Password Management System

CHARAN ANNADURAI, ENITS, Hochschule Offenburg, Germany

The password management system is made in accordance with the customer's specifications. All the best practices for software in every part of the software development were followed. Every core function is accompanied by Unit Tests which check for functionality and exception handling. This document contains the architecture with all the functionality requested along with the Static code analysis for the implementation. The Implementation uses Python to develop the endpoints, specifically the FastAPI Library. The unused passwords and the policy are stored in a .CSV file while the passwords are stored in a Mysql Database. The Master's password is hashed before storing. The System has 10 API endpoints including one which calls the REST API of HaveIBeenPwned.com and returns the leak status. The dominant data structures used are JSON and Python Dictionaries.

CCS Concepts: • **General and reference** → **Validation**; **Verification**; • **Software and its engineering** → **Designing software**; **Software design engineering**; **Requirements analysis**; **Software design techniques**; **Software prototyping**; **Error handling and recovery**; *Documentation*; **API languages**; **Unified Modeling Language (UML)**.

Additional Key Words and Phrases: Secure Development, APIs, Python Development, Password Management System

ACM Reference Format:
Charan Annadurai. 2023. Password Management System. 1, 1 (September 2023), 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Secure software development is a methodology (often associated with DevSecOps) for creating software that incorporates security into every phase of the software development life cycle (SDLC). Security is branded into the code from the start rather than addressed after testing reveals critical product flaws.[2] In today's digital world, software security has become an increasingly important aspect of software development due to the rise in cyber attacks and data breaches. To prevent such incidents, organizations must prioritize the security of their software and ensure that their applications are robust and resistant to cyber threats.

Secure software development involves following a set of best practices and guidelines, such as using secure coding techniques, implementing strong authentication and authorization mechanisms, conducting regular security testing, and continuously monitoring for vulnerabilities. In addition, software development teams should also stay updated on the latest security trends and technologies, and be proactive in addressing potential security issues as they arise.

> But the business of software building isn't really high-tech at all. It's most of all a business of talking to each other and writing things down. Those who were making

Author's address: Charan Annadurai, cannadur@stud.hs-offenburg.de, ENITS, Hochschule Offenburg, Badstraße 24, Offenburg, Baden-Württemberg, Germany, 77652.

XXXX-XXXX/2023/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

> major contributions to the field were more likely to be its best communicators than its best technicians. Tom DeMarco[4].

Compared to fixing a bug at the design stage fixing it in the implementation stage it is 6 times costlier and fixing it in the testing phase is 15 times costlier[2]. What's worse is Bugs are tolerable, but design flaws lead to critical system failures, this is why spending most of the design time in the design phase is recommended in order to produce fewer or ideally no design flaws. That is why this document will use the data from its design phase to convince the customer that this solution is secure. It is important at this juncture to define Verification and Validation. While Validation tells one whether they are building the right product what is more important is Verification which tells one whether they are building the product right.

2 TECHNICAL ARCHITECTURAL MODEL

A Technical Architectural Model (TAM) is a visual representation of the technical architecture of a software system. It provides a high-level overview of the components, relationships, and interactions between various software systems, applications, and technologies. A TAM is used to design, plan, and communicate the technical architecture of a software system, and is an important tool for understanding the technical aspects of a software solution.

The purpose of a TAM is to serve as a blueprint for the technical design of a system, providing a clear understanding of the different components, their interdependencies, and how they work together to achieve the desired outcome. A TAM also helps in identifying potential technical challenges and addressing them early in the software development life cycle, reducing the risk of technical failures or performance issues down the road.[4]

There are multiple components in a TAM diagram, which help an architecture represent the data flow in the system.

- Active components: These are the processes in the system that transform or manipulate the data. They are represented by rectangles in a TAM.
- Static components: These are the data stores or repositories in the system that hold data for later use. They are represented by rounded rectangles in a TAM.
- Request/response channels: These are the pathways through which data flows from one component to another. They are represented by lines with and R in a TAM.
- Read/write arrows: These are used to show the flow of data into and out of data stores. A read arrow that points out of a data store indicates that data is being retrieved from the store, while a write arrow that points into a data store indicates that data is being stored.

Figure 1 shows the TAM diagram for this particular implementation. The model contains containers which define boundaries of the systems. The structure described contains a "Legacy app" container, A PMS container and a Master Password System container which is a part of the PMS. This diagram includes the various active and passive components. Active components are components which have a defined purpose and can access passive components. Passive components cannot access other components but are accessed by active components and serve as data stores or sinks. The passive components in the diagram are the password.csv file, the master password Database, the applicaiton password Database and the policy.csv file The active components and their accesses are:-

(1) Admin - Has a request/response channel to Password Creator, Policy Creator and Password assigning/revoking.
(2) User - Has a request/response channel to View and Login.
(3) HaveIBeenPwned threat intel source - Only has a unidirectional response channel to the password creator.

(4) Password Creator - Has a unidirectional request channel to the HaveIBeenPwned threat intel source. Read access to the password policy and write access to the Batch generated passwords.
(5) Policy Creator - Has read and write access to the password policy and also to the App Password Database
(6) Password Assigning/Revoking - Has read access to the Barch generated passwords and read and write access to both the master and the application specific password databases.
(7) View - Has read access to both the master and the application specific password databases.
(8) Login - Has read access to the application specific database

The general flow would work like this, the admin would first create the policy and thereby create the policy.csv file. Using the policy.csv file a batch of passwords is generated and stored in the password.csv file after a leak check. When the admin wants to create a new user, a random password from the batch is selected and assigned as the master password temporarily. The applications that this user has access to also receive a separate password. This would enable login. The master password is hashed and stored in the master password database. If the policy is updated, all the application-specific passwords are replaced forcibly and the master password status is set to reset. The user when they try to log in to the master password system in order to view their application-specific passwords, would be met with an error asking them to reset their password. The users can then reset their password using a separate function as long as they do it within the stipulated time frame, otherwise, their account would be disabled. While resetting and while logging in their password is checked for leaks. The users can then log in into the master password system to view their application-specific passwords and use that to authenticate their legacy app session.

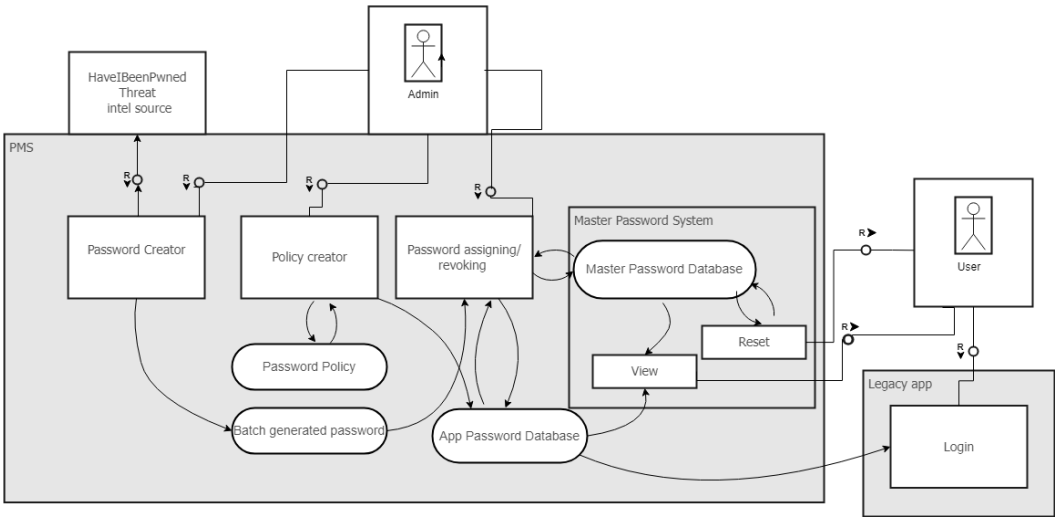

Fig. 1. TAM Diagram

3 REQUIREMENTS

A requirements table is an essential tool for capturing, organizing, and tracking the various requirements of a project, product, or system. It helps to ensure that all stakeholders have a clear and comprehensive understanding of what needs to be achieved, and it acts as a reference point throughout the development process. A requirements table can also be used to prioritize and

Table 1. Requirements Table

ID	Type	Description	Unit test purpose
SR 1	NFR	The system needs to efficiently manage application-specific passwords	
SR 1.1	FR	A function must be provided to create and update password policies.	Check if a policy file is created.
SR 1.2	FR	A function must be provided to generate a single / batch of random passwords according to current password policies	Check if a small number of passwords follow the policy and the necessary checks.
SR 1.3	FR	A function must check if a password has been leaked on the internet	Intentionally leak a password in vulnerable websites and measure the time taken for it to reflect on the database.
SR 1.4	FR	A function should assign passwords to a username - application_id pair.	Check in the database if a certain user has all allowed access possible.
SR 1.5	FR	A function must replace all current passwords with newly generated passwords (SR 1.2).	Check to see if old passwords authenticate.
SR 2	NFR	The system should use the PMS to authorize connections made from legacy applications.	
SR 2.1	FR	A function should authenticate connections from legacy/in-house applications using the database	Check if the function inputs are sanitized and it authenticates in a reasonable amount of time.
SR 3	NFR	A Master password system must be implemented so users don't have to remember application-specific passwords.	
SR 3.1	FR	A function should be provided to assign a random master password to a specific user.	Check if the credentials are stored in a database separate from the application specific passwords.
SR 3.2	FR	A function should be provided which sets all current user account status indicating that their password has expired	Check if the status update reflects automatically when policies update.
SR 3.3	FR	A function should be provided that allows a user to change his master password.	
SR 3.4	FR	A function must be provided that resets account status.	Check if the status update reflects automatically when policies update.
SR 3.5	FR	A function should return all application-specific passwords for a user.	Check if data is safe in transit and if correct application and password pairs are displayed.

manage changes to requirements, as well as to evaluate the impact of changes and track progress against the original plan. The use of a requirements table can help to improve communication, reduce misunderstandings and errors, and ultimately lead to a more successful outcome. Table 1 shows the requirements both functional and non functional which are associated with this project

4 USE CASE DIAGRAM

A relevant use case diagram would show all the actions that can be performed by the user and the admin. The user can login to the master password database, view application passwords and use that to login to the application. The user can aslso check leak passwords same as the admin and can reset their master passwords. The admin can create and update the policy, generate passwords, check passwords on their own, create new users and assign these passwords to them and reset their account state when requested for. They also have the option to delete a particular user record. Figure 2 shows all the respective usecases for each actor.

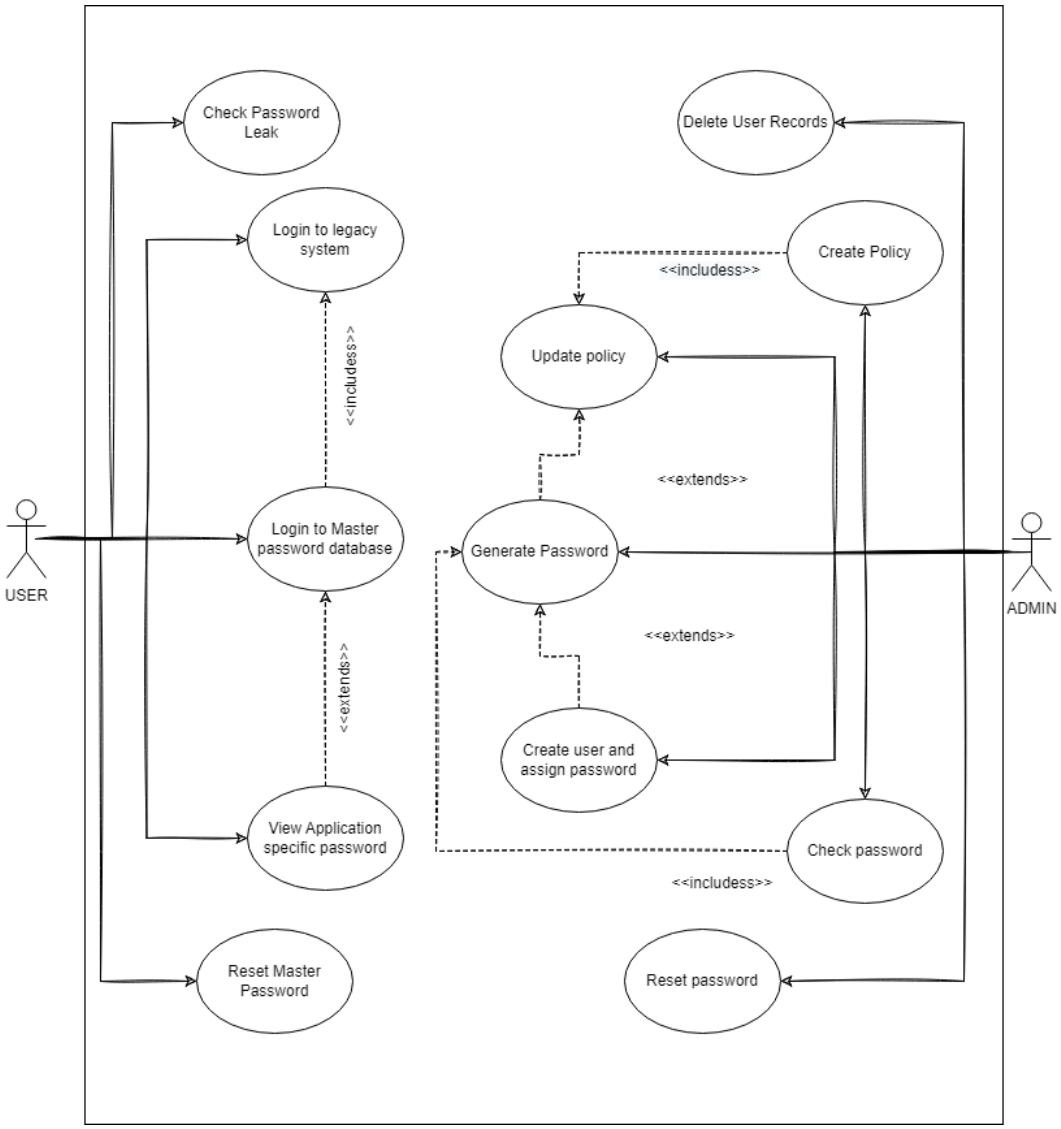

Fig. 2. Use Case Diagram

Use case diagrams also give us the opportunity to develop abuse and misuse case diagrams which show how these use cases can be abused by a threat actor and unintentionally misused by the admin or user. Figure 3 shows the abuse cases which include inserting records to gain access, and gaining access to the policy and password files before they are encrypted. On the user side, the attacker can work through social engineering or bypass authentication to gain access to their accounts.

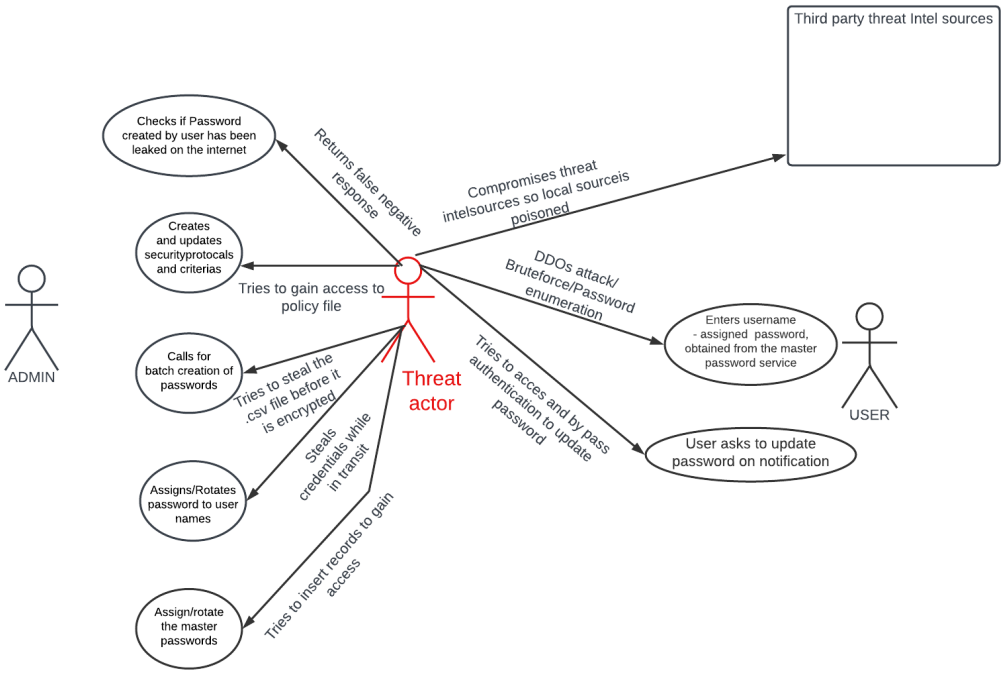

Fig. 3. Abuse case Diagram

Misuse cases are when the admin or the user intentionally or unintentionally misuses their privileges to create or exploit a vulnerability.

(1) Misuse cases by an administrator:
   (a) Intentionally changing or removing password policies: The administrator may change the password policy in a way that makes it easier for them to gain access to sensitive information.
   (b) Viewing or accessing confidential information: The administrator may use their privileges to view or access confidential information that they are not authorized to see.
   (c) Disabling user accounts: The administrator may disable user accounts without proper justification, which could result in disruptions to the system.

(2) Misuse cases by a user:
   (a) Sharing passwords: A user may share their password with unauthorized individuals, which could result in security breaches and unauthorized access to sensitive information.
   (b) Using weak or easily guessable passwords: A user may use weak or easily guessable passwords, which increases the risk of security breaches.

(c) Writing down passwords: A user may write down their passwords and leave them in an unsecured location, which increases the risk of security breaches.

5 SEQUENCE DIAGRAM

A sequence diagram is a type of interaction diagram that is used to model the interactions between objects or components in a system. It is a graphical representation of the flow of messages between objects over time, showing the sequence of messages, their source and destination, and the timing of the messages. Sequence diagrams are commonly used in the analysis and design phase of software development to help understand and communicate the behaviour of a system. They can also be used to model complex interactions and to identify potential issues that may arise during implementation. With their ability to clearly depict the order of events and the relationships between objects, sequence diagrams are a valuable tool for effectively communicating the design of a system to stakeholders and ensuring that all parties have a common understanding of the system's behaviour.[4]

The diagram consists of various objects on the horizontal axis and time in the vertical axis. Time in this diagram is not absolute but relative i.e. it describes the order or the "Sequence" of interactions between processes. Usually, objects are arranged left to right in the order in which they are called or executed. The rectangles on the diagram represent the life cycle of each function or hoe long they are relevant Figure 4 shows the sequence diagram for the Login process of a user.

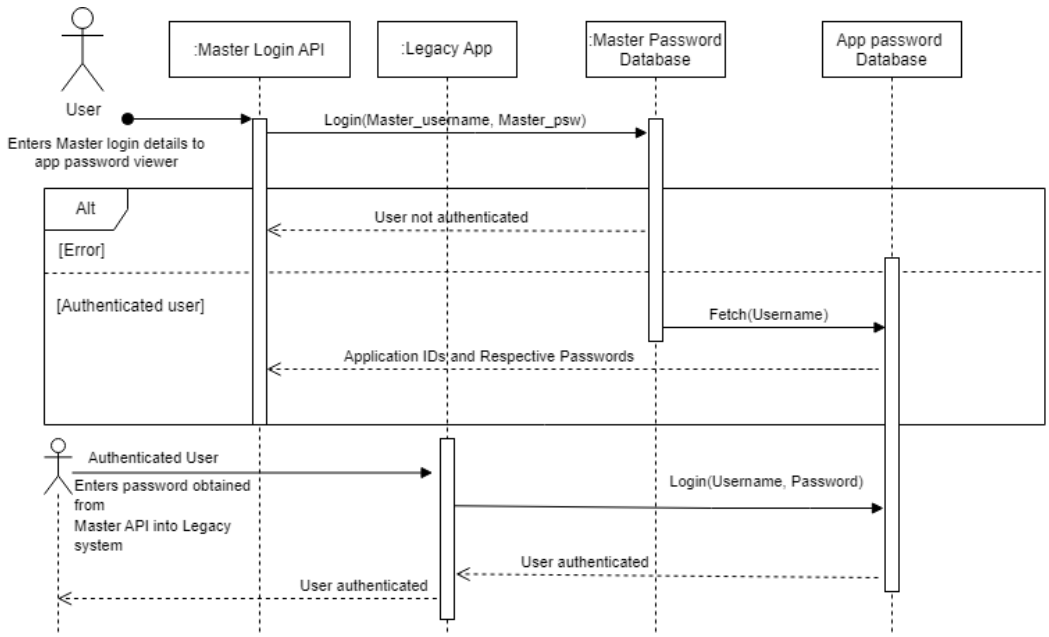

Fig. 4. Sequence Diagram for Login Process

The sequence diagram starts with the user providing their credentials to the master PMS API, which in turn cross-references it with the master password database and returns authentication results. If the user is not authenticated it returns an error statement. In this case, if it is authenticated. The user is provided with the application-specific password. The user then uses these credentials to log in to the legacy app, which in a similar fashion cross-references the data entered with the application-specific database and then the user is authenticated and logged in to the legacy system.

6 STATE MACHINE DIAGRAM

A state machine diagram, also known as a state diagram or statechart, is a type of behavior diagram that is used to model the behavior of a system. It is a graphical representation of a system's behavior over time, showing the different states that the system can be in, the events that trigger transitions between states, and the actions that result from those transitions. State machine diagrams are commonly used in the analysis and design phase of software development to help understand and communicate the behavior of a system[4]

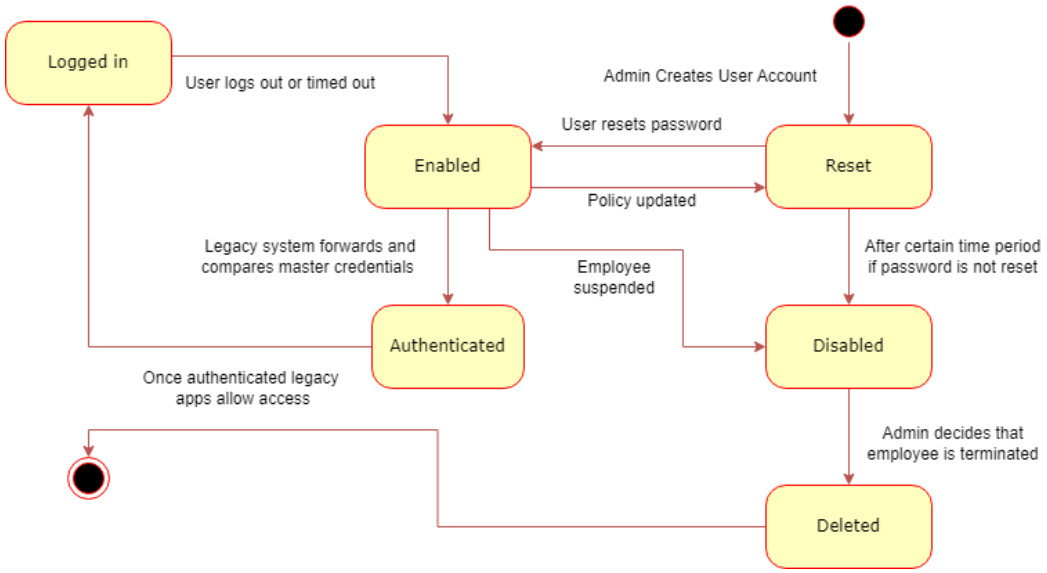

Fig. 5. State Machine Diagram for User Account

In a state machine diagram, states are represented by rounded rectangles and are labeled with the name of the state. Transitions between states are represented by arrows and are labeled with the event that triggers the transition and the resulting action. Each state represents a certain condition or stage in the system's life. Figure 5 shows the State diagram for the user account associated with the PMS. When the admin creates the user account, it is by default in the reset stage and when the user updates their password it moves into the enabled state. When application IDs and passwords are attached to the account it moves to a parallel and independent state called 'assigned app password'. When the user needs to log in, the account first moves to the authenticated stage then the logged-in stage and then the logged-out stage and then after logout, it returns back to the enabled stage. When the policy is updated, it moves back to the reset stage. If The user does not take action within the decided time period the account moves into the disabled stage and upon admin decision it moves to the deleted stage.

7 ACTIVITY DIAGRAM

Activity diagrams are a type of Unified Modeling Language (UML) diagram that show the flow of activities in a system. They are used to model the behavior of a system, including the sequence of activities and their interactions with one another.

Activity diagrams are drawn using symbols and flow lines. The symbols used in activity diagrams represent different elements of the system's behavior, such as actions, decisions, and events. The

flow lines in an activity diagram show the order in which activities occur and how they are connected to one another.

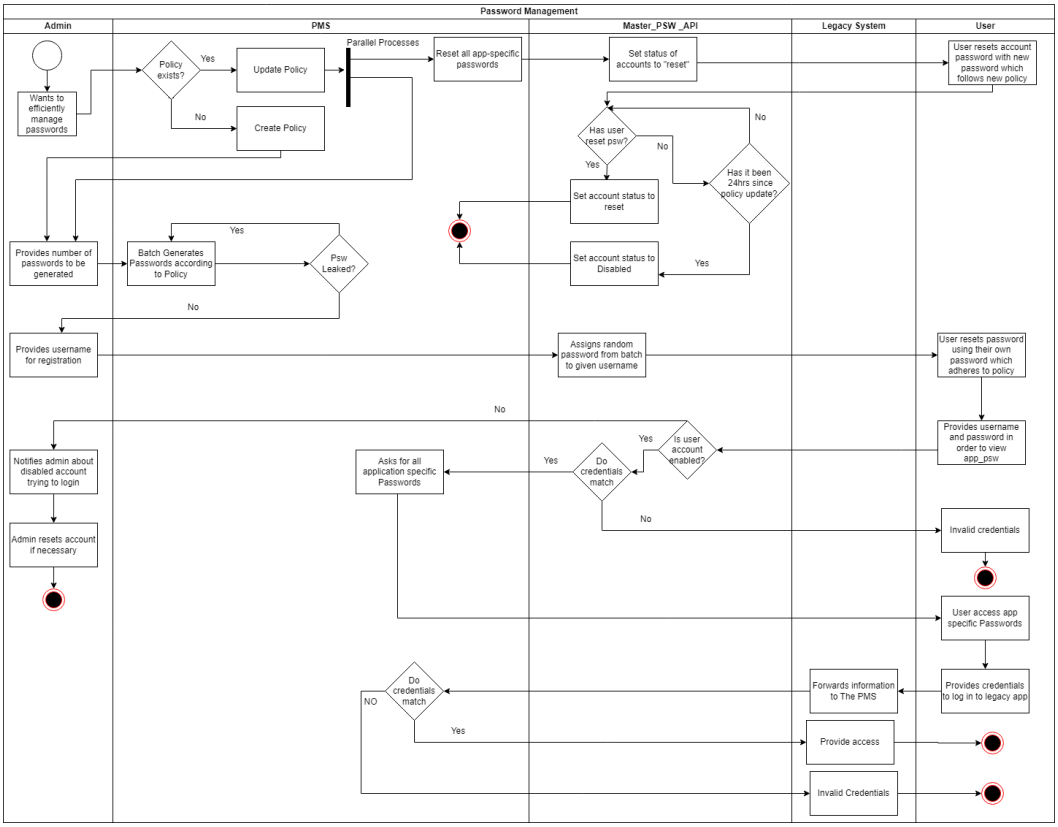

Fig. 6. Activity Diagram for the full PMS

Figure 6. Shows the activity diagram of the entire system. The control starts with the admin creating the policy if there exists no policy and updating one if it exists. Updating the process creates two parallel processes on which changes the states of the password from 'enabled' to 'reset' until the password fits the current policy or a user account is disabled. After the policy is created the control passes back to the admin who calls for creation of password, this process involves generating passwords according to the police and continuously checking if these passwords are leaked on the internet or not. If These passwords have been leaked then the process returns the number of leaked to the generator which compensates with new passwords and this continues until all passwords stored have passed the leak check.The passwords are then assigned to username and application IDs and stored in pairs. When the user tries to l;ogin they first have to go through the master login in order to view the application specific password and then login to the corresponding legacy app. The master login checks if the account is enabled and only then proceeds with the login. For any other state, control is transferred to the specific use-case. If the account is in 'reset' state then it will remind the user to reset their account. If it is in the disabled state the system sends a notification to the admin, about a disabled account trying to access the PMS. After obtaining the application specific passwords, the user can directly go to the respective legacy system and

log-in using the credentials. The legacy app forwards the credentials to the PMS specifically the application specific passwords database and cross-references.

8 CLASS DIAGRAM

Class diagrams are a graphical representation of a system's objects and their relationships to each other. Class diagrams are used to model the classes, attributes, and methods of a system, as well as the relationships between these classes.

In a class diagram, classes are represented by rectangles and are labeled with the class name. Attributes of a class are represented by oval shapes and are labeled with the attribute name and type. Methods of a class are represented by rectangular shapes and are labeled with the method name, parameters, and return type. Relationships between classes, such as inheritance and association, are represented by arrows connecting the classes. Fig 7 represents the class diagram for the PMS.

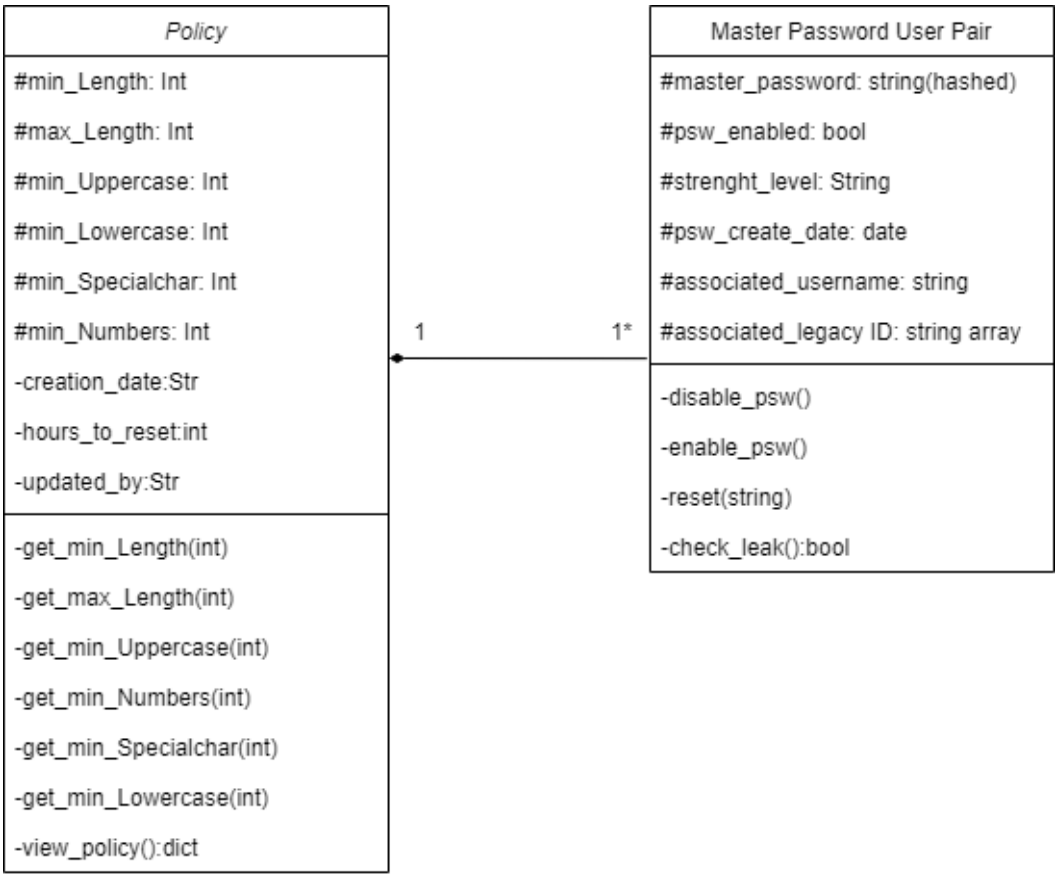

Fig. 7. Class Diagram for Password Policy and User Account

The PasswordPolicy class defines specific policies for passwords, such as minimum length, minimum number of uppercase letters, minimum number of lowercase letters, minimum number of digits, and minimum number of special characters.It has several attributes, including minLength, maxLength, minUppercase, minLowercase, minDigits, and minSpecial, which define the specific requirements of the password policy. For example, the minLength attribute defines the minimum

length that a password must be to meet this policy, while the minUppercase attribute defines the minimum number of uppercase letters that a password must contain. Methods include getMinLength, getMaxLength, getMinUppercase, getMinLowercase, getMinDigits, and getMinSpecial, which allow other parts of the system to retrieve the values of the policy's attributes. Additionally, the class may have other methods to implement other functionality specific to password policies, such as checking if a given password string meets the policy's requirements. The user password pair extends this class and adds a few methods on its own to enable diasable and reset the account, while having other attributes such as associated appIDs and username et cetera.

9 DATA STRUCTURES

In the project, the dominant data structures used are JSON for APIs and Python dictionaries. JSON (JavaScript Object Notation) is a lightweight data interchange format that is widely used for exchanging data between a client and a server. It is a text-based format that uses a simple syntax to represent data structures, such as objects, arrays, and values, making it easy to parse and generate.[3]

Python dictionaries are another commonly used data structure in the project. A dictionary is a collection of key-value pairs, where each key is unique and maps to a corresponding value. Dictionaries are dynamic, mutable, and unordered, making them a powerful tool for managing complex data structures in Python. They can be used to store and retrieve data in a structured manner, making them an ideal choice for tasks such as data modeling and data processing.

In this project, the use of JSON for APIs and Python dictionaries highlights the importance of choosing the right data structure for the task at hand. By using JSON for APIs, the project is able to take advantage of its widespread use and compatibility with many different programming languages, while using Python dictionaries allows for efficient and flexible data management within the Python environment.

Figure 8. shows the policy as a JSON file in the Postman Interface. This policy is sent as the input for the create policy API endpoint.

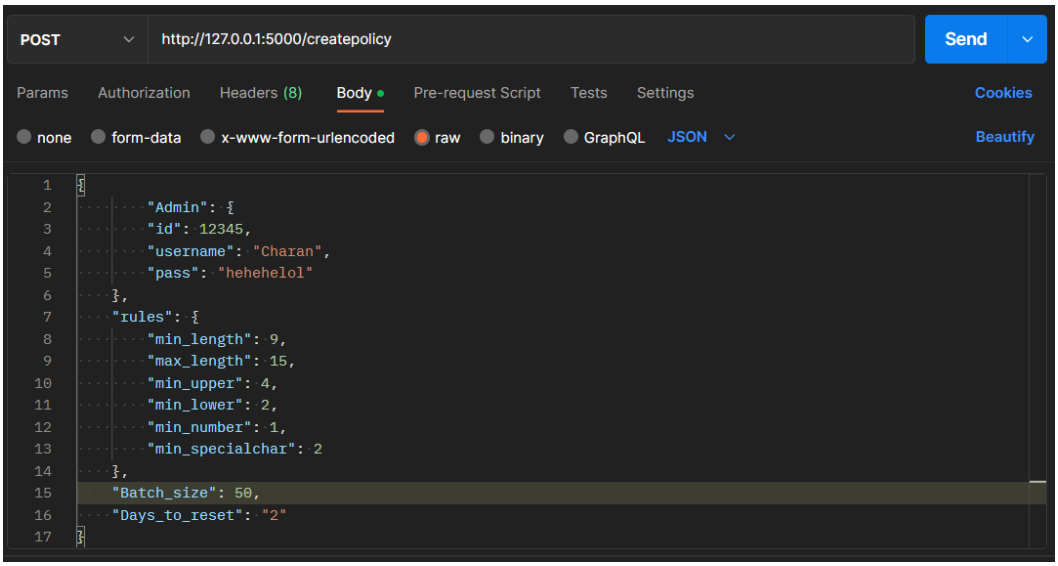

Fig. 8. JSON Data Structure Example for the PMS

10 REST API

REST (Representational State Transfer) APIs are a way of designing and implementing web services. They are based on the principles of the World Wide Web, and provide a standardized way of accessing and manipulating resources over the internet. In REST APIs, resources are identified by URIs (Uniform Resource Identifiers) and can be manipulated using standard HTTP methods, such as GET, POST, PUT, and DELETE. The HTTP methods correspond to CRUD (Create, Read, Update, Delete) operations on the resources. For example, a GET request to a resource URI retrieves information about the resource, while a POST request creates a new resource.

```python
def leak_check(pass1):
    # Hashing the input password using sha1 algorithm
    sha = hashlib.sha1(pass1.encode()).hexdigest()
    # Converting the hashed password to uppercase
    sha = sha.upper()
    # Extracting the splitting the hashed password
    query = sha[0:5]
    lastbit = sha[5:]

    # Requesting the data from the pwnedpassword api
    url = "https://api.pwnedpasswords.com/range/"
    url = url + query
    payload = {}
    headers = {}
    response = requests.request("GET", url, headers=headers, data=payload)

    # Split the received data into a list
    out_list = response.text.split('\r\n')
    # Creating a dictionary with the received data
    out_dict = {}
    for i in out_list:
        rem_hash = i.split(':')[0]
        leak_num = i.split(':')[1]
        out_dict[rem_hash] = leak_num

    # Checking if the last 5 characters of the password's hash in the out_dict
    if lastbit in out_dict.keys():
        return 1
    else:
        return 0
```

Fig. 9. Rest API Call from the PMS

REST APIs also use standard data formats, such as JSON or XML, to exchange data between the client and the server. This allows for easy interoperability between different systems and platforms.

Table 2. STRIDE Security Controls

TYPE OF THREAT	SECURITY CONTROL
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of privilege	Authorization

One of the key benefits of REST APIs is their ability to support statelessness. This means that each request to a REST API contains all the information necessary to complete the request, and does not rely on information stored on the server. This makes REST APIs very scalable and easy to implement in a distributed environment.

Have I Been Pwned (HIBP) is a website that allows you to check if your email address or password has been involved in a data breach. They provide a REST API that allows you to check whether a specific password has been included in any known data breaches. The REST API works by taking a SHA-1 hash of the password you want to check and sending it to the HIBP API. The API then returns a response indicating whether the password has been seen in a data breach. To protect the privacy of users, the HIBP API only returns the number of times the password has been seen in a breach, not the specific breaches in which the password was included. To use the HIBP password breach check REST API, one sends an HTTP GET request to the API endpoint with the SHA-1 hash of the password appended to the end of the URL. For example, to check a password with the SHA-1 hash of "5BAA61E4C9B93F3F0682250B6CF8331B7EE68FD8",one would send a request to the following URL:

https://api.pwnedpasswords.com/range/5BAA6

The API would then return a response indicating whether the password has been seen in a breach, in the form of a series of SHA-1 hashes and the number of times each hash has been seen. Then the requester can then compare the returned hashes to the original SHA-1 hash to determine whether the password has been involved in a breach. By using the HIBP password breach check REST API, you can check whether a password has been involved in a breach. The code for the request to mentioned API is shown as a screenshot in Figure 9.

11 STRIDE

STRIDE is a threat categorizing framework which can be implemented to systematically identify threats in an application in a structured and repeatable manner [1]. STRIDE classifies threats based on attacker goals, namely:

- Spoofing: The act of pretending to be someone else to gain unauthorized access to a system or its data.
- Tampering: The act of modifying data or the system itself for malicious purposes.
- Repudiation: The act of denying responsibility for a specific action, such as a transaction or message.
- Information Disclosure: The act of exposing sensitive or confidential information to unauthorized parties.
- Denial of Service (DoS): The act of making a system or network unavailable to its intended users by overloading it with traffic or malicious requests.

- Elevation of Privilege: The act of obtaining higher levels of access to a system than intended, such as bypassing normal security checks or elevating a user account to administrative privileges.

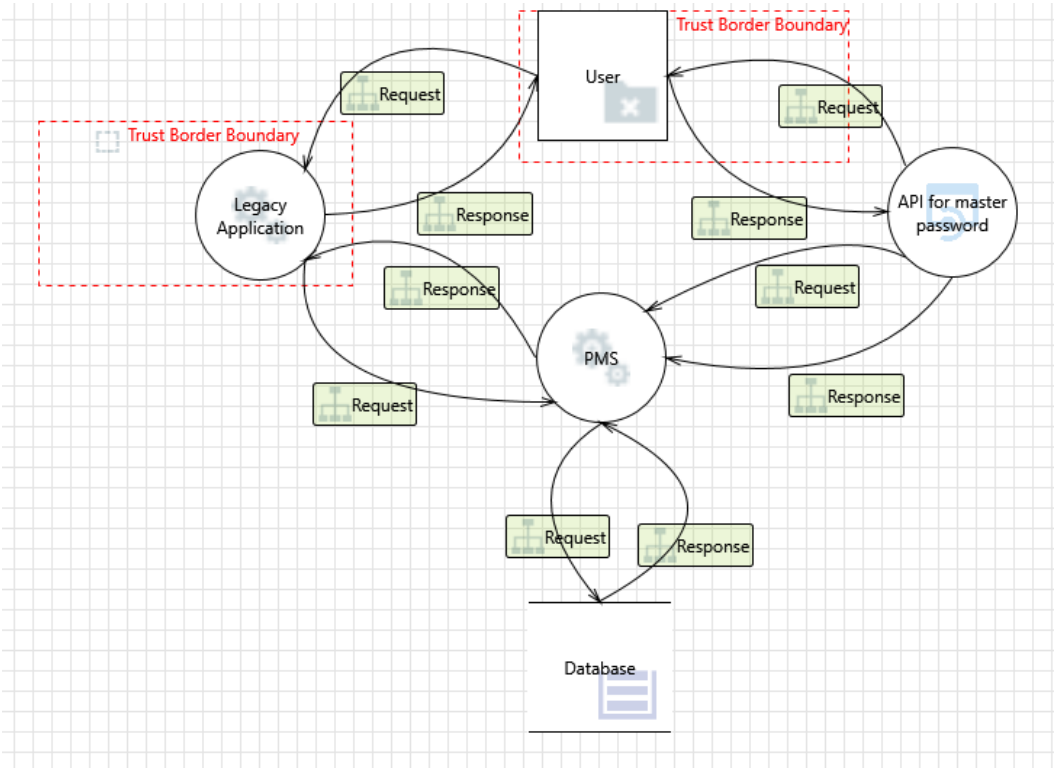

Fig. 10. STRIDE THREAT MODELING LAYER 1

STRIDE is used in conjunction with a model of the target system that can be constructed in parallel. This includes a full breakdown of processes, data stores, data flows, and trust boundaries [5]. The corresponding security controls for each specific threat type are given in Table 2.

STRIDE analysis is the process of applying the STRIDE framework to a system or application to identify potential security threats. The goal of STRIDE analysis is to provide a comprehensive, structured approach to identifying and assessing security risks, so that appropriate measures can be taken to mitigate them.

To perform a STRIDE analysis, a model of the target system is created, including a full breakdown of its processes, data stores, data flows, and trust boundaries. The next step is to use the STRIDE framework to systematically examine each element of the system and identify potential security threats that correspond to each of the six categories. Once these threats have been identified, they can be prioritized based on their potential impact and likelihood of occurrence, and appropriate security controls can be put in place to mitigate them.

STRIDE analysis is a valuable tool for software security professionals and developers, as it provides a systematic and repeatable method for identifying and mitigating security threats, and can help to ensure that applications and systems are designed with security in mind from the start.

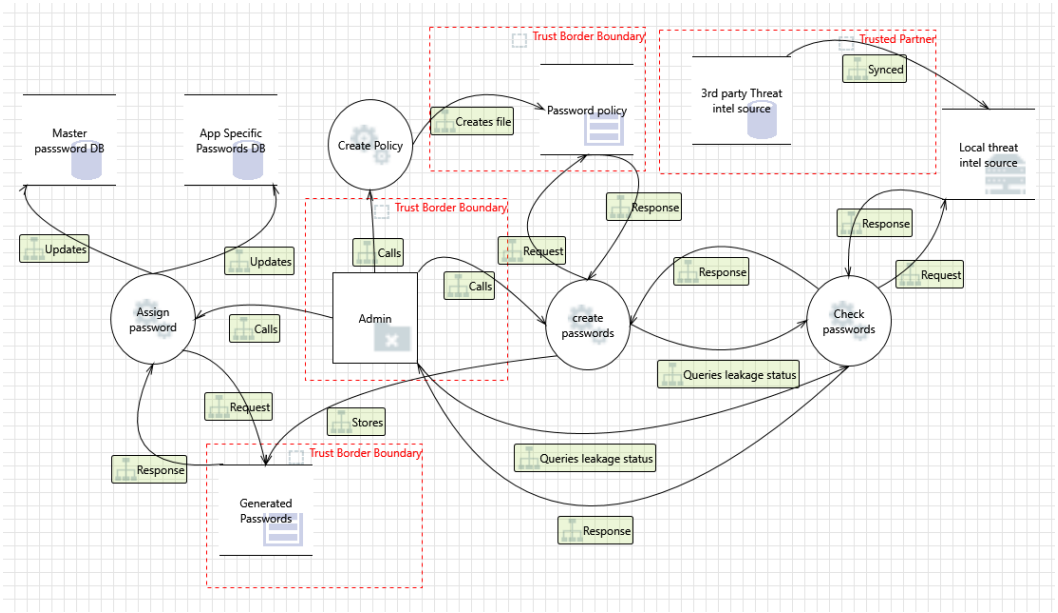

Fig. 11. STRIDE THREAT MODELING LAYER 2

Figure 10 shows the Data Flow Diagram on which STRIDE analysis is done using Microsoft threat modelling tool. Figure 11 shows the data flow diagram for layer 2 of the PMS, which is inside the PMS. Diagrams play a crucial role in STRIDE threat modeling as they provide a visual representation of the system and help to identify potential threats. A DFD shows the flow of data within a system and is used to identify potential information disclosure or tampering threats.By using these diagrams in combination with STRIDE, software designers and developers can get a comprehensive view of the potential security risks in their system and take appropriate measures to mitigate them.

The results are given in the following tables Table 3 and Table 4. The table 3 summarizes the various threats that have been recognized, along with the corresponding priority level. From the table, it can be inferred that the system has identified several high-priority threats to the Web Application and the database. The threats include Spoofing, Tampering, Repudiation, Information Disclosure, and Elevation of Privilege. The absence of proper authentication measures, lack of intrusion detection and prevention, insufficient auditing, and poor access control checks are some of the key factors that increase the risk of these threats. The high priority level indicates that these threats pose a significant risk to the security of the system. These threats are mostly out of scope for the system desing requested as the customer has asked us not to worry about the API security.

The table 4 lists recommendations to mitigate the high-priority threats identified in the system. For each threat, the table provides a list of mitigation measures that can be implemented to reduce the risk of the threat. For example, to mitigate the threat of Spoofing, the recommendation is to use standard authentication techniques to secure Web APIs. To mitigate Tampering, the recommendation is to enable threat detection and add digital signatures to critical database securables. To mitigate Repudiation, the recommendation is to ensure that logon auditing is enabled on SQL Server and auditing and logging on Web API. To mitigate Information Disclosure, the recommendation is to ensure proper exception handling, force all traffic over HTTPS connections, and encrypt

Table 3. Threats Recognized

Type	Threat	Priority level
Spoofing	If proper authentication is not in place, an adversary can spoof a source process or external entity and gain unauthorized access to the Web Application.	High
Tampering	An adversary may leverage the lack of intrusion detection and prevention of anomalous database activities and trigger anomalous traffic to the database. An adversary can also tamper critical database securables.	High
Repudiation	Attackers can deny a malicious act on an API leading to repudiation issues.An adversary can deny actions on a database due to lack of auditing.	High
Information Disclosure	An adversary can gain access to sensitive data through verbose error messages or sniffing.	High
Elevation of Privilege	An adversary may gain unauthorized access to Web API due to poor access control checks. Database access should be configured with roles and privileges based on least privilege and need to know principle.	High

Table 4. Mitigations Recommendation

Type	Mitigation
Spoofing	• Use standard authentication technique to secure Web APIs
Tampering	• Enable Threat detection • Add digital signature to critical database securables
Repudiation	• Ensure that logon auditing is enabled on SQL Server • Auditing and logging on Web API
Information Disclosure	• Ensure that proper exception handling is done • Force all traffic to Web APIs over HTTPS connection. • Encrypt sections of Web API's configuration files
Elevation of Privilege	• Implement proper authorization mechanism • Implement Row Level Security RLS to prevent users acce to others data • Sysadmin role should only have valid necessary users

sections of Web API's configuration files. To mitigate Elevation of Privilege, the recommendation is to implement proper authorization mechanisms, implement Row Level Security RLS, and limit the number of users with sysadmin role. Again all relevant mitigations were adopted and the rest were added to this document as an advise for the Customer when they integrate the PMS into their workflow.

12 CORE FUNCTIONALITY PROOF

The customer had requested a password management system with minimal delivered functionality. The system must have the ability to generate passwords based on a configurable policy which includes length and the number of upper/lower/special characters. In the case of policy configuration changes, existing passwords must be forced to renew. The passwords generated must be stored and evaluated as part of the authentication process. The services offered by the system must be made

available via REST/JSON API calls. Additionally, the system must make use of an external pwned password service. Only an example code is shown here while the complete application is available in the edugit project.

```
pass_list=[]
while len(pass_list)<(policy["Batch_size"]):
    psw = generate_password(csv_dict["min_length"],csv_dict["max_length"],
                            csv_dict["min_specialchar"], csv_dict["min_upper"],
                            csv_dict["min_lower"],csv_dict["min_number"])
    try:
        flag = leak_check(psw)
    except requests.exceptions.HTTPError as errh:
        return{"Http Error": str(errh)}
    except requests.exceptions.ConnectionError as errc:
        return {"Connection Error": str(errc)}
    except requests.exceptions.Timeout as errt:
        return {"Timeout Error": str(errt)}
    except requests.exceptions.RequestException as err:
        return {"Undefined API Error": str(err)}
    if flag==0:
        pass_list.append(psw)

csv_file_psw(pass_list)
```

Fig. 12. Password Batch Generation

The use of external pwned service is already shown in Figrue 9 that is also the function leak check, Figure 12 shows the batch generation of password code and the storage.

The function generate password takes 6 parameters: min length, max length, min special chars, min uppercase, min lowercase, and min number, which are used to configure the password policy. The length of the password is determined by randomly picking a number between min_length and max length. Then a random character is picked from the characters string to start the password. Special characters are added to the characters string to reach the desired count of min special chars. The rest of the characters in the characters string are randomly rearranged and added to the password. If the length of the password is less than randlen, additional characters are added to reach the desired length, and if the length is greater than randlen, the password is truncated to reach the desired length. Finally, the password is returned as the result of the function.

This looped for as many times as necessary until the batch size criteria is fulfilled. Passwords are only added if they pass the leak check. They are then written into a CSV to be assigned later on. Figure 13 shows the password generated according to the following policy:

- "min.length": 4
- "max.length": 6
- "min.upper": 1
- "min.lower": 1
- "min.number": 1

- "min.specialchar": 1

	Password (1)	Upper_case_chars (2)	Lower_case_chars (3)	Special_chars (4)	Digits (5)
1	Password	Upper_case_chars	Lower_case_chars	Special_chars	Digits
2	mK*XZ9(kLc.	4	3	3	1
3	AEk]a3*BoF@j	4	4	3	1
4	jCLGx4%X+LoU~T	7	3	3	1
5	ZU&x8a]BC4	4	2	2	2
6	2Se.JNwC.u/2#	4	3	4	2
7	B9%+gOTdDx^	4	3	3	1
8	o_C^Y6EFwb#PT(	6	3	4	1

Fig. 13. Password Batch Generation

13 STATIC CODE ANALYSIS

SonarQube is a code analysis tool that performs static analysis on your code to detect bugs, vulnerabilities, and code smells. It can be integrated with a variety of programming languages and can be run on-premises or in the cloud. The tool also provides metrics and reports on code quality and maintainability, making it very useful. The current existing codebase was run using sonarqube and the following outputs were obtained. The outputs indicated that my unit tests covered 27.9% of the existing code base. Which was enough for the core functionality. It had 31 code smells and 42 security hotspots out of which 2 were categorized as major. Sonarqube picked up some but not all of the expected hotspots. For example, my implementation had multiple instances of hard-coded passwords, the name of the file where the passwords and policy are stored and even a flags which can be bypassed but SonarQube didn't recognise them. It only recognized passwords that have the variable name "Password". Sonarqube revealed a few flaws that were apparent and some suggestions for code smells. It revealed no bugs. It also calculated that my debt was around 2.5 Hrs but the smells it suggested I implement very mostly naming issues and one would say that they take much less time especially when one uses an IDE which can refactor all instances of a certain variable name. The other high-security vulnerability reported was the use of a weak hashing algorithm and that was as expected. All the other code smells were provided live while coding by sonarlint, a linter based on Sonarqube. They could have entirely been avoided before the static code analysis.

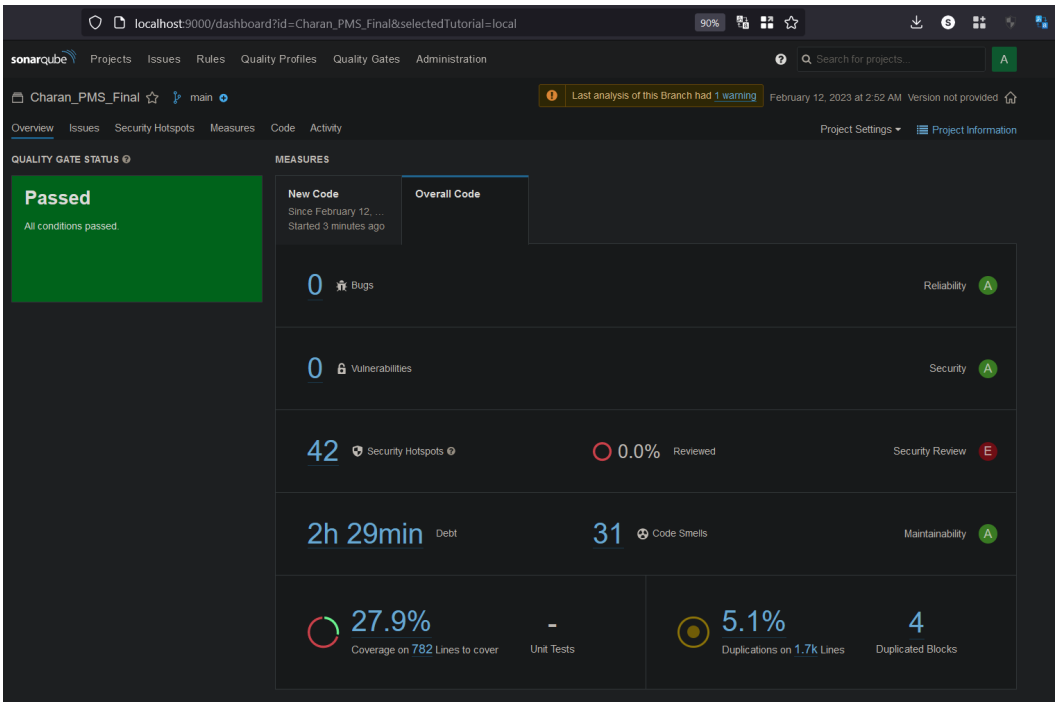

Fig. 14. Static Code Analysis PMS using SONARQUBE

14 CONCLUSION

In conclusion, after thorough testing and evaluation, it can be confidently stated that the developed software is secure. The security measures put in place, such as the implementation of Hashing algorithms and admin authentication, ensure the protection of sensitive information and the system against potential threats. Threat analysis has shown very few high-priority errors and most of them were mitigated. The use of UML diagrams removes any design flaw from the equation. Various exceptions were handled through out the implementation so that errors don't hinder the server. Additionally, the use of static code analysis tools has allowed for early detection and correction of any vulnerabilities in the code. The software has been designed with security in mind and all necessary steps have been taken to ensure its protection. As technology continues to evolve, the software will need to be regularly reviewed and updated to maintain its security.

ACKNOWLEDGMENTS

To Prof. Dr. Andreas Schaad for the opportunity. To Badrinath Kulkarni, Vaibhav Magar, Poojitha Lalagari and Jeet Ardeshna for their valuable input and to Deekshtia M for teaching me how to use latex.

REFERENCES

[1] Larry Conklin et al. 2022. Threat Modeling. owasp.org/www-community/Threat_Modeling_Process
[2] Hyperproof. 2022. Secure Software Development Best Practices. https://hyperproof.io/resource/secure-software-development-best-practices/
[3] json.org. 2021. JSON (JavaScript Object Notation). https://www.json.org/json-en.html [Online; accessed 11-February-2023].

[4] Anne R. Rémy A. 2003. Fundamental Modeling Concepts (FMC) Notation Reference. http://www.fmc-modeling.org/download/notation_reference/FMC-Notation_Reference.pdf
[5] Adam Shostack. 2014. *Threat Modeling: Designing for Security.* Wiley. 61–64 pages.