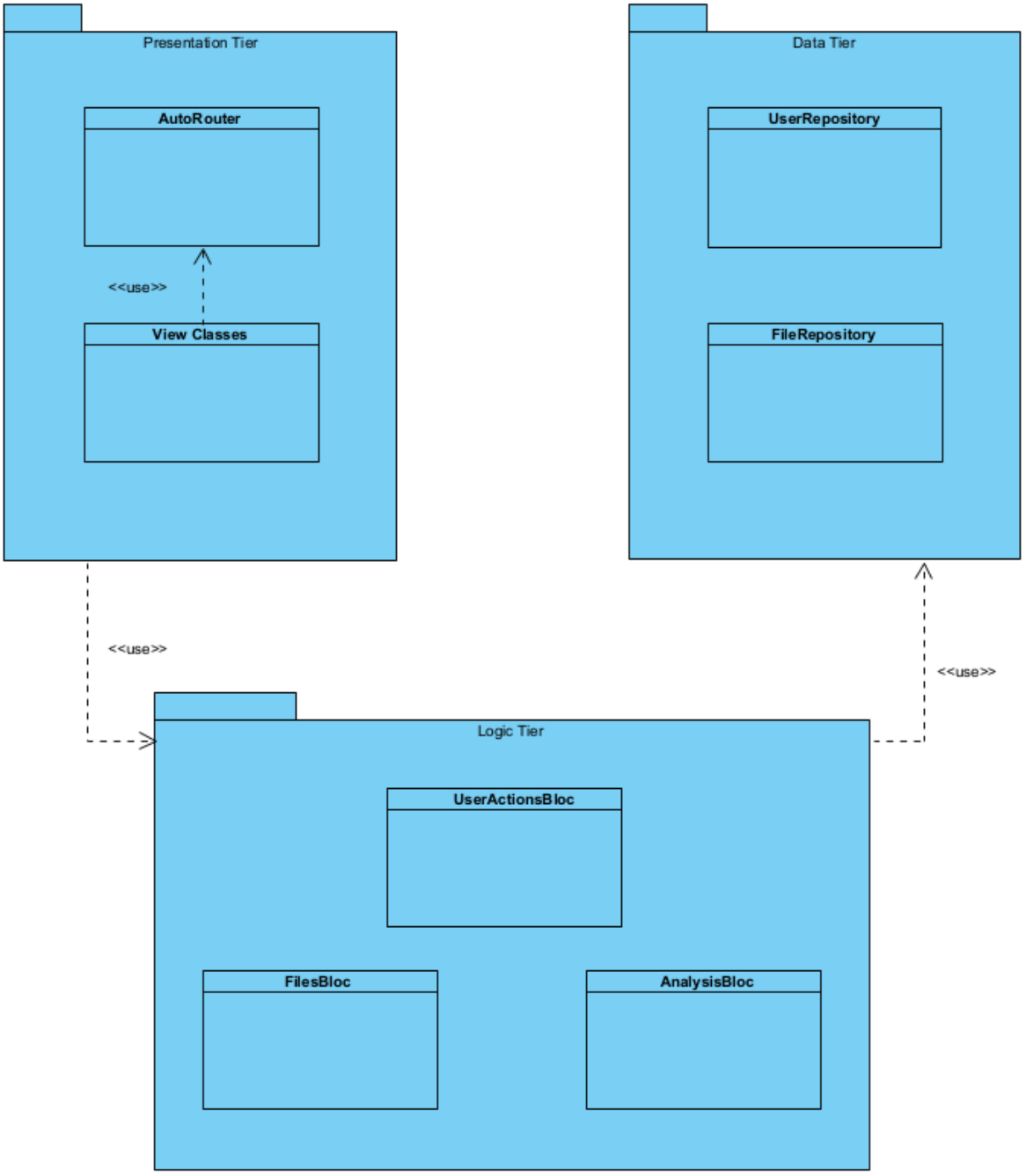


*Figure 3: Subsystem decomposition of the Client Side.*

CleaverWall client consists of mobile and desktop, however at its current iteration they both have the same functionalities. The client subsystem is divided into three tiers: Presentation, Logic and Data. Presentation tier is used to display and allow the user to

interact with the project. While the Presentation tier is a dummy, Logic tier handles all the functionalities. Data tier works mostly like a storage, and is responsible for both managing local temporary files and requesting data from the server, as well as sending data. The presentation tier interacts with the Logic tier in order to make the UI functional, and the Logic tier interacts with the Data tier to make the functionality meaningful.

### 4.1.1 Presentation Tier

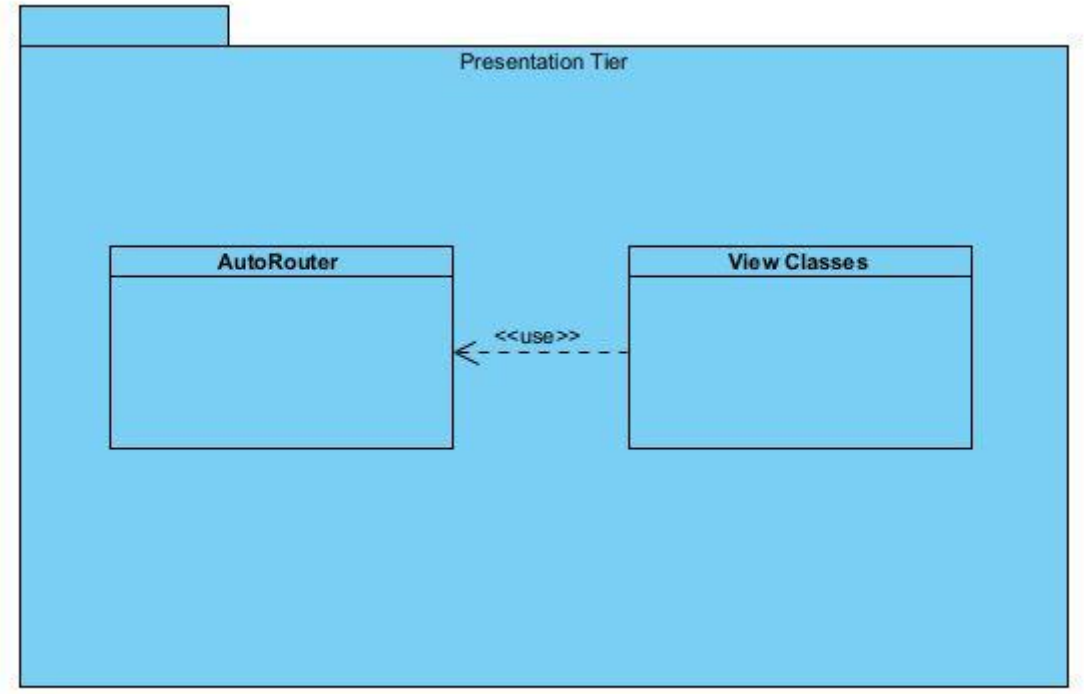


*Figure 4: Subsystem decomposition of the Client's Presentation Tier.*

The Presentation tier consists of AutoRouter and all the view classes. Autorouter is a flutter library that allows easy transition between the views. Additionally, it automatically handles page URLs, making it more user friendly on the web-side. View classes are flutter classes that return build functions that build the UIs as described previously with the mock UIs. They contain no functionality whatsoever -other than navigating through the app and viewing server data etc.- , except for the uploadFileRoute, which requires the file upload pop-up.

### 4.1.2 Logic Tier

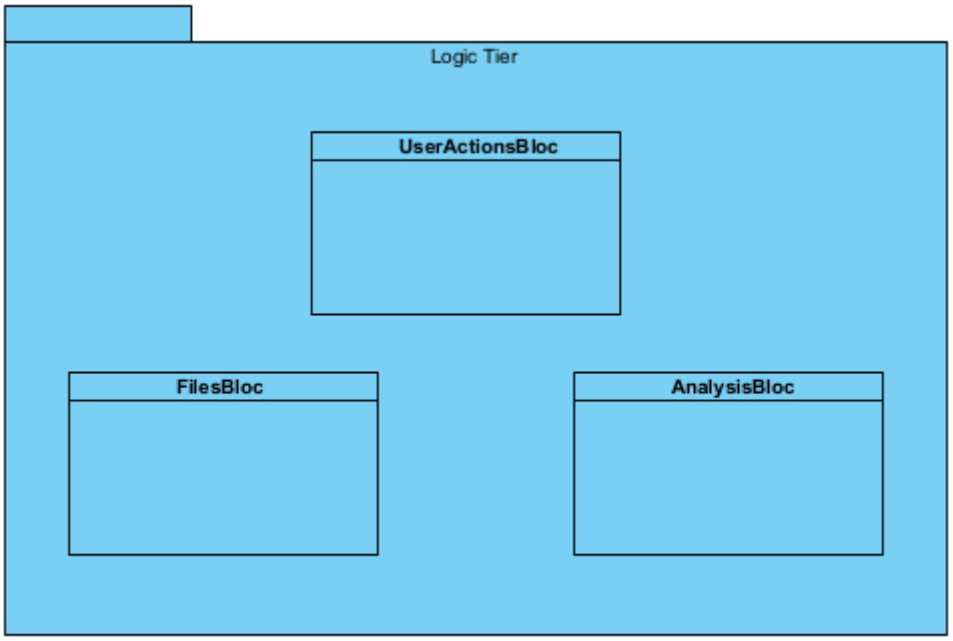


*Figure 5: Subsystem decomposition of the Client's Logic Tier.*

The Logic tier consists of business logic components (blocs) matching the functionality contexts.

Blocs consist of 3 classes: bloc, state and event. States are blocs' mutable variable storage. Every bloc has only one state. Events are fired through the UI on specific interactions and are used to notify the bloc that it needs to do something. Bloc itself is where the logic runs: it tells the repositories to change or to request some data, and then modifies its states. In return, UIs listening to the related bloc update themselves according to the new bloc state.

**UserActionsBloc:** This bloc handles the user login and signup. Also includes validation logic for text boxes on both login and signup.

**FilesBloc:** This bloc is responsible for file upload functionality. It handles things like storing the file path, checking the file size and file type.

**AnalysisBloc:** This is the bloc that requests analysis results from the server, both for the last uploaded file and for the analysis history.

### 4.1.3 Data Tier

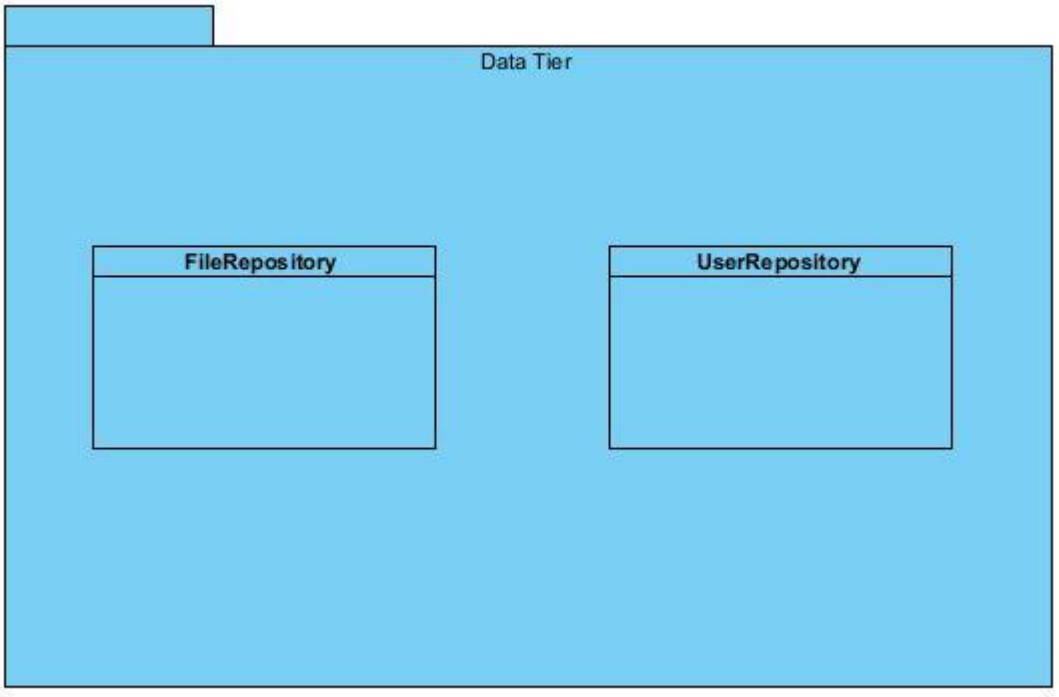


*Figure 6: Subsystem decomposition of the Client's Data Tier.*

The Data tier stores and requests data and delivers it to the blocs whenever necessary.

**UserRepository:** This repository handles the current user data and requests such as storing the user token and requesting a login.

**FileRepository:** This repository is responsible for storing the uploaded file until it's sent as well as the analysis data that is received from both the latest upload and from the analysis history. Both analysis and files are currently managed in one repository because of a potential future merge on the analysis and file classes.

## 4.2 Main Server

### 4.2.1 Logic Tier

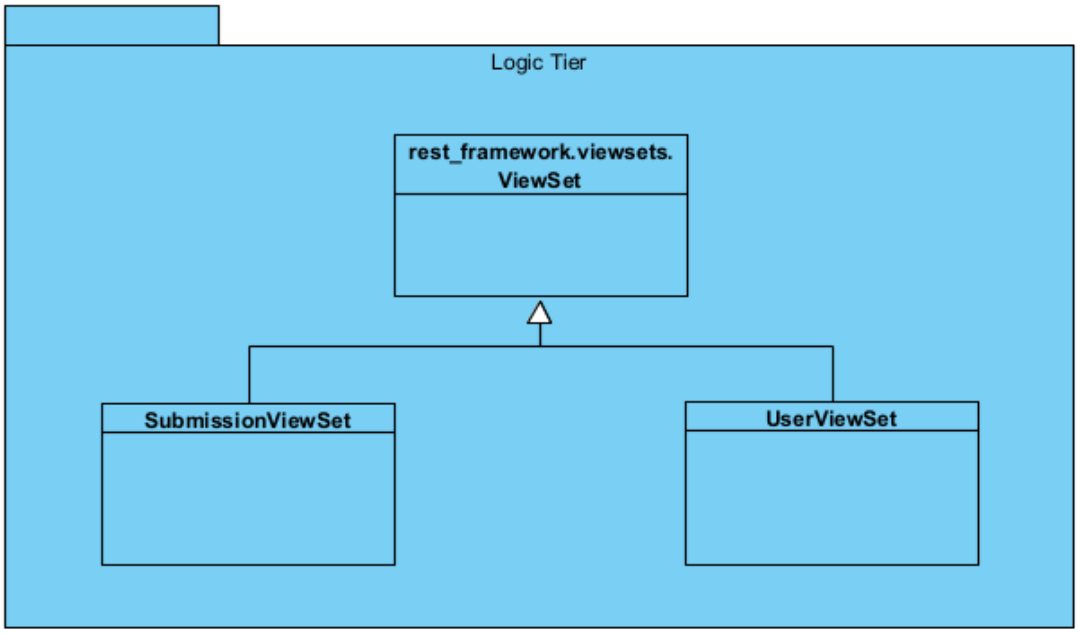


*Figure 7: Subsystem decomposition of the Main Server's Logic Tier.*

Logic tier of the server is responsible for business decisions: using different modes, holding states during script executions, communication with the Ubuntu server when needed. It is the equivalent of Controller classes in other popular backend frameworks such as ASP.NET. This layer also provides endpoints for clients in sets.

- UserViewSet: Handles the registration, login, and logout operations for the user.
- SubmissionViewSet: Implements create, list, and retrieve functionalities for Submission objects. Accesses miscellaneous utils files and scripts to execute analysis. If the submission mode requires a simple type of analysis, classifies and returns the result. Else requests the Ubuntu server to do more expensive operations.

### 4.2.2 Data Tier

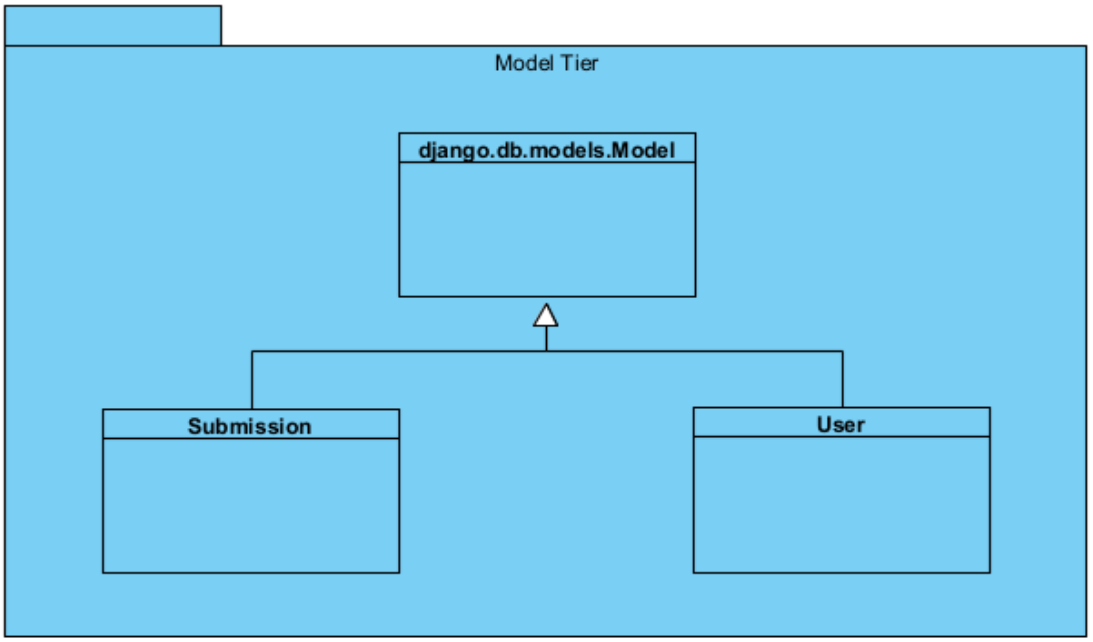


*Figure 8: Subsystem decomposition of the Main Server's Data Tier.*

Data tier of the main server consists of model classes. It defines the entities and their fields for the SQLite engine.

- User: Holds basic user data
- Submission: Holds information about submissions regarding the file, submission details, results and whether they are still valid.

## 4.3 Ubuntu Server

### 4.3.1 Logic Tier

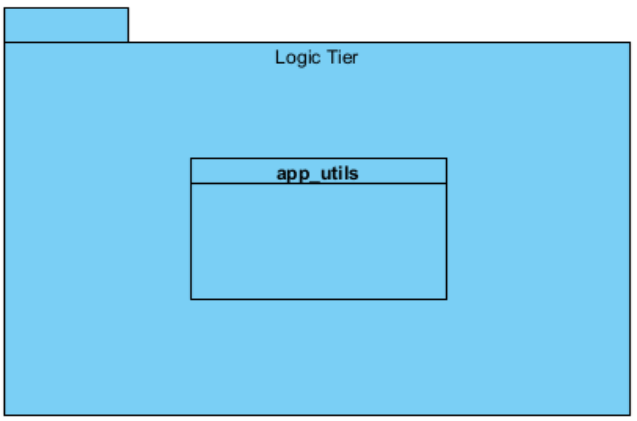

*Figure 9: Subsystem decomposition of the Ubuntu Server's Logic Tier.*

Logic tier of the Ubuntu server is responsible for managing expensive operations with the Cuckoo sandbox and ensuring a stable communication with it and the main server.

- app_utils: Only the class of this application that contains business logic. Manages analysis requests, and contains functions to communicate with the Cuckoo sandbox and apply classification.

### 4.3.2 Data Tier

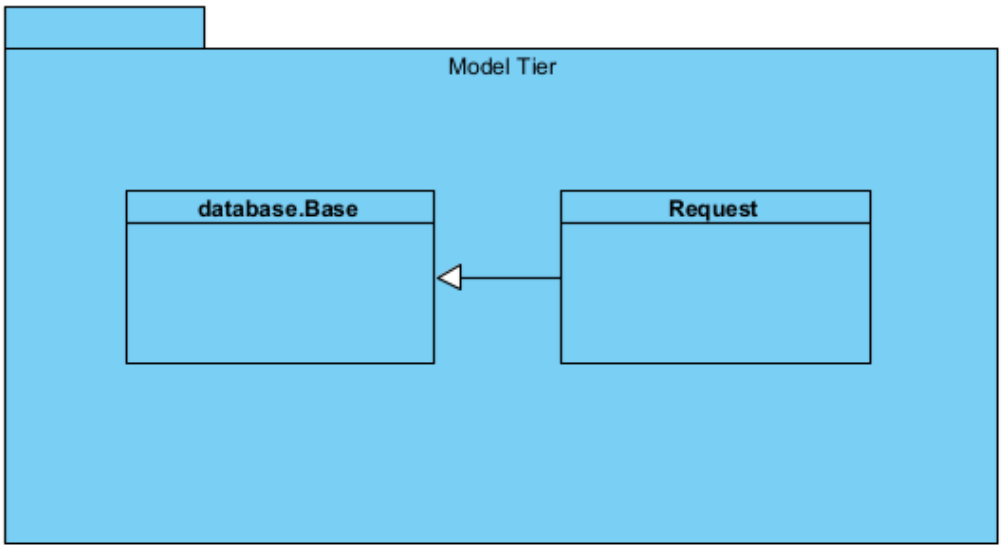


*Figure 10: Subsystem decomposition of the Ubuntu Server's Data Tier.*

Data tier of the Ubuntu server is only to store Requests for practicality.

- Requests: Holds basic information about current requests temporarily.

# 5. Test Cases

Functional and non-functional test cases related to client side, server side and machine learning are listed below. Id's of the test cases for client, server, and machine learning sides starts with CLNT, SRV, and ML, respectively.

## 5.1 Client Side Test Cases

Test ID: CLNT001
Test Type/Category: Functional
Summary/Title/Objective: User Login
Procedure of Testing Steps:

1. Open the app and navigate to the login page

2. Enter valid credentials and click on the login button
3. Verify that the user is logged in successfully and redirected to the home page

Expected Results/Outcome: The user should be able to log in successfully and access the app.
Priority/Severity: Critical

Test ID: CLNT002
Test Type/Category: Functional
Summary/Title/Objective: Token Expiration
Procedure of Testing Steps:

1. Login to the app with valid credentials
2. Wait for the token to expire
3. Attempt to access any protected resource
4. Verify that the user is logged out and redirected to the login page

Expected Results/Outcome: The user should be logged out and redirected to the login page when the token expires.
Priority/Severity: Major

Test ID: CLNT003
Test Type/Category: Functional
Summary/Title/Objective: Manual Logout
Procedure of Testing Steps:

1. Login to the app with valid credentials
2. Click on the logout button
3. Verify that the user is logged out and redirected to the login page

Expected Results/Outcome: The user should be able to log out successfully and be redirected to the login page.
Priority/Severity: Minor

Test ID: CLNT004
Test Type/Category: Functional
Summary/Title/Objective: New User Form Submission
Procedure of Testing Steps:

1. Navigate to the new user form
2. Fill in the required fields with valid data
3. Click on the submit button
4. Verify that the form is submitted successfully

Expected Results/Outcome: The new user form should be submitted successfully without any errors.
Priority/Severity: Minor

Test ID: CLNT005
Test Type/Category: Functional
Summary/Title/Objective: File Upload - Non-Executable File
Procedure of Testing Steps:

1. Navigate to the file upload page
2. Select a non-executable file and click on the upload button
3. Verify that the file is rejected and an error message is displayed

Expected Results/Outcome: The app should reject the non-executable file and display an error message.
Priority/Severity: Major

Test ID: CLNT006
Test Type/Category: Functional
Summary/Title/Objective: File Upload - Maximum File Size

1. Procedure of Testing Steps:
2. Navigate to the file upload page
3. Select a file that exceeds the maximum file size limit and click on the upload button
4. Verify that the file is rejected and an error message is displayed

Expected Results/Outcome: The app should reject the file that exceeds the maximum file size limit and display an error message.
Priority/Severity: Major

Test ID: CLNT007
Test Type/Category: Functional
Summary/Title/Objective: File Queue Status Streaming
Procedure of Testing Steps:

1. Navigate to the file queue page
2. Verify that the file queue status is being streamed correctly

Expected Results/Outcome: The app should stream the file

Test ID: CLNT008
Test Type/Category: Functional
Summary/Title/Objective: File History Deletion
Procedure of Testing Steps:

1. Navigate to the file history page
2. Select a file to delete and click on the delete button
3. Verify that the file history is deleted successfully

Expected Results/Outcome: The app should delete the file history correctly when requested by the user.
Priority/Severity: Minor

Test ID: CLNT009
Test Type/Category: Functional
Summary/Title/Objective: File Report Fetch
Procedure of Testing Steps:

1. Navigate to the file report page
2. Enter the required information to fetch the report
3. Click on the fetch button
4. Verify that the report is displayed correctly

Expected Results/Outcome: The app should fetch the file report correctly and display it without any errors.
Priority/Severity: Minor

Test ID: CLNT010
Test Type/Category: Compatibility
Summary/Title/Objective: Client Type Compatibility
Procedure of Testing Steps:

1. Use a desktop client to access the app
2. Verify that the app behaves correctly on the desktop client
3. Use a web client to access the app
4. Verify that the app behaves correctly on the web client

Expected Results/Outcome: The app should behave correctly on both desktop and web clients.
Priority/Severity: Minor

Test ID: CLNT011
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for User Interface Appearance
Procedure:

1. Wait until all packages arrive.
2. Check if the user interface appears.

Expected results/Outcome: The user interface should appear when all packages have arrived.
Priority/Severity: Major

## 5.2 Server Side Test Cases

Test ID: SRV001
Test Type/Category: Integration
Summary/Title/Objective: Main Server Availability
Procedure of Testing Steps:

1. Shutdown the Ubuntu server
2. Send a request to the main server that requires Ubuntu server
3. Verify that the main server responds successfully with an appropriate error message

Expected Results/Outcome: The main server should respond successfully and return an appropriate error message when the Ubuntu server is down.
Priority/Severity: Critical

Test ID: SRV002
Test Type/Category: Integration
Summary/Title/Objective: Database Operations Race Condition
Procedure of Testing Steps:

1. Trigger multiple requests to the server to the same resource simultaneously
2. Verify that the server returns accurate data without any conflict or errors

Expected Results/Outcome: The server should handle simultaneous requests without any race condition errors and return accurate data accordingly.
Priority/Severity: Major

Test ID: SRV003
Test Type/Category: Integration
Summary/Title/Objective: Main Server Update
Procedure of Testing Steps:

1. Start an execution in the Ubuntu server
2. Monitor the main server for the update of the results
3. Verify that the main server updates results when the execution in the Ubuntu server is done

Expected Results/Outcome: The main server should update the results correctly when the execution in the Ubuntu server is done.
Priority/Severity: Major

Test ID: SRV004
Test Type/Category: Integration
Summary/Title/Objective: Valid Result Availability
Procedure of Testing Steps:

1. Check if there is a valid result already exists
2. Send a request to the server with the valid data
3. Verify that the server responds immediately with the existing result

Expected Results/Outcome: The server should respond immediately if a valid result already exists and return the data accordingly.
Priority/Severity: Minor

Test ID: SRV005
Test Type/Category: Integration
Summary/Title/Objective: Model Update

Procedure of Testing Steps:

1. Check if there is a valid result already exists
2. Send a request to the server to update the machine learning model
3. Verify that the server updates the valid boolean fields to false

Expected Results/Outcome: The server should update the valid boolean fields accordingly when the model is updated.
Priority/Severity: Minor

Test ID: SRV006
Test Type/Category: Integration
Summary/Title/Objective: File Size Limit
Procedure of Testing Steps:

1. Send a file to the server that exceeds the maximum file size limit
2. Verify that the server rejects the file and returns an appropriate error message

Expected Results/Outcome: The server should reject files that exceed the maximum file size limit and return an appropriate error message.
Priority/Severity: Critical

Test ID: SRV007
Test Type/Category: Integration
Summary/Title/Objective: Executable File
Procedure of Testing Steps:

3. Send a file to the server that is not a portable executable
4. Verify that the server rejects the file and returns an appropriate error message

Expected Results/Outcome: The server should only accept executable files without any issue.
Priority/Severity: Critical

Test ID: SRV008
Test Type/Category: Integration
Summary/Title/Objective: Required Fields
Procedure of Testing Steps:

1. Send a submission request to the server with some required fields missing
2. Verify that the server rejects the request and returns an appropriate error message

Expected Results/Outcome: The server should reject submission requests without the required fields and return an appropriate error message.
Priority/Severity: Major

Test ID: SRV009
Test Type/Category: Security
Summary/Title/Objective: User Access Control
Procedure of Testing Steps:

1. Log in as User A and attempt to access the submissions of User B
2. Verify that the server denies access and returns an appropriate error message

Expected Results/Outcome: The server should not allow users to access other users' submissions and return an appropriate error message.
Priority/Severity: Critical

Test ID: SRV010
Test Type/Category: Integration
Summary/Title/Objective: Submission Listing
Procedure of Testing Steps:

1. Log in as User A and send multiple submissions to the server
2. Request the list of submissions for User A from the server
3. Verify that the server lists all the submissions of User A correctly

Expected Results/Outcome: The server should correctly list all the submissions of a user when requested.
Priority/Severity: Minor

Test ID: SRV011
Test Type/Category: Integration
Summary/Title/Objective: User Creation
Procedure of Testing Steps:

1. Send a request to create a new user with valid fields
2. Verify that the server creates the user and returns an appropriate success message

Expected Results/Outcome: The server should create a new user with valid fields and return an appropriate success message.
Priority/Severity: Critical

Test ID: SRV012
Test Type/Category: Integration
Summary/Title/Objective: Session Termination
Procedure of Testing Steps:

1. Log in as User A
2. Log out as User A
3. Attempt to access any protected resource as User A
4. Verify that the server denies access and returns an appropriate error message

Expected Results/Outcome: The server should end the session of a user when they log out and deny access to any protected resource when not authenticated.
Priority/Severity: Critical

Test ID: SRV013
Test Type/Category: Security
Summary/Title/Objective: User Access Control
Procedure of Testing Steps:

1. Log in as User A and create multiple submissions

2. Log in as User B and attempt to access the submissions of User A
3. Verify that the server denies access and returns an appropriate error message
4. Log in as User A and request the list of their own submissions
5. Verify that the server lists all the submissions of User A correctly

Expected Results/Outcome: The server should only allow users to access their own submissions and deny access to any other submissions.
Priority/Severity: Critical

Test ID: SRV014
Test Type/Category: Security
Summary/Title/Objective: Non-authenticated Access
Procedure of Testing Steps:

1. Attempt to access any protected resource without authentication
2. Verify that the server denies access and returns an appropriate error message

Expected Results/Outcome: The server should deny access to any protected resource without authentication and return an appropriate error message.
Priority/Severity: Critical

Test ID: SRV015
Test Type/Category: Integration
Summary/Title/Objective: Dynamic Analysis
Procedure of Testing Steps:

1. Send a request to perform dynamic analysis
2. Verify that the server performs dynamic analysis successfully and returns an appropriate success message

Expected Results/Outcome: The server should perform dynamic analysis successfully and return an appropriate success message.
Priority/Severity: Major

Test ID: SRV016
Test Type/Category: Integration
Summary/Title/Objective: Server-to-Server Communication
Procedure of Testing Steps:

1. Send a request from outside of the main server to the Ubuntu server
2. Verify that the Ubuntu server only listens to requests from the main server

Expected Results/Outcome: The Ubuntu server should only listen to requests from the main server.
Priority/Severity: Major

Test ID: SRV017
Test Type/Category: Functional

Summary/Title/Objective: Test Cases for Duplicate Username Rejection
Procedure:

1. Register a new user with a username that has already been used.

Expected results/Outcome: The system should reject the registration attempt and display an error message indicating that the username has already been taken.
Priority/Severity: Critical

Test ID: SRV018
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Cuckoo Server Recovery
Procedure:

1. Simulate a Cuckoo server failure.
2. Check if the Ubuntu server can recover from the failure and start running cuckoo.

Expected results/Outcome: The Ubuntu server should be able to recover from the failure and start running cuckoo.
Priority/Severity: Major

Test ID: SRV019
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for File Submission Rejection
Procedure:

1. Submit a file without selecting any file.

Expected results/Outcome: The system should reject the file submission and display an error message indicating that no file was selected.
Priority/Severity: Critical

## 5.3 Test Cases for Algorithms for Machine Learning

Test ID: ML001
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Size of Optional Header Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the size of optional header from the executable successfully.
Priority/Severity: Critical

Test ID: ML002
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Major Linker Version Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the major linker version from the executable successfully.
Priority/Severity: Critical

Test ID: ML003
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Minor Linker Version Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the minor linker version from the executable successfully.
Priority/Severity: Critical

Test ID: ML004
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Size of Code Header Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the size of code from the executable successfully.
Priority/Severity: Critical

Test ID: ML005
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Size of Initialized Data Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the size of initialized data from the executable successfully.
Priority/Severity: Critical

Test ID: ML006
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Size of Uninitialized Data Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the size of uninitialized data from the executable successfully.
Priority/Severity: Critical

Test ID: ML007
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Address of Entry Point Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the address of entry point from the executable successfully.
Priority/Severity: Critical

Test ID: ML008
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Base of Code Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the base of code from the executable successfully.
Priority/Severity: Critical

Test ID: ML009
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Base of Data Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the base of data from the executable successfully.
Priority/Severity: Critical

Test ID: ML010
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Image Base Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the image base from the executable successfully.
Priority/Severity: Critical

Test ID: ML011
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Section Alignment Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the section alignment from the executable successfully.
Priority/Severity: Critical

Test ID: ML012
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful File Alignment Extraction

Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the file alignment from the executable successfully.
Priority/Severity: Critical

Test ID: ML013
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Major Operating System Version Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the major operating system version from the executable successfully.
Priority/Severity: Critical

Test ID: ML014
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Minor Operating System Version Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the minor operating system version from the executable successfully.
Priority/Severity: Critical

Test ID: ML015
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Byte to Pixel Conversion
Procedure:

1. Input a byte data to the conversion algorithm.

Expected results/Outcome: The algorithm should convert the byte data to pixel data successfully.
Priority/Severity: Critical

Test ID: ML016
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Image Resizing
Procedure:

1. Input a grayscale image to the resizing algorithm.

Expected results/Outcome: The algorithm should resize the image to the specified dimensions successfully.
Priority/Severity: Critical

Test ID: ML017
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Output of First Static Model
Procedure:

1. Input a dataset to the first static model.

Expected results/Outcome: The first static model should give softmax output successfully.
Priority/Severity: Major

Test ID: ML018
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Output of Second Static Model
Procedure:

1. Input a dataset to the second static model.

Expected results/Outcome: The second static model should give softmax output successfully.
Priority/Severity: Major

Test ID: ML019
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Output of Dynamic Model
Procedure:

1. Input a dataset to the dynamic model.

Expected results/Outcome: The dynamic model should give output successfully.
Priority/Severity: Major

Test ID: ML020
Test Type/Category: Functional
Summary/Title/Objective: Test Cases for Successful Size of Headers Extraction
Procedure:

1. Input a dataset to the feature extraction algorithm.

Expected results/Outcome: The algorithm should extract the size of headers from the executable successfully.
Priority/Severity: Critical

# 6. Consideration of Various Factors in Engineering Design

## 6.1 Public Health

CleaverWall and public health are not correlated.

## 6.2 Public Safety

Every year, ransomware alone costs the public $20 billion yearly. Although some of these attacks may not be caught by current signature-based anti-viruses, since CleaverWall utilizes quick static model responses for malware detection it could prevent some percentage of these attacks, potentially saving millions of dollars worldwide.

## 6.3 Public Welfare

Governments or agencies are backbones of welfare since they provide items and services to the public. These entities employ a lot of personnel who are doing civil servant jobs, who generally sit at a desk and work on computers 8 to 5. Although these people are expected to use computers very often, they are not always tech savvy and are more likely to fall for malware scams while browsing the internet. Their time is precious to waste on such setbacks and they're better off practicing their finesse for easing the bureaucracy or helping people. CleaverWall is a potential helping hand for these situations by providing a quick feedback for malicious software and preventing people from falling for potential scams. Also since it is open-source, these entities could modify CleaverWall for their own use, resulting in potential widespread use.

## 6.4 Global Factors

Since CleaverWall is open source, its success means that people will dissect and analyze the project. Although it is currently not popular, a successful project that uses machine learning only could light new ideas in the analyzers' heads. Additionally, the file recognition model could expand outside the malware detection application and find new uses in other fields.

## 6.5 Cultural Factors

The project could only have a meta effect on the culture, meaning it could affect the roots it came from: programmers. Although not for scale, the effect could be that of some algorithms often taught in programming. It could set a common ground for some certain set of coders, making them form sentences such as "It's similar to Dijkstra" or "It would be nice to use a Knapsack here".

## 6.6 Social Factors

CleaverWall has nothing to do with any social status such as age, gender, ethnicity, race etc. Therefore social factors are not determining factors for CleaverWall usage. Also it seems like CleaverWall will not have any effect on society.

| | Effect level | Effect |
|---|---|---|
| Public health | 0 | The project has nothing to do with health. |

| | | |
|---|---|---|
| Public safety | 4 | Static responses of the project are expected to protect its users from obvious scams of easy-viruses. |
| Public welfare | 1 | Governments or other agencies might decide to provide the project, or a version of theirs (since it is open source) to their civil servant jobs so that some simple setbacks could be avoided, which in turn could increase their efficiency. |
| Global factors | 5 | The project's success could lead to some breakthroughs on file recognition using ML, and even might extend the use outside of malware detection. |
| Cultural factors | 2 | If successful, the project may have an antsy effect on the programmer culture. |
| Social factors | 0 | The project is unlikely to impact the current era of civilization. |

Table 1: Factors that can affect analysis and design.

# 7. Teamwork Details

## 7.1 Contributing and functioning effectively on the team

Each member of our team is expected to actively contribute and function effectively within their respective work packages (WPs) and within the team as a whole. We understand that effective teamwork requires clear communication, active participation, and a commitment to meeting deadlines and fulfilling responsibilities.

To ensure that we are all on the same page, we will establish communication channels that enable us to keep each other informed of project developments and provide regular updates. We will also be available to offer support and assistance to our fellow team members whenever needed.

In addition to our individual contributions, we recognize the importance of working collaboratively to achieve our shared goals. We will be open to feedback, suggestions, and ideas from all members of the team, regardless of their role or seniority. We believe that by valuing and incorporating each other's perspectives, we can develop a more comprehensive understanding of the project and achieve better outcomes.

## 7.2 Helping creating a collaborative and inclusive environment

We believe that creating a collaborative and inclusive environment is essential to building a successful team. We will work together to foster a safe and supportive environment where everyone feels heard, respected, and valued.

To achieve this, we will actively listen to one another, provide constructive feedback, and encourage open and honest communication. We recognize that diverse perspectives and experiences can help us to identify opportunities and potential solutions that we might not have otherwise considered. Therefore, we will actively seek out and incorporate ideas and suggestions from all members of the team.

In addition to creating a supportive team culture, we are committed to maintaining a sense of accountability and responsibility for our actions. We will hold ourselves and each other accountable for upholding the principles of inclusivity and respect, and will take appropriate action if these principles are ever compromised.

## 7.3 Taking lead role and sharing leadership on the team

We understand that effective leadership is critical to the success of any team. While we have identified leaders for each WP, we also recognize that leadership is a shared responsibility that extends beyond formal roles. We believe that everyone has valuable skills and expertise to offer, and we encourage all team members to take on leadership roles and share their knowledge with the team.

As leaders, we will be responsible for ensuring that our team members are meeting their deadlines and fulfilling their responsibilities. We will also be available to offer guidance and support when needed. However, we also recognize that our success as a team is dependent on our ability to work collaboratively and take collective ownership of our project. Therefore, we will actively seek out and incorporate input from all members of the team, and encourage everyone to take an active role in shaping our project's direction and success.

In summary, we believe that effective teamwork requires clear communication, active participation, and a commitment to creating a collaborative and inclusive environment. By upholding these principles and working together, we believe that we can achieve our shared goals and develop a successful project.

# 8. Glossary

**Malware:** Harmful software aiming to cause damage to computer systems.
**Sandbox:** Testing environment on which potentially harmful softwares can be run safely.

# 9. References

[1] “Virustotal,” *VirusTotal*. [Online]. Available: https://www.virustotal.com/. [Accessed: 13-Mar-2023].

[2] “Autonomous AI Endpoint Security Platform: S1.ai,” *SentinelOne*, 08-Mar-2023. [Online]. Available: https://www.sentinelone.com/. [Accessed: 13-Mar-2023].

[3] “Cybersecurity as a service delivered,” *SOPHOS*, 09-Mar-2023. [Online]. Available: https://www.sophos.com/en-us. [Accessed: 13-Mar-2023].