

UNIVERSITATEA POLITEHNICĂ DIN BUCUREȘTI  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE  
DEPARTAMENTUL DE CALCULATOARE



## PROIECT DE DIPLOMĂ

Manager de parole

Alin Călin Duțu

**Coordonator științific:**

As. drd. ing. Giorgiana-Violeta Vlasceanu

**BUCUREȘTI**

2023

UNIVERSITY POLITEHNICA OF BUCHAREST  
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS  
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT



## DIPLOMA PROJECT

Password Manager

Alin Călin Duțu

**Thesis advisor:**

As. drd. ing. Giorgia-Violeta Vlasceanu

**BUCHAREST**

2023

# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Problem . . . . .	1
1.3	Objectives . . . . .	1
1.3.1	Security objectives . . . . .	1
1.3.2	User oriented objective . . . . .	2
1.4	Proposed solution . . . . .	2
1.5	Obtained Results . . . . .	3
1.6	The Structure of the paper . . . . .	3
<b>2</b>	<b>Requirements Analysis/ Motivation</b>	<b>4</b>
2.1	Functional requirements . . . . .	4
2.1.1	Use cases . . . . .	4
2.2	Non-functional requirements . . . . .	5
<b>3</b>	<b>State of the Art</b>	<b>6</b>
3.1	1password . . . . .	6
3.2	LastPass . . . . .	6
3.3	Keepass Password Safe . . . . .	7
3.4	Comparison . . . . .	8
<b>4</b>	<b>Proposed Solution</b>	<b>9</b>
4.1	Password Manager . . . . .	10
4.2	Docker environment . . . . .	11
4.3	Application with authentication . . . . .	11
<b>5</b>	<b>Implementation Details</b>	<b>12</b>

5.1	Libraries used . . . . .	12
5.2	Encryption Protocol . . . . .	12
5.2.1	PBKDF2 . . . . .	13
5.2.2	Argon2 . . . . .	13
5.2.3	ASCII85 . . . . .	14
5.2.4	AES-GCM . . . . .	14
5.2.5	Encryption scheme . . . . .	14
5.3	Communication Protocol . . . . .	16
5.3.1	CORS . . . . .	16
5.3.2	STP . . . . .	18
5.3.3	HTTPS . . . . .	19
5.4	Application Functionalities . . . . .	19
5.4.1	Welcome Page . . . . .	19
5.4.2	Home Page . . . . .	22
5.4.3	Form Page . . . . .	25
5.4.4	QR Page . . . . .	27
5.4.5	Other functionalities . . . . .	28
5.5	Application Flow . . . . .	28
5.5.1	Startup Flow . . . . .	28
5.5.2	Main Flow . . . . .	29
5.6	Technical difficulties . . . . .	29
5.6.1	QR limited storage . . . . .	29
5.6.2	Encryption matching . . . . .	30
5.6.3	CORS and STP . . . . .	30
5.6.4	HTTPS certificate . . . . .	30
<b>6</b>	<b>Evaluation</b>	<b>31</b>
6.1	Password generation speed . . . . .	31
6.2	Generated Password Complexity . . . . .	32

6.3	Application Security . . . . .	33
6.3.1	Communication Encryption . . . . .	33
6.3.2	Brute-force resistance . . . . .	35
6.3.3	XSS attacks resistance . . . . .	36
6.4	Ease of Use . . . . .	36
<b>7</b>	<b>Conclusions</b>	<b>38</b>
7.1	Future development . . . . .	38
	<b>Bibliography</b>	<b>42</b>

## **SINOPSIS**

Parolele sunt cel mai comun instrument pentru a securiza accesul la conturile de pe Internet ale unui utilizator, dar în timp ce tehnologia progresează, din ce în ce mai multe aplicații de exploatare a securității sunt create, unele din ele fiind mai eficiente decât precedentele crescând standardele de securitate mai mult ca oricând. În consecință, companiile au impus politici de securitate mai stricte pentru a ține pasul, dar aplicarea unor măsuri atât de drastice a copleșit oamenii atât de mult, încât au fost forțați să folosească modalități mai puțin securizate de a ține minte parolele lor, cum ar fi scrisul pe hârtie sau într-un fișier text. Soluția propusă din această teză este un manager de parole simplu și elegant care va ajuta oamenii să genereze parole mai puternice și mai securizate folosind cei mai buni algoritmi criptografici ai momentului.

## **ABSTRACT**

Passwords are the most common tool to secure access to users' accounts on the Internet, but as technology progresses forward, more and more security exploiting applications are being created, some of them being more effective than the previous ones raising the security standards higher than ever before. Consequently, the companies have imposed stricter security policies to keep up with the new security standards. However, such drastic measures have overwhelmed people so much that forced them to use insecure ways to remember their passwords, such as writing them on paper or in a file. The solution proposed in this thesis is a simple and elegant password manager application that will help people generate stronger and more secure passwords using the best cryptography algorithms of the moment.

## **Special Thanks**

I would like to thank my project coordinator, Giorgiana-Violeta Vlasceanu, who permanently provided constant assistance, feedback, and improvements to achieve the best results.

# **1 INTRODUCTION**

## **1.1 Context**

Passwords are the most common tool to secure access to users' accounts on the Internet and since there are so many services used for communication, entertainment, productivity, health, and so on, one individual can hardly remember the passphrases for all accounts.

To solve this specific issue, password managers have been created with different security and storage standards, although it is arguable that most of them could not be vulnerable to security breaches or exploits.

## **1.2 Problem**

As technology progresses forward, more security exploits are being created, some of them being more effective than others raising the security standards higher than before. Online attacks such as the TOP 10 attacks presented by OWASP[1] or other types of exploits can raise the risk of potential security breaches, provided that the manager is vulnerable.

To mitigate the damage of data leaks, stricter security policies have been imposed to limit the possibility of hashes being easily cracked using brute-force or dictionary attacks. However, such drastic measures overwhelmed people so much that forced them to use insecure ways to remember their passwords[2].

## **1.3 Objectives**

This thesis aims to present a solution to these security concerns by providing a password manager that is easy to use and reliable security-wise, meaning that companies can relax their security policies and mitigate their password vulnerabilities by using this tool.

### **1.3.1 Security objectives**

The first and most important objectives are addressed for the security concerns raised earlier. The password manager should minimize the potential risks and vulnerabilities or make it as power and time-consuming for the attacker to make his attacks impossible to succeed.



The conclusion of the study and the penetration test reports should provide a clear view of how secure the password manager is. Depending on the attacking types we can formulate 3 objectives.

### **Online attacks objective**

The first objective of the password manager is to mitigate online attacks. The potential vulnerabilities which could be exploited by these attacks are related to the database usage and the application's internal communication implementation.

### **Offline attacks objective**

Moreover, offline attacks should be slowed down to make password hashes nearly impossible to crack, as a secondary objective. In the event of data leaks, the exposed password hashes should take an extreme amount of computational power and time to make them harder to crack. The encryption algorithm should be exceptionally complex to achieve the objective.

### **Application attacks objective**

In addition, the number of exploits found in the password manager's graphical interface and internal implementation must be reduced to a minimum.

### **1.3.2 User oriented objective**

Another essential objective refers to the application's ease of use. The graphical interface must be intuitive and easy to use so a user can have a natural feeling when using the manager.

## **1.4 Proposed solution**

The solution proposed by this thesis is an application with a client-server architecture where the graphical interface is powered on the client side and its implemented commands on the server side. Both environments will be presented as separate containers in Docker and their communication will be implemented using REST API.

The proposed algorithm for password encrypting is PassImg[3], a 3-parameter-based algorithm that ensures the required complexity. The multi-device synchronization mechanism will work based on generating and scanning QR codes.

## 1.5 Obtained Results

The obtained solution constitutes of a functioning system with an intuitive graphical user interface that allows the user to set up a new password configuration for a specific website, access passwords using the corresponding master password, and synchronize with multiple devices with QR codes.

## 1.6 The Structure of the paper

**The next chapter** emphasizes the motivation for developing the password manager and real use cases.

**Chapter 3** provides a rigorous analysis of the State of the Art, where similar products are analyzed and compared with the purposed solution.

**Chapter 4** describes the architecture of the proposed solution and motivates the decisions for the technologies used.

**Chapter 5** gives more details about the developing process which consists of encryption algorithms with their strengths and usage in the application, multiple protocols with a brief description and their purpose in the implementation, and a complete description of the client functionalities.

**Chapter 6** offers an in-depth analysis regarding the performance of the encryption algorithms and the security of the entire application and the complexity of its generated passwords providing and interpreting data obtained under various loads.

**The final chapter** presents a brief description of the proposed solution entirely, providing additional improvements for future development

## **2 REQUIREMENTS ANALYSIS/ MOTIVATION**

In the current times, as technology progresses forward the line between cybercriminal activity and security protection raises even higher as more attacks and more security protocols are created. However, some security protocols regarding the password-changing period are so strict they can engulf people into finding new comfortable ways to adapt, even if it means trading security.

The motivation for the project comes from companies that seek better protection for their processed data, a valuable resource and a target for hackers due to its sensitive nature, and from employees who work in the mentioned companies and who seek more comfortable security procedures.

So both categories require a tool that should solve the addressed problems. However, not only employees or companies can use the app, but anyone who wishes to use a secure tool for creating and managing passwords.

### **2.1 Functional requirements**

Of course, the actual requirements are vast and may vary depending on the working environment. Still, the main goal is targeted, to enhance security in a simplified procedure.

#### **2.1.1 Use cases**

The first use case, as in any password manager, is password creation. The user wants to create a new password, so he should access the specific menu and provide the required parameters to finish the process.

Moreover, the user wants to make it easier to remember his passwords. The app has to provide a mechanism that is required to ease the process but also offers a hard-to-guess password. A great model would be a system that asks for a master password to give the required password.

An additional use case covers the recovery process. The user wants to regain access to his account in case he forgot his password and to do that, he will need to access the corresponding password entry, and either use a new master password or create a new password configuration.

A final use case regards multi-device synchronization. The user wants to move its passwords to another device and to be able to do that he should select the corresponding options from the main menu and follow the instructions provided by the app to export or import the

configurations via QR codes.

## **2.2 Non-functional requirements**

The scalability is the amount of speed the execution gains when involving additional processors in parallel computing operations. In the project's case, the scalability should lower the time taken to use encrypting algorithms for generating the passwords.

Code legibility is an important aspect, too, as people who need a custom implementation of this app can easily modify the internal code or implement additional functionality to suit their needs. The code should respect the SOLID principles, as well.

The project documentation comes with comments in the source code for describing the internal behavior of the defined methods and with an overall behavior of the app which will be described in this document in the following chapters.

### 3 STATE OF THE ART

At the moment, there is a considerable amount of solutions for managing passwords, each one focusing on ease of usage, extra features, and security. Most of them can be used not only by individual users, but also by companies as password managers have implemented special features for enterprise usage. However, some of them, as security incidents appeared over the years[4], seem to sacrifice application security over ease of use provided by the new features. Let's analyze 3 popular password managers, what they offer, and how secure these password managers have been over the years:

#### 3.1 1password

1password is one of the better password managers with a good set of features such as auto-completion, code signature validation, and biometric access.

The 1password's security architecture[5][4] provides a detailed description of its security model which will take the master password, add locally-saved salt, and encrypt the ensemble with *PBKDF2-HMAC-SHA512* algorithm for a number of iterations, and the result together with a local data key will be encrypted using *HMAC-SHA256* encryption.

The resulting hash will be verified using the ensemble of a local and a server key combined by the *2SDK* protocol. The local information is provided from an encrypted local vault. This security model is an outstanding solution and not easy to crack.

The 1password application is compatible with Windows, Linux, Android, and IOS devices and offers a fair amount of features. However, reports from 2016 to 2020[4] showed that the application presented moderate to high-risk vulnerabilities.

For instance, in 2020, experts from the Independent Security Evaluators conducted an investigation where it was found that popular password managers would expose master passwords in plain text while their specific apps were running in the background, 1password included. In addition, 1password didn't limit the number of login attempts which is a required restriction for mitigating brute-force attacks.

#### 3.2 LastPass

LastPass is a popular password manager which stands out from the competition with its vast amount of features such as the Digital Security Dashboard, Data Breach Monitoring, and a

"Passwordless authenticator" which uses biometrics for creating complex master passwords to simplify the authentication process[6].

The security architecture of LastPass is also very good[4][6], as it relies on *PBKDF2-HMAC-SHA256* for encrypting the master password along with a username for a specified number of iterations, adding salt to the encrypted data in the last iteration. The result will be, then, encrypted with *AES-ECB* along with a pre-defined plain text. The local parameters are provided from a local vault encrypted with *AES-256* encryption algorithm.

LastPass offers compatibility with mainstream operating systems for PC and mobile which is good, however, its infrastructure had serious issues over the years.

This password manager was subject to flaws in one of its browser plugins which could expose credentials in plain text, a significant data breach that led later to credential spoofing attacks, and a critical zero-day flaw that allowed any remote attacker to compromise accounts completely[4].

### 3.3 Keepass Password Safe

Keepass is a strong and flexible open-source password manager whose main focus on security has increased its popularity over time. It is not as feature-rich as LastPass or 1password, but it offers flexibility in choosing the encrypting algorithms.

While LastPass and 1password offer one encryption solution, KeePass provides the users with the option to choose between 2 KDF algorithms: *AES-KDF* and *Argon2*, and between 3 database encryption algorithms: *AES-256*, *TwoFish* and *ChaCha20*, depending on the version used[7]. The security architecture of the KeePass password manager relies on either an online or local backup. The backups use an encrypted database that stores parameters required for password authentication in a header form, while the password verifier will be stored in a data form. The master password which will be introduced by the user will be encrypted twice with *SHA-256*. The result will be concatenated with a transform seed and encrypted with the chosen KDF algorithm. The result of the encryption will be encrypted again with *SHA-256* along with a master seed and then again with another KDF algorithm.

The official app is compatible only with Windows. Despite the Windows-only compatibility, the open-source community implemented unofficial versions of the app to run on other mainstream operating systems.

Keepass is a strong password manager application, but like other password managers, it had its vulnerabilities. In 2019 Keepass could expose data in plain text while running on background[4] and in 2014 Keepass local databases were subject to being cracked by the KeeFarce tool[8] which uses dll injection to infiltrate KeePass process and retrieve database information.

### 3.4 Comparison

For this section, Table 1 and Table 2 will show a brief comparison of the 3 solutions by encrypting algorithms they use and by environment susceptibility.

Password Manager	PBKDF2	Argon2	AES	Other Protocols
1password	✓			✓
LastPass	✓		✓	
Keepass		✓	✓	✓

Table 1: Encryption algorithm usage table

Password Manager	Online Attack Susceptible	Local Attack Susceptible
1password	✓	✓
LastPass	✓	
Keepass	✓	✓

Table 2: Environment vulnerability table

## 4 PROPOSED SOLUTION

The proposed solution is a password manager with strong encryption that can withstand online attacks such as Man in the Middle and Sensitive Data Exposure, and offline attacks such as Brute-force and Dictionary attacks.

In addition, the usual security policies regarding password changing interval and password complexity have strengthened for better security, but with the effect of overwhelming people[2]. People have to use more vulnerable ways for saving their passwords, so the proposed solution also aims to be a simple, easy, and user-friendly alternative that can be used either as a local application or as integrated into a working environment.

Looking from a general point of view, there can be 3 entities that perform different actions and interactions [Figure 1] to securely use the app:

- Password Manager
- Docker environment
- Application with authentication

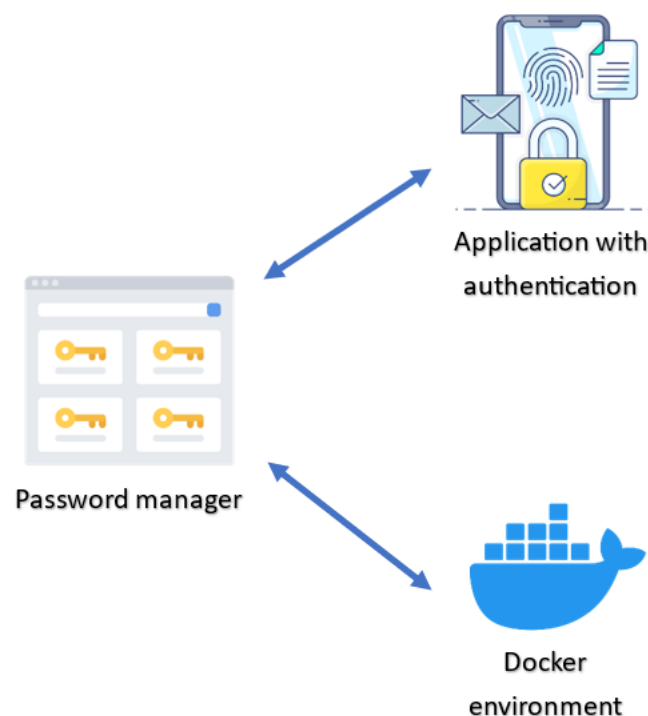


Figure 1: Application architecture <sup>1</sup>

<sup>1</sup><https://www.flaticon.com/icons>. Icons created by Smashingstocks, Freepik, and Nikita Golubev



## 4.1 Password Manager

The Password Manager will be the main application and will represent the implementation of the proposed solution. The application, itself, has 2 components: the client component, a user interface implemented in the Angular[9] framework with which the user will have access to the manager's capabilities by interacting with the interface elements, and the server component, an interface developed in Spring Boot framework[10] combined with the Maven[11] build tool which executes the operations called by the client.

The connection between the client and the server is secured by the REST API protocol. The client will send all the commands via HTTP requests, while the server will process them accordingly as soon as its endpoints will receive them [Figure 2].

Considering that the application runs on a local environment and the passwords generated by the manager will not be saved, there is no need for database storage. In addition, the app configuration will be stored in files as an encrypted string which will be sufficient for a secure local application.



Figure 2: Front-end interactions with Back-end <sup>2</sup>

<sup>2</sup><https://www.flaticon.com/icons>. Icons created by Freepik and FlatIcon

## **4.2 Docker environment**

Docker is a core part of the architecture. It is a deployment tool that offers a secured and isolated environment that will act as a host for the app components that will run as loosely coupled containers.

The containers will be configured using Docker files. For the server, the docker file will use Maven to compile the server interface. For the client, the docker file will make Angular build the user interface and start a server to host it. Finally, the docker-compose file will start and run both the client and server containers.

## **4.3 Application with authentication**

The application with authentication can be any application that accepts authentication with a password. After the user generates the password for the specific account, he has to copy it from the manager to the application and then log in.

In case the user doesn't remember the master password he can use a different one with the same password metadata, but he will have to reset his application account separately and add the generated password when changing it. String

## 5 IMPLEMENTATION DETAILS

The proposed solution mainly consists of an application composed of a client and a server. These components make use of different protocols which can ensure an efficient and secure inter-communication and a powerful encryption scheme along with meaningful functionalities which will be detailed in 4 sub-sections:

- Encryption Protocol
- Communication Protocol
- Application Functionalities
- Application Flow

### 5.1 Libraries used

- Server libraries
  - Spring Boot Starter - Spring functionalities[12]
  - Spring Boot Starter Security - Spring Security functionalities[13]
  - Lombok - Logging and Generators for Getters, Setters and Constructors[14]
  - MapStruct - Object Conversion[15]
  - JSON - String format for Java Objects[16]
  - Bouncy Castle - Implementation for cryptography algorithms[17]
  - Google ZXing - Creating and Reading QR Codes[18]
  - ASCII85 - Data format conversion[19]
  - JUnit - Testing Framework[20]
  - AssertJ - Assertion functions for JUnit tests[21]
- Client libraries
  - PrimeNG - Complex User Interface elements[22]
  - Bootstrap - Simple User Interface elements[23]
  - ngx-cookie-service - Service to gather and add cookies[24]
  - ngx-scanner-qrcode - Scanning, Reading and Reading QR Codes[25]

### 5.2 Encryption Protocol

For an efficient encryption protocol, the algorithms were chosen based on algorithm encryption complexity, encryption speed, and ease of use, so the following algorithms have been chosen.

### 5.2.1 PBKDF2

*Password-Based Key Derivation 2* or *PBKDF2*[26] is a key derivation algorithm that uses a pseudorandom function to derive keys. The pseudorandom function prevents cracking tools from easily guessing the hash even with the use of graphic processing units which reduces the rate of success significantly.

This algorithm generates derivation of keys faster than its predecessor *PBKDF1*[26]. However, it is outclassed by *Argon2* or *Script*[27] algorithms in terms of encryption complexity. The algorithm will be used for implementing a key feature in the encryption scheme. The configuration for PBKDF2 [Listing 5.1] will include the PBKDF2 secret key, which is a predefined string.

```
PBKDF2_HASH_ITERATIONS = 500;  
PBKDF2_HASH_SIZE = 256;
```

Listing 5.1: Configuration for PBKDF2

### 5.2.2 Argon2

*Argon2*[28] is a key derivation algorithm created to mitigate brute-force attacks along with other functionalities. It uses a memory-hard approach that not only reduces the chance of cracking the password hash by a large margin, it also mitigates brute-force attacks by having an impossible-to-parallelize manner as one stage requires the output from the previous one, and by reserving a specified amount of memory which means that it is resource and time-consuming. For password generation, it is considered to be one of the best algorithms at the moment which is why it will be used for this scope in the proposed solution.

The algorithm has the capability to use parallelism for encrypting which can be controlled by setting the right parameters in the configuration [Listing 5.2].

```
ARGON2_SALT_LENGTH = 0;  
ARGON2_HASH_ITERATIONS = 10;  
ARGON2_HASH_SIZE = 128;  
ARGON2_THREAD_NUM = 4;  
ARGON2_MEMORY = 65536;
```

Listing 5.2: Configuration for Argon2

### 5.2.3 ASCII85

*ASCII85* or *Base 85*[29] is a binary-to-text encoding used in PDF files and other environments such as software development. The encoding uses 85 ASCII characters to represent the binary data instead of the 64 characters that *Base 64*[30] provides. It is also more efficient in the conversion process as *Base 85* uses 5 ASCII characters for 4 bytes of binary data instead of 4 ASCII characters for 3 bytes. By providing 85 characters instead of 64 the diversity of the characters from the output will increase, which means a decrease in the rate of the cracking tools to guess the password.

### 5.2.4 AES-GCM

The Advanced Encryption Standard (*AES*) combined with Galois Counter mode (*GCM*) is an encryption algorithm that provides security, confidentiality, and speed[31]. *AES-GCM* is designed to be implemented efficiently in network communication with speeds of 10 gigabits per second and above because the latency it introduces and the cost it takes to run are close to a minimum, but it can also be used in software implementation [31]. It is the best amongst the other versions of *AES* in terms of security, confidentiality, and speed and one of the best along with *RSA*[32] and *Blowfish* [33]. This algorithm will not be used in the proposed solution in the generation scheme. Instead, it will be used for data encryption and decryption for the QR codes and configuration files.

### 5.2.5 Encryption scheme

The purpose of the encryption scheme is to create a combination of the previously presented algorithms and encryption standards to mitigate any brute-force attacks and create a password that is complex enough to be impossible to guess.

The encryption scheme used in this thesis is called *PassImg*[3] [Figure 3] and it generates a password based on the following inputs:

- User Image
- Metadata
- Master Password
- Password Length

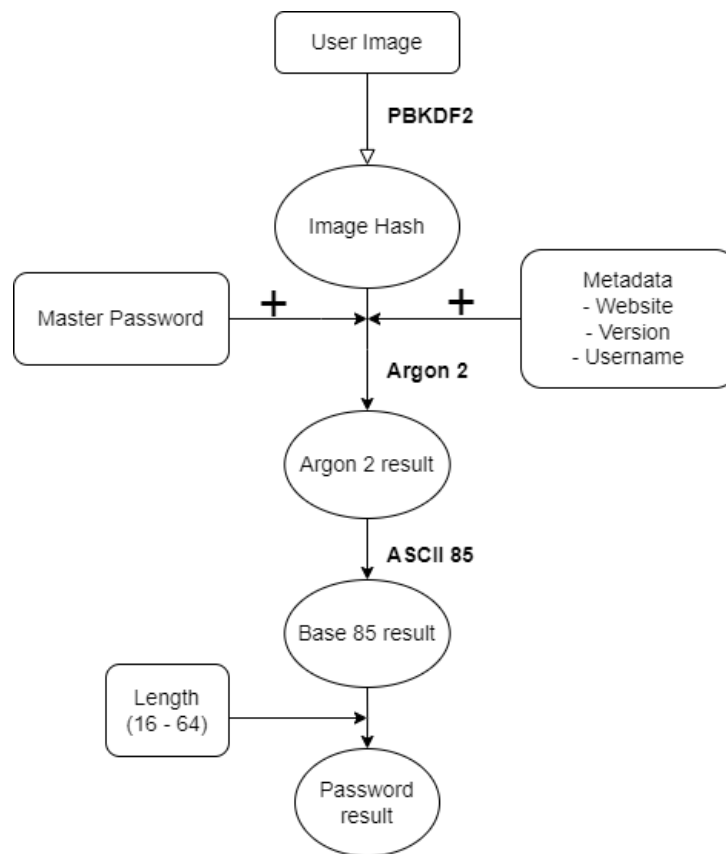


Figure 3: PassImg Encryption Scheme[3]

## User image

The User Image has an important role in password generation. Using the *PBKDF2* algorithm on the user image generates the image hash that will be used as input in *Argon2*. In the proposed solution the user has to upload the image only when he starts the application for the first time. The image hash will be saved in a configuration file that will be read at startup. Even if it is not recommended to use default system images or pictures downloaded from the Internet, the application can mitigate this vulnerability as the configuration settings for *PBKDF2* will generate different hashes for the same picture.

## Metadata

The password metadata is another input required for the *Argon2* algorithm and it consists of 3 fields: username, website, and version which can be completed with the desired values.

## Master password

The master password is the third input for *Argon2* algorithm and it is a mandatory text phrase. Here the user will insert a password regardless if it is weak or not as it will be used

for generating the secure password.

## **Password length**

The password length will be used in the final step which will extract a part of the Base 85 encoding result with the desired length.

## **5.3 Communication Protocol**

The purpose of the communication protocol is to guarantee a secure scheme for the transmission and reception of information between the client side and the server side of the application.

The communication is made possible by using the HTTP protocol implemented with the REST API[34] which ensures that data is correctly sent, transmitted, and received using a modern architecture model of communication between services.

To decrease the chances of attacks through the communication layer, the communication protocol is bundled with 3 security techniques:

- CORS
- STP
- AES-GCM

### **5.3.1 CORS**

The Cross-Origin Resource Sharing is a mechanism that uses HTTP headers to control cross-origin communication of the application by setting the server to share resources only with specified origins” [35]. The CORS mechanism, implemented in the server, will look for CORS-specific headers from every incoming request to check if a response can be shared cross-origin[Listing 5.3].

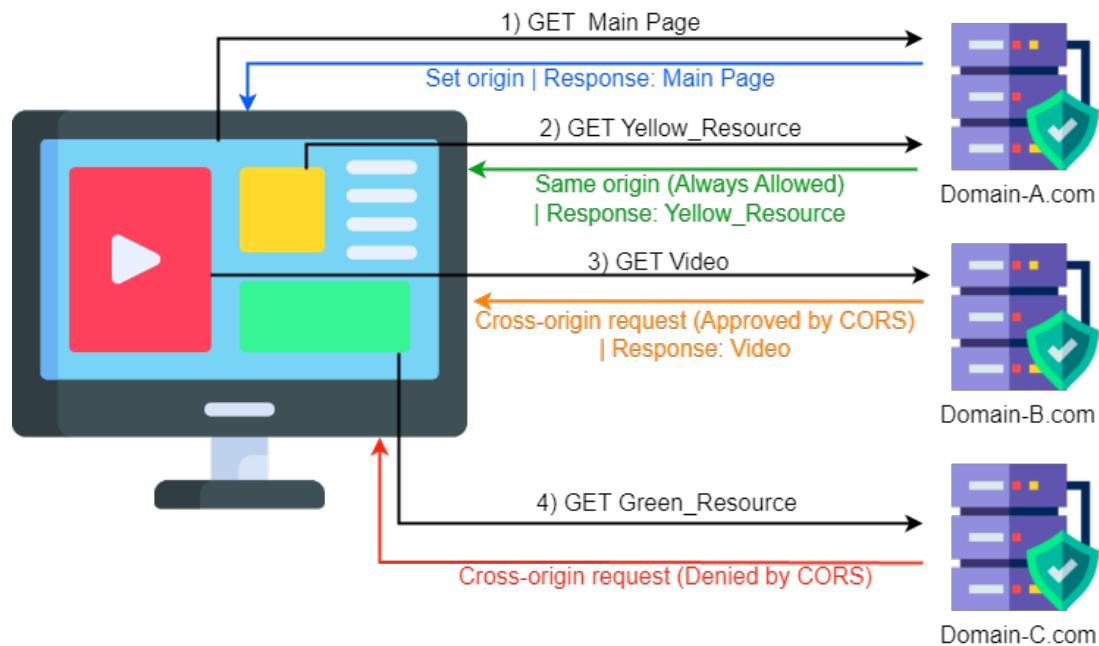


Figure 4: Cross Origin Resource Sharing flow <sup>1</sup>

The flow of the CORS mechanism is best described in the Figure 4 where the first request the client sends is to get the main page. This will set the origin of the client to the origin of the page, which is *Domain-A.com* until the client navigates to another page.

The second request asks for a page resource from the same domain as the client. When asking for a resource from the same origin as the client, the request will always be allowed.

The third request asks for a page resource from a different domain, in this case *Domain-B.com*. Here the CORS mechanism will come up and check the request for its headers to see if the request origin is in the allowed lists of origins configured in the domain server. The CORS will either allow the request to be processed as shown in the third situation or it will refuse it and send a response accordingly as *Domain-C.com* described in the fourth situation will do.

@Bean

```

public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.addAllowedOrigin("http://localhost:4200");
    configuration.addAllowedMethod("*");
    configuration.addAllowedHeader("*");
    configuration.addExposedHeader("*");
    configuration.setAllowCredentials(true);
    configuration.setMaxAge(3600L);
}
  
```

<sup>1</sup><https://www.flaticon.com/icons>. Icons created by Freepik and FlatIcon



```

    UrlBasedCorsConfigurationSource source =
        new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);
    return source;
}

```

Listing 5.3: Configuration for CORS

### 5.3.2 STP

The Synchronization Token Pattern[36] is a technique with the main purpose to mitigate Cross-Site Request Forgery attacks by creating a token for every request that modifies the application state. For instance, any request that creates, modifies, and deletes resources from the server, will trigger STP to create a new token which must be included in the specific requests to be accepted.

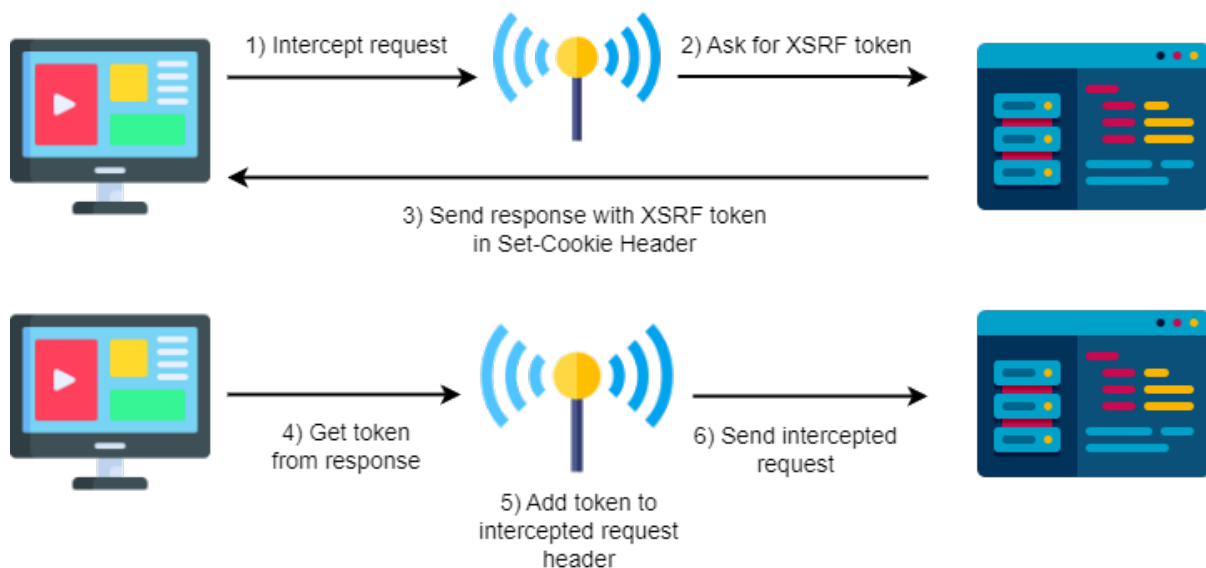


Figure 5: Synchronization Token Pattern flow <sup>2</sup>

The flow of *STP* [Figure 5] starts with the client service intercepting a state-changing request command that the client will send to the server. This will trigger the client interception method to ask the server for an XSRF token by calling a dedicated endpoint. The server will send a response that contains the special header *Set-Cookie* containing the *XSRF* token. The header will trigger the client's browser to create a cookie from where the *XSRF* token can be retrieved. As the token has been successfully gathered, the client will insert it as a new header to the intercepted request and then sends it to the server.

<sup>2</sup><https://www.flaticon.com/icons>. Icons created by Freepik and FlatIcon

### 5.3.3 HTTPS

To encrypt data in the communication line the *HTTPS* protocol is a great solution[37]. The idea is that *HTTPS* can be described as HTTP over *TLS* protocol, a cryptography protocol that uses symmetric encryption to ensure data privacy and integrity[38] which is a great addition to the proposed solution.

## 5.4 Application Functionalities

The proposed solution includes a diverse range of functionalities that can be effectively categorized according to the specific pages they are allocated within. There are 4 main pages:

- The Welcome Page
- The Home Page
- The Form Page
- The QR Page

### 5.4.1 Welcome Page

The Welcome Page is the first page that appears when the user opens the app. The app greets the user with a warm welcome followed by a loading screen where it notifies the user that the application is loading. The loading screen comes up with a progressive bar with which the user can relatively check the progress of the app startup process.

Behind the user interface, a request is sent to the server to retrieve the configuration file. If the configuration file is present and can be read by the server, the server will send a response with status 200 which will finish the startup procedure. However, if the configuration file is not present, the server will respond with a Not Found error and the user will have to choose between 2 options [Figure 6] to complete the configuration.

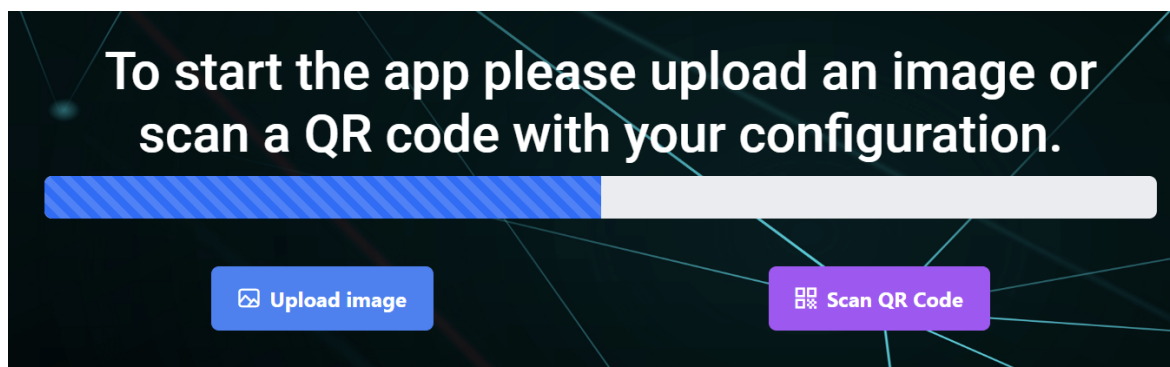


Figure 6: Options for a new configuration

## Upload image

The Upload Image option will open a dialog that contains:

- A preview square - Shows a preview of the selected image. At first, it will show an alternative message as there is no image uploaded
- Cancel button - Closes the dialog without uploading anything to the server
- Select button - Lets the user choose an image from his desktop for configuring the application
- Upload button - It will send the selected image to the server. If no image is selected, the upload button will be disabled until one is selected

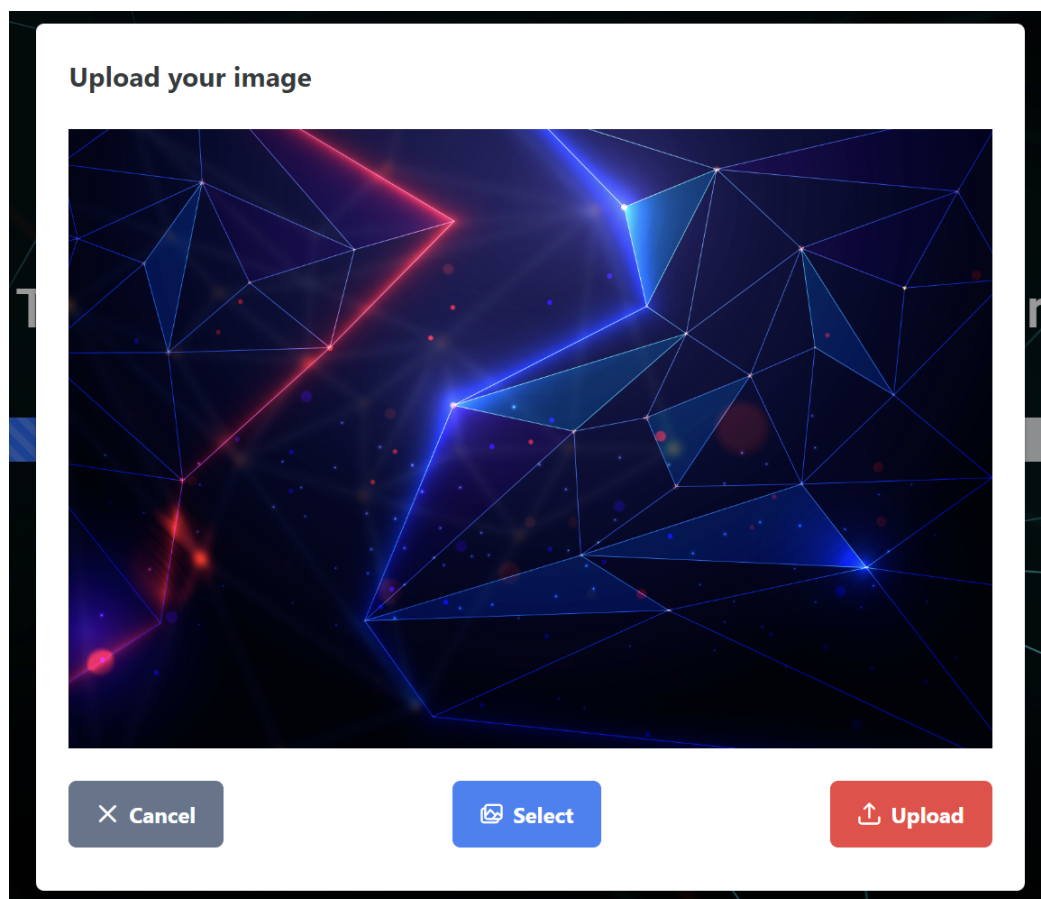


Figure 7: Upload image dialog after selecting an image

When the user selects an image, a preview of the selected image will appear in the dialog box, and the upload button will become enabled [Figure 7]. After the user uploads the image, the client will send it to the server endpoint. There, the server will get the image, encrypt the image to create the hash image required for password generation, and save it in the application configuration. The response given by the server will close the dialog on the client side and the startup process will finish.

## Scan QR Code

The QR codes[39] are used to transfer application configuration between users' devices. In the proposed solution, the QR code not only stores the hash image, but password configurations as well. The data stored is encrypted using AES-GCM.

The QR Scan option will open a dialog that will have:

- A video component - Opens one of the available cameras, the rear cameras being prioritized, and waits for the user to capture a QR code
- A dropdown component - It lets the user select the camera for scanning
- Cancel button - Closes the dialog without uploading anything to the server
- Rescan button - Only after a QR code has been scanned, the user can use the Rescan button to restart the scan
- Upload button - It will send the selected QR code to the server. Until a QR code is scanned, the upload button will remain disabled

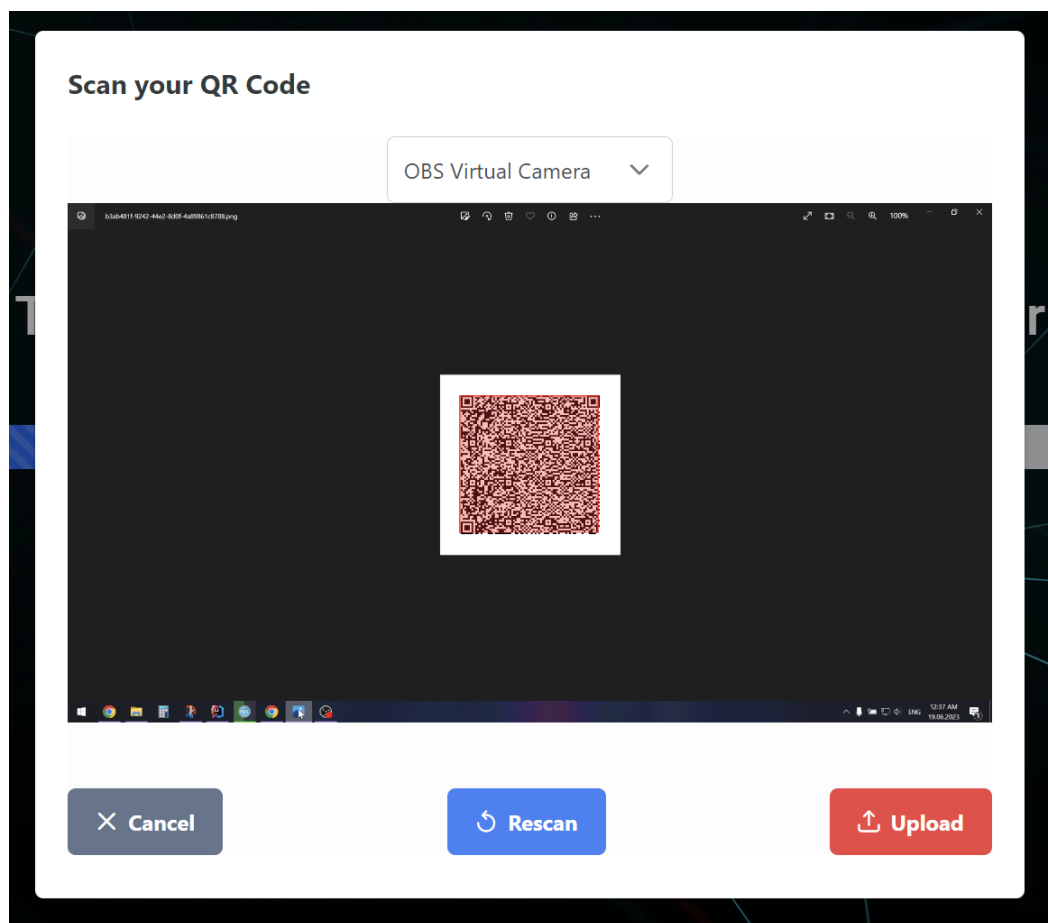


Figure 8: QR scanning dialog after a QR code has been scanned

When the user scans a QR Code, the scanner will stop the camera, will highlight the captured

QR code in red, and the upload button will become enabled [Figure 8]. By uploading the QR code, the client will send its data to the server's endpoint. There, the server will get the QR data, will decrypt it with AES-GCM, and parse it. The representation of the QR data is a concatenation between the image hash and the password configuration represented in JSON format[40]. The hash image will be extracted and saved, while the password configurations will be verified for duplicates and added to the list of password configurations. The response given by the server will close the dialog and will finish the startup process.

## 5.4.2 Home Page

The Home Page [Figure 9] is the main command point where the user will find a high diversity of functions at his disposal. In the upper part of the Home Page there is a navigation bar with 3 buttons: New Configuration, QR Menu and Help button, along with a panel that contains the password configuration entries which will be fetched from the server every time the user navigates or refreshes the page, a search bar for finding the preferred entries based on the username a website and a page menu at the bottom to easily navigate between pages.

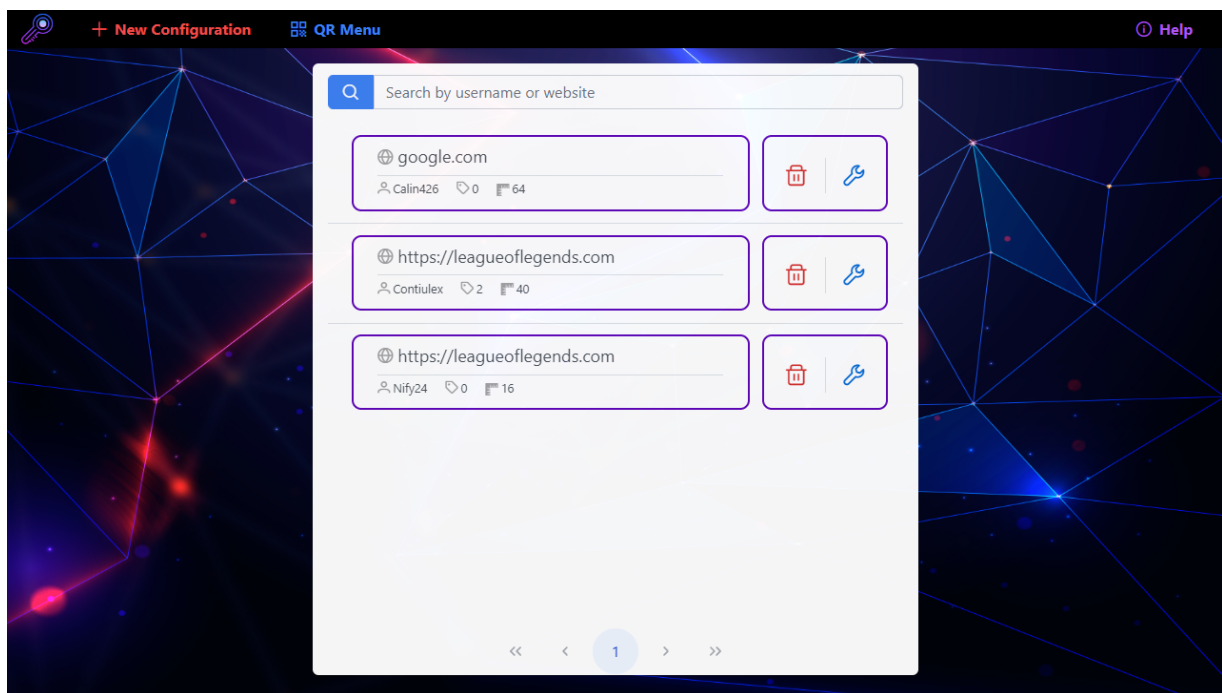


Figure 9: Home page

## Password configuration

A **Password Configuration** [Figure 10] is a set of metadata used not only to generate a specific password but to also identify the account associated with it. The password metadata

consists of a website, a username, a version number, and the length of the generated password.

A password configuration on the client side [Figure 10] will be represented by an entry that will have the metadata shown on the left side along with a list of 2 icons shown on the right side which represents the commands for that specific entry: the trash icon which deletes the entry and the wrench icon which will start the process of generating a new password.

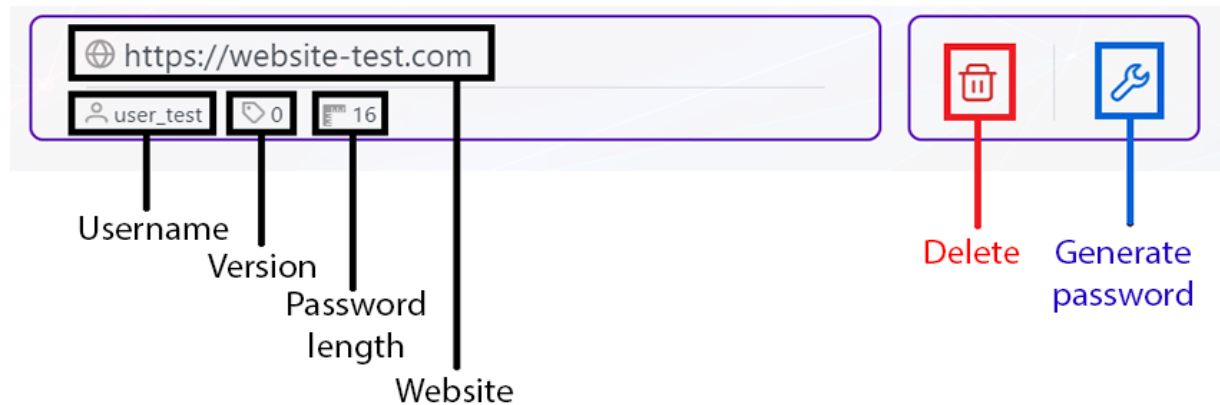


Figure 10: Password configuration

To create a new password configuration, the user has to press the "New Configuration" button which sends the user to the Form Page where he will have to complete a form with the required metadata.

To delete a password configuration, the user has to press the trash icon and then confirm the action. The server side will grab the corresponding id from the client side and it will delete the entry from the list if it exists and confirms the deletion through a 200 status response. If it doesn't exist then a bad request response will be sent to the client.

## Export to QR

The "Export to QR" functionality is the first option that appears in the "QR Menu". When pressed, it will switch the Home Page to a selective state where the user can select which entry he wants to be exported to the QR code [Figure 11]. In this state, the Home Page will have in the navigation bar 4 different buttons:

- Back button - When pressed it will cancel the exporting process and switch back to the normal state
- Select all - When pressed it will select all the password configuration entries from all the pages
- Help button - The helping button will give you instructions about what the user has to do to complete the exporting process

- Export button - It will export the image hash together with the selected password entries. If no password entry was selected, the client will export only the image hash.

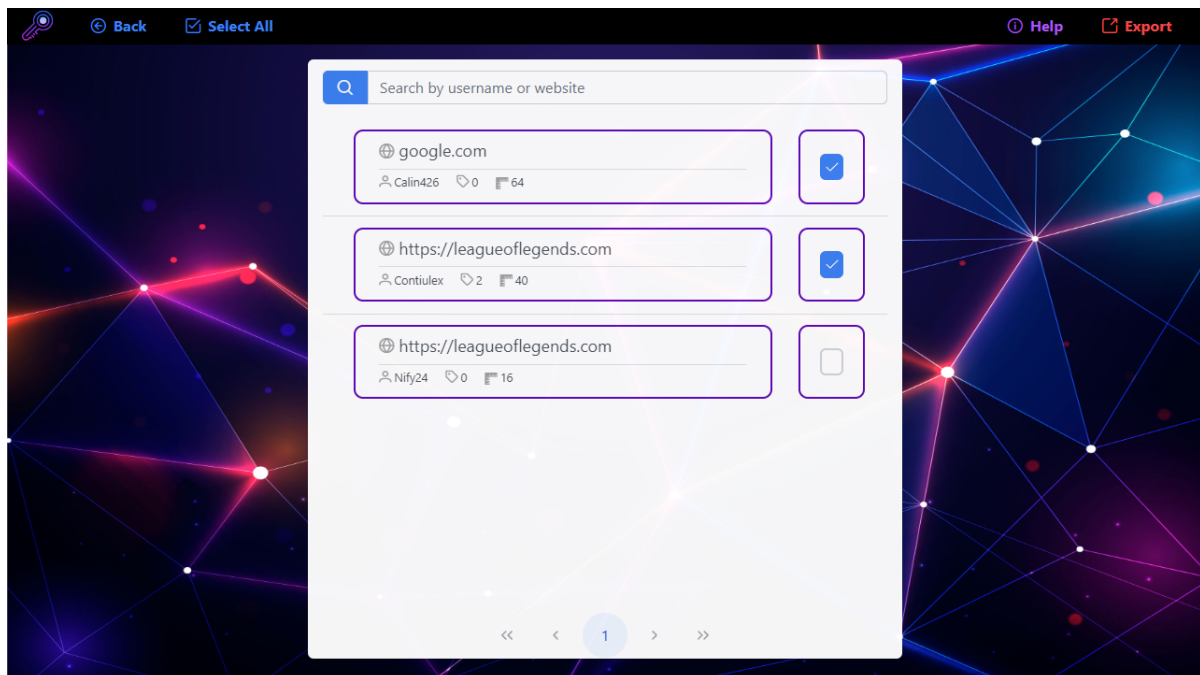


Figure 11: Home Page in QR Export mode

After the client exports the QR, the server will receive it through the endpoint the exporting password configurations, it will convert the list of configurations to JSON format[40] which will be concatenated with the image hash. The resulting string will be encrypted with AES-GCM and then converted to an image with the QR Code.

The generated image will be sent to the client and converted to a blob which will be processed by the QR Page. This QR code can be used for exporting a configuration from one device to another.

## Import from QR Code

To import a QR code from the Home Page the user has to select the "Import from QR Code" from the QR menu. This option will open the same QR scanner as the one on the Welcome Page. The user has the option to cancel, rescan and upload after a QR code has been scanned just like in the Welcome Page.

The differences can be seen on the server side. After receiving the QR data from the client, the server will start processing the data, but the configuration has been set already. For the server to import the passwords from the QR, the image hash from the app configuration must match the image hash from the QR code. Otherwise, the server will send an error response to the client.

## Generate Password

To generate a password the user has to press the wrench button from one of his password configurations, insert the master password in the panel's input and then send it to the client.

The master password input is a mandatory field that is restricted to have only alphanumeric characters along with underline and space characters.

The server will receive the master password along with the id of the check the master password again and if it passes, the input for the Argon2 encryption algorithm will be formed by the concatenation of password metadata, the stored image hash, and the master password.

The Argon2 algorithm will start to generate the password, meanwhile, in the client a dialog box will appear loading, waiting for the password to be generated. After being generated, the client will get the generated password from the server's response and show it in the dialog box [Figure 12]. In addition, after password generation, the dialog box will have a button that will copy the password to the clipboard.

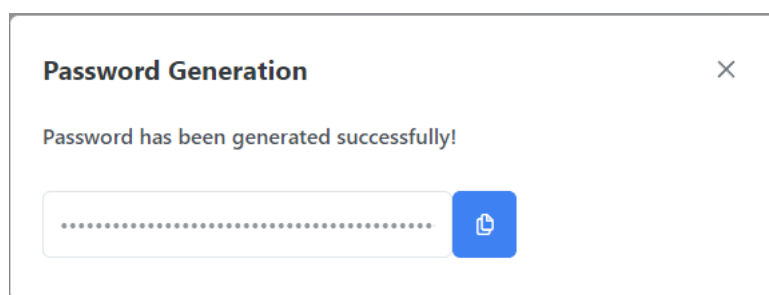


Figure 12: Password configuration form

### 5.4.3 Form Page

The Form Page [Figure 13] is used to create new password configurations. The user can reach this page by pressing the "New Configuration" button from the Home Page.

This page has a form with the metadata needed to be completed by the user along with a set of functional buttons. The form has the following fields:

- The username accepts a text input with a maximum of 20 characters, the first being a letter and the other characters being alphanumeric or the underline character. This field is not mandatory.
- The website is a mandatory field that must match the following pattern:

$[http[s] : //www.] < websitename > . < domain >$



where the parts of the pattern surrounded by arrows are mandatory and have to be filled with alphanumeric characters and the parts in the brackets are not mandatory.

- The version is a mandatory number field with the only restriction that it must not be a negative number.
- The length is a mandatory number field that will give the length of the generated password. The value can be set by using the slider.

The form is titled "Create a new password" and is enclosed in a light gray border. It contains the following elements:

- A "Username" text input field.
- A "Website" text input field with a placeholder text: "Make sure your website has the following pattern "[http[s]://www.]<website name>.<domain>".
- A "Version" text input field with the value "0".
- A "Length" slider control with a blue handle and a range from 0 to 100.
- A "Cancel" button (gray) and a "Submit" button (red) at the bottom.

Figure 13: Password configuration form

The form's buttons consist of the Cancel button with which the user will cancel the procedure and go back to the Home Page and the Submit button which will make the client check the input values introduced by the user. If the requirements are not met, the client will highlight the fields with a red outline and show the corresponding messages.

If all the fields pass the check, the client will send the metadata to the server. The server will receive the metadata, will make the same input checks as the client did, and then check for duplicates in the list.

If there is a duplicate, the server will return a bad request exception, but there is an exception to the rule for password configurations with no usernames since a person can have multiple

accounts on one platform, but doesn't want to add usernames to the configurations. In this specific case, the version will increase until there is no password with the same metadata.

If all the checks are passed successfully the password configuration will be added to the list of configurations.

#### 5.4.4 QR Page

The QR Page [Figure 14] will show to the user after he has completed the exporting process on the home page. As soon as the server has processed the qr data and has sent it to the client the image from the response will be interpreted as a blob that will be sent to the QR page.

The QR page will show a preview of the QR code together with a Download button that will let the user save the QR code on his device as a PNG image and the Home button which will send the user back to the Home page.

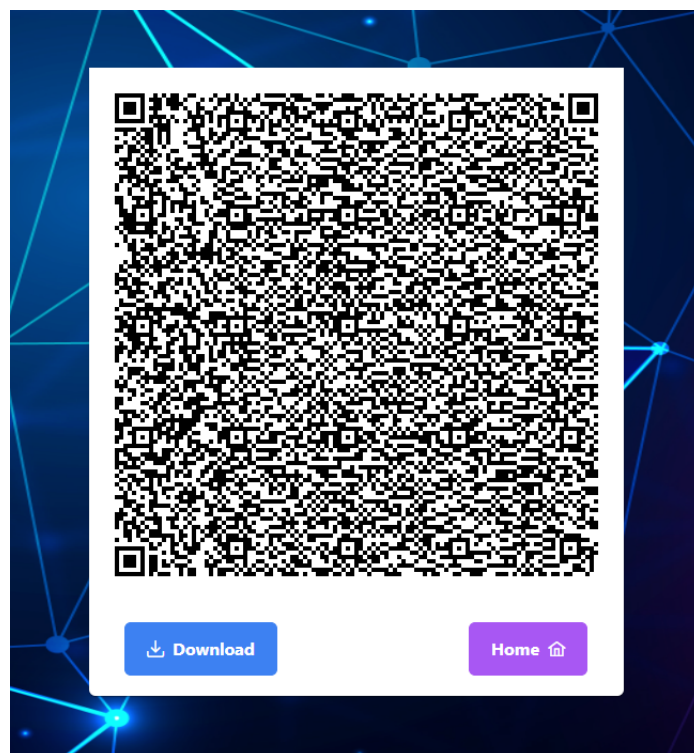


Figure 14: QR Page

## 5.4.5 Other functionalities

### Configuration saving

The application configuration will be saved when the user will close the app. This action will trigger the event that will send a request to the server to save the app configuration to the configuration file.

The server will concatenate the image hash with the list of password configurations converted to JSON format[40], encrypt the string using AES-GCM, and then save the result in a configuration file in the server's storage.

### Not Found Page

If the URL path doesn't match with any of the pages, the app will send the user to the Not Found Page. The Not Found Page has a message that emphasizes the error and a button to go back to the previous page.

## 5.5 Application Flow

In this section, 2 diagrams representing the User Flows for Startup Process [Figure 15] and Main functionalities [Figure 16] will be presented.

### 5.5.1 Startup Flow

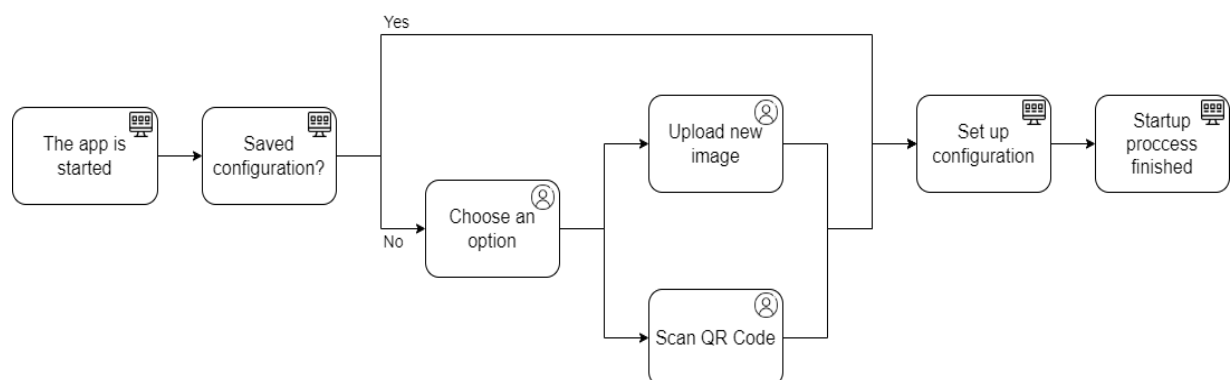


Figure 15: Startup flow diagram

## 5.5.2 Main Flow

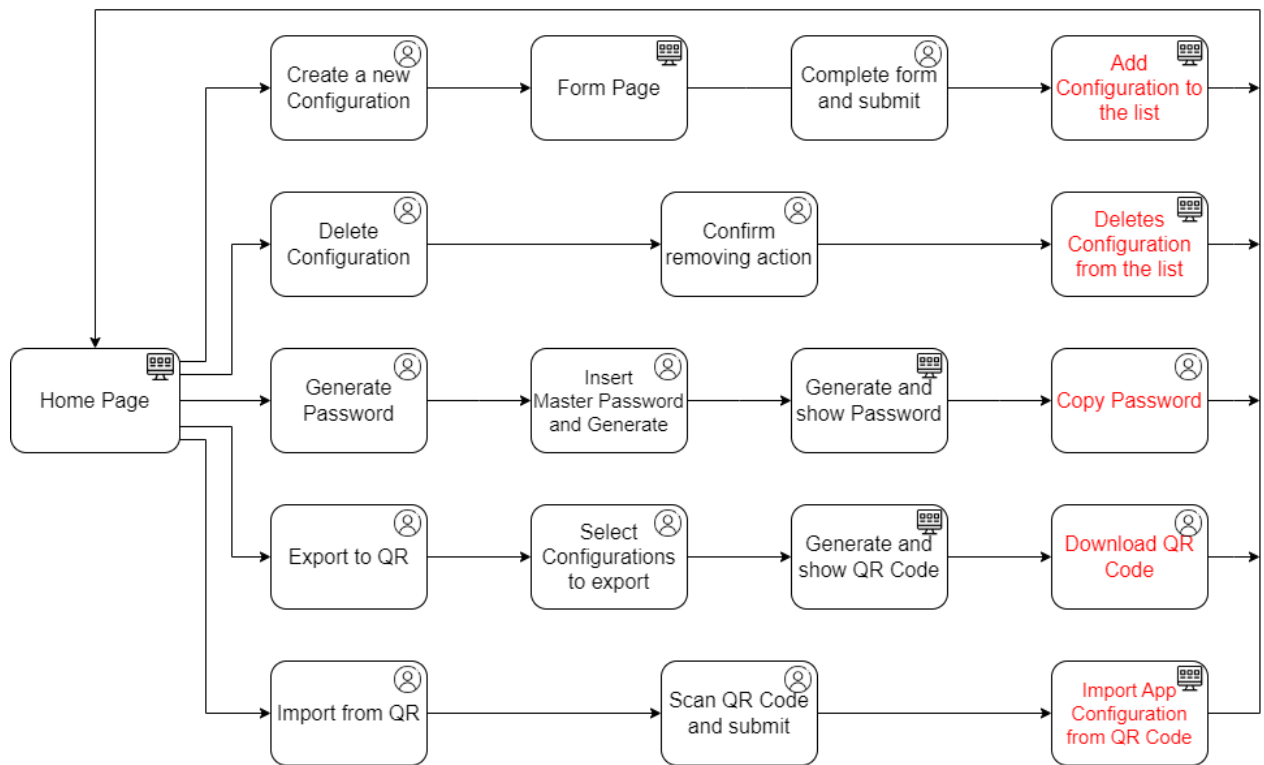


Figure 16: Main Flow diagram

## 5.6 Technical difficulties

### 5.6.1 QR limited storage

Since the QR code is limited to 4235 alphanumeric characters, there might be a possibility that the amount of exported data might exceed the storage of a QR code which could lead to creating more than 1 QR code for one configuration.

At the moment of discovering the issue, the app could support QR importing only from the Welcome Page, meaning that only 1 QR code could be read and the password configurations from it. The other password configurations had to be added manually.

To solve this issue, an import button has been added to the Home Page. This option can scan other QR codes that share the same image hash as the application. So, with this solution, different password configurations from different QR codes can be imported, with the condition that they come from the same device.

## 5.6.2 Encryption matching

When implementing the password generation, the Argon2 algorithm required a salt that was set to a specific value. It turned out that by generating a new password using the same inputs, the results were different and I didn't notice until I tested the algorithm outputs.

Hadn't I noticed the problem through testing, it would have been much harder to fix it in a future version of the implementation.

## 5.6.3 CORS and STP

The implementation of the encryption and key derivation algorithms required the addition of Spring Security dependencies. However, by adding this dependency the requests between the client and the server were blocked by CORS policy.

My first attempt was to disable CORS policies and CSRF protocol, but these options failed as Spring couldn't completely disable CORS policy. The way Spring Security works, by default, is to refuse any request that passes through the filter chains.

So, to let specific requests pass through, CORS and CSRF procedures had to be configured to accept the specific requests.

## 5.6.4 HTTPS certificate

To secure communication between the client and the server, the HTTPS protocol was implemented. However, the main requirement for this protocol was an SSL Certificate.

Because this is a local application the certificate authorities cannot create an authorized certificate since the localhost is not a website domain so there was no other solution than create a self-signed certificate.

The main problem with a self-signed certificate is that websites won't allow applications with HTTPS implemented to send or receive requests and will show a warning screen before accessing the server from the browser unless it is an authorized certificate.

The only workaround possible is to let the browser know that this is a personal certificate that it should trust by adding it to a special list of certificates, but this only solves the communication problem. Accessing the server's endpoint from the browser will still show the warning.

## 6 EVALUATION

For evaluating the proposed solution, the following metrics have been considered:

- Encryption Speed
- Generated Password Complexity
- Application Security
- Ease of use

### 6.1 Password generation speed

The generation speed of the proposed solution was measured using unit tests created with JUnit framework[20] and integration tests performed with Postman[41] by considering the execution time of the password generation requests.

The measurement of speed will be taken by setting up the same metadata and the same master password for all password managers. The following results have been obtained:

Password Manager	First Run(s)	Second Run(s)	Third Run(s)	Time Average(s)
<b>1password</b>	0,82	0,77	0,74	0,78
<b>LastPass</b>	1,08	0,88	0,94	0,97
<b>Keepass</b>	1,10	0,97	1,04	1,04
<b>Proposed solution</b>	2,05	1,80	1,86	1,90

Table 3: Password generation times

The results [Table 3] obtained make sense. Since the password managers discussed in the State of The Art, use the database storing option it is normal to retrieve the password faster. In the proposed solution, the password is generated every time, while the other managers generate the password once, save it in the database and then retrieve the value when it is needed.

## 6.2 Generated Password Complexity

The complexity of a password is strongly correlated with the time it takes to be cracked. A simple, weak password is an easy guess for cracking tools, while a long password with a big diversity of characters will take an unreasonable amount of time to crack.

To measure the complexity of the generated passwords, 3 passwords with the same meta-data and specified lengths of 16, 32, and 48 respectively will be generated in each password manager. The resulting passwords will be introduced in password strength tests from Comparitech[42] and UIC[43] websites which will measure the strength through an estimate of the time required for cracking or by assigning points for specific rules that the passwords respect.

Please note that each password test is using a model of strength estimation different from another so the results between the tests may vary, but the main goal is to compare the strength of the passwords generated by the password managers.

Password Manager	Comparitech (years)	UIC (points)
<b>1password</b>	$28 * 10^{12}$	123
<b>LastPass</b>	$5 * 10^{12}$	149
<b>KeePass</b>	$165 * 10^{12}$	172
<b>Proposed solution</b>	$165 * 10^{12}$	205

Table 4: Password strength comparison (length 16)

Password Manager	Comparitech (years)	UIC (points)
<b>1password</b>	$9 * 10^{45}$	343
<b>LastPass</b>	$7 * 10^{42}$	301
<b>KeePass</b>	$9 * 10^{45}$	337
<b>Proposed solution</b>	$9 * 10^{45}$	319

Table 5: Password strength comparison (length 32)

Password Manager	Comparitech (years)	UIC (points)
<b>1password</b>	$447 * 10^{75}$	465
<b>LastPass</b>	$11 * 10^{72}$	366
<b>KeePass</b>	$447 * 10^{75}$	391
<b>Proposed solution</b>	$447 * 10^{75}$	507

Table 6: Password strength comparison (length 48)

From the presented results [Table 4, Table 5, Table 6], the proposed solution performs well in terms of generating stronger passwords compared to the other password managers. It outperforms Last Pass in all scenarios and it is better than 1password and KeePass when the length value is in the spectrum of smaller values or bigger values, but slightly worse when in the average spectrum of values.

## 6.3 Application Security

To measure application security the following aspects will be tested:

- Communication Encryption
- Brute-force resistance
- XSS attacks resistance

### 6.3.1 Communication Encryption

For communication encryption, the main concern is if data is encrypted between communication and if the encryption protocol is strong. When data is sent between client and server, it can be observed with the Wireshark tool[44] that packages are sent with TLS 1.3 protocol.

By capturing HTTPS packets from the communication between the client and the server [Figure 17] it can be noticed that the TLS version seems to be 1.2, but it is not the correct field.



No.	Time	Source	Destination	Protocol	Length	Info
1306	229.693978	:::1	:::1	TLSv1.3	128	Change Cipher Spec, Application Data
1307	229.694000	:::1	:::1	TCP	64	8443 → 58334 [ACK] Seq=1292 Ack=582 Win=2618880 Len=0
1308	229.694124	:::1	:::1	TLSv1.3	856	Application Data
1309	229.694139	:::1	:::1	TCP	64	8443 → 58334 [ACK] Seq=1292 Ack=1374 Win=2618112 Len=0
1310	229.695886	:::1	:::1	TLSv1.3	1123	Application Data
1311	229.695908	:::1	:::1	TCP	64	58334 → 8443 [ACK] Seq=1374 Ack=2351 Win=2616576 Len=0
1312	230.084076	127.0.0.1	127.0.0.1	TCP	50	56612 → 1042 [PSH, ACK] Seq=133 Ack=45 Win=10215 Len=6
1313	230.084097	127.0.0.1	127.0.0.1	TCP	44	1042 → 56612 [ACK] Seq=45 Ack=139 Win=10179 Len=0
1314	230.084239	127.0.0.1	127.0.0.1	TCP	46	1042 → 56612 [PSH, ACK] Seq=45 Ack=139 Win=10179 Len=2
1315	230.084256	127.0.0.1	127.0.0.1	TCP	44	56612 → 1042 [ACK] Seq=139 Ack=47 Win=10215 Len=0
1316	230.307536	:::1	:::1	TLSv1.3	748	Application Data
1317	230.307559	:::1	:::1	TCP	64	58334 → 8443 [ACK] Seq=1374 Ack=3035 Win=2615808 Len=0
> Null/Loopback > Internet Protocol Version 6, Src: ::1, Dst: ::1 > Transmission Control Protocol, Src Port: 8443, Dst Port: 58334, Seq: 1292, Ack: 1374, Len: 1059 > Transport Layer Security						
> TLSv1.3 Record Layer: Application Data Protocol: Application Data Opaque Type: Application Data (23) Version: TLS 1.2 (0x0303) Length: 1054 Encrypted Application Data: d75e51f71efbb69d51fc9758f9031195378489153066b9b0785f75620333903bb5121583...						
0000	18 00 00 00 60 0d 85 4f	04 37 06 80 00 00 00 00	.....0 7.....			
0010	00 00 00 00 00 00 00 00	00 00 00 01 00 00 00 00	.....			
0020	00 00 00 00 00 00 00 00	00 00 00 01 20 fb e3 de	.....			
0030	a8 eb ea 13 07 aa 1b 58	50 18 27 f3 6c fc 00 00	.....X P..1...			
0040	17 03 03 04 1e d7 5e 51	f7 1e fb b6 9d 51 fc 97	.....^Q.....Q...			
0050	58 f9 03 11 95 37 84 89	15 30 66 b9 b0 78 5f 75	X.....7...0f...x_u			
0060	62 03 33 90 3b b5 12 15	83 52 41 8c 03 68 b8 5c	b 3 ;...RA..h\			
0070	6b c6 43 be eb 87 3b 31	94 f7 e1 98 8a 2b af 08	k C...;1 .....			
0080	00 25 13 1f 4b d7 6c fa	56 e8 88 9f e6 64 39 db	..%..K..l V....d9.			
0090	22 e3 1f aa 7b 04 d4 38	56 67 f9 35 23 b4 2c b2	"...{-8 Vg.5#.,.			

Figure 17: Wireshark packet

In the past, the version field was used for version negotiation, but nowadays this mechanism is outdated or is not implemented properly and the solution for this is to leave the version field as it is and add the "supported version" [Figure 18] field where the best version of TLS will be used[45].

- > Extension: supported\_versions (len=7)
  - Type: supported\_versions (43)
  - Length: 7
  - Supported Versions length: 6
  - Supported Version: Reserved (GREASE) (0x3a3a)
  - Supported Version: TLS 1.3 (0x0304)
  - Supported Version: TLS 1.2 (0x0303)

Figure 18: Supported version field

Security-wise, TLS 1.2 was vulnerable mainly due to the support it offered for old cryptography algorithms that eventually became easy to crack. TLS 1.3 solves this issue by removing support for the old algorithms along with other features such as perfect forward secrecy, which means creating a unique id for each session and efficiency, by reducing the handshake protocol from 6 to 3 requests[46].

Compared with the other password managers, TLS 1.3 seems to be the norm in terms of communication encryption because it is easy to implement and offers meaningful features.

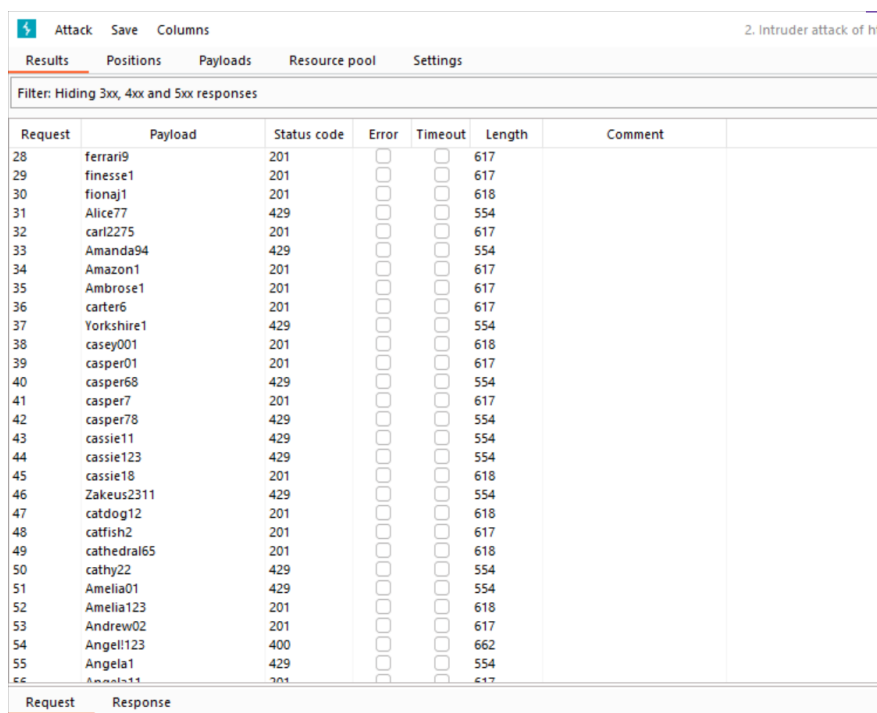
## 6.3.2 Brute-force resistance

App brute-force resistance will be measured using Burp Suite, a suite of tools used for penetration testing an application[47], along with a dictionary of the most used passwords[48]. It uses a local proxy that can capture multiple HTTP requests that pass through the local network.

For the evaluation scope, the request for password generation will be captured and sent to the intruder tool. The dictionary becomes a key element as it entries for the brute-force attack. For each entry, a new request will be created, where the master password will be changed with the respective entry.

After a 30-minute brute-force attack [Figure 19] on the password generation endpoint, the following has been observed:

- The brute-force attack manage to send 12 requests in a matter of seconds.
- Next, the request mechanism kicked in, and the intruder tool managed to send only 2 requests per minute.
- In 30 minutes, the intruder managed to successfully send only 70 requests.
- A good proportion of the requests have been denied and received the 429 error code from the server
- The password dictionary has 528.136 entries. To process all entries, it would require 3,74 months to run the attack, day and night.



The screenshot shows the 'Results' tab of the Burp Suite Intruder tool. The table displays the results of an attack on the password generation endpoint. The columns are: Request, Payload, Status code, Error, Timeout, Length, and Comment. The table shows 28 requests, with status codes ranging from 201 to 400. The 'Error' column contains checkboxes, and the 'Timeout' column contains checkboxes. The 'Length' column shows values ranging from 554 to 618. The 'Comment' column is empty.

Request	Payload	Status code	Error	Timeout	Length	Comment
28	ferrari9	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
29	finesse1	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
30	fionaj1	201	<input type="checkbox"/>	<input type="checkbox"/>	618	
31	Alice77	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
32	carl2275	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
33	Amanda94	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
34	Amazon1	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
35	Ambrose1	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
36	carter6	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
37	Yorkshire1	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
38	casey001	201	<input type="checkbox"/>	<input type="checkbox"/>	618	
39	casper01	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
40	casper68	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
41	casper7	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
42	casper78	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
43	cassie11	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
44	cassie123	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
45	cassie18	201	<input type="checkbox"/>	<input type="checkbox"/>	618	
46	Zakeus2311	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
47	catdog12	201	<input type="checkbox"/>	<input type="checkbox"/>	618	
48	catfish2	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
49	cathedral65	201	<input type="checkbox"/>	<input type="checkbox"/>	618	
50	cathy22	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
51	Amelia01	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
52	Amelia123	201	<input type="checkbox"/>	<input type="checkbox"/>	618	
53	Andrew02	201	<input type="checkbox"/>	<input type="checkbox"/>	617	
54	Angell123	400	<input type="checkbox"/>	<input type="checkbox"/>	662	
55	Angela1	429	<input type="checkbox"/>	<input type="checkbox"/>	554	
56	Angela11	201	<input type="checkbox"/>	<input type="checkbox"/>	617	

Figure 19: Brute-force attack results

### 6.3.3 XSS attacks resistance

Cross-Site Scripting or XSS attacks are injection-type attacks in which scripts with malicious code are injected through application inputs[49].

Due to the application working on a local environment, the attack surface for XSS vulnerabilities is limited because the app doesn't rely on web-based inputs or external content like the rest of the competition. However, XSS attacks should still be considered, because one vulnerability can lead to app instability.

The inputs in this app can be found as:

- The image upload input from the Welcome Page
- The QR scanner from the Welcome page
- The QR scanner from the Home page
- The 3 inputs from the Form page
- The master password input from the password generation dialog box

To test the application for XSS attacks, the guidelines present in the XSS page from the OWASP organization have been followed[49]. The input fields have been tested manually with various types of XSS attacks that should have shown an alert or retrieved a value and the application is secure.

## 6.4 Ease of Use

To make the app attractive for users, and to feel that the process has been simplified for them, the user experience must be reviewed. To test the user experience, an opinion survey has been conducted to gather data about what people think about the application. The following results have been achieved:

Function \ Intuitive Rating	1(%)	2(%)	3(%)	4(%)	5(%)
<b>Add startup picture</b>	1.15	2.35	4.7	19.8	72
<b>Add password configuration</b>	1.15	1.15	1.15	14	82.55
<b>Remove password configuration</b>	1.15	1.15	3.5	12.8	81.4
<b>Generate password</b>	1.15	1.15	2.3	14	81.4
<b>Export to QR</b>	1.15	1.15	2.3	11.6	83.7
<b>Import from QR</b>	1.15	1.15	2.3	11.6	83.7
<b>Results</b>	1.15	1.35	2.7	14	80.8

Table 7: Feedback statistics

The result of the survey [Table 7] suggests that 95% of the questioned users consider that the interface overall is simple enough to understand and the application process feels natural. However, 5% of the respondents didn't think that the application is simple to use and provided feedback on what should be improved:

1. When adding the startup image, the uploading button being red might lead the user to think that it is a canceling action.
2. When setting the length of the password configuration, the slider might be frustrating to use. A normal input text might be a better option.
3. When deleting a password configuration, the popup might make people think that they delete the password, not the configuration.
4. When generating the password, both the master password and the generated password should be hidden from view when written.
5. When generating the password, the input text areas for the master password and generated password should be a bit longer.

The last three items have been solved after the survey analysis finished. The other issues are subjective and could be solved by adding customization features. However, customization does not enter the project scope so these issues should be implemented in a future development.

## 7 CONCLUSIONS

In conclusion, this thesis proposes a password manager with a different approach from the other managers presented in the *State of the Art*, a personal local application that simplifies the process of creating secure passwords with the main focus on security and ease of use

The architecture of the proposed solution shows the design simplicity, a client-server application that just needs to be downloaded and started in Docker, a convenient solution for a simple user.

The implementation required different security algorithms and protocols according to their main strengths, mainly for encryption and communication. In addition, the main functionalities which have been inspired by the other password managers have been analyzed, implemented, and simplified for a better user experience.

On the other hand, future improvements must be implemented in the application as the actual state is far from reaching the scale of the current password managers. However, scaling up the application should not be a hard problem, as improvements can be easily identified and implemented.

### 7.1 Future development

The proposed solution, even if it was designed to run on a local environment, can be deployed on a website after developing certain improvements:

- Creating a new SSL certificate signed by a Certificate Authority and adding it to the project
- Tweaking algorithm parameters depending on the level of security required and the computational power available
- Tweaking CORS and STP protocol parameters to establish security restrictions for the communication protocol

Furthermore, releasing the application might require updating or implementing new encryption algorithms depending on the moment of consideration because as technology is constantly changing and improving, the actual algorithms might become vulnerable or outdated.

In addition, further testing the application for other vulnerabilities that might lead to sensitive data exposure is mandatory and it should be done before the release.

## BIBLIOGRAPHY

- [1] The OWASP Foundation Inc. Top 10 OWASP. <https://owasp.org/www-project-top-ten/>, 2021. Latest access: 03.05.2023.
- [2] Donald K. Davis, Md Minhaz Chowdhury, and Nafiz Rifat. Password security: What are we doing wrong? In *2022 IEEE International Conference on Electro Information Technology (eIT)*, pages 562–567, 2022.
- [3] Yuhua Yin, Julian Jang-Jaccard, and Nilufar Baghaei. Passimg: A secure password generation and management scheme without storing. In *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 341–346, 2022.
- [4] Fei Yu and Hao Yin. A security analysis of the authentication mechanism of password managers. In *2021 IEEE 21st International Conference on Communication Technology (ICCT)*, pages 865–869, 2021.
- [5] AgileBits. 1password Security Design. Technical report, 1password, 4711 Yonge St, 10th Floor, Toronto, Ontario, M2N 6K8, Canada, April 2023. Latest access: 03.05.2023.
- [6] Katy Wills. Technical and Organizational Measures for LastPass. Technical report, LastPass, July 2022. Latest access: 03.05.2023.
- [7] Dominik Reichl. Security - KeePass. <https://keepass.info/help/base/security.html>. Latest access: 03.05.2023.
- [8] denadz. Github - KeeFarce tool. <https://github.com/denandz/KeeFarce>. Latest access: 03.05.2023.
- [9] Google. Angular. <https://angular.io>. Latest access: 17.06.2023.
- [10] VMware Tanzu. Spring Boot. <https://spring.io/>. Latest access: 17.06.2023.
- [11] Apache Software Foundations. Maven. <https://maven.apache.org/>. Latest access: 17.06.2023.
- [12] Baeldung. Intro to Spring Boot Starters. <https://www.baeldung.com/spring-boot-starters>. Latest access: 21.06.2023.
- [13] VMware Tanzu. Securing a Web Application. <https://spring.io/guides/gs/securing-web/>. Latest access: 21.06.2023.

- [14] Roel Spilker and Reinier Zwitterloot. Project Lombok. <https://projectlombok.org/>. Latest access: 21.06.2023.
- [15] Map Struct - Java bean mapping the easy way! <https://mapstruct.org/>. Latest access: 21.06.2023.
- [16] JSON. <https://www.json.org/json-en.html>. Latest access: 21.06.2023.
- [17] Baeldung. Introduction to BouncyCastle with Java. <https://www.baeldung.com/java-bouncy-castle>. Latest access: 21.06.2023.
- [18] Github - zxing. <https://github.com/zxing/zxing>. Latest access: 21.06.2023.
- [19] Farid Zakaria. Github - ASCII85. <https://github.com/fzakaria/ascii85>. Latest access: 21.06.2023.
- [20] JUnit5. <https://junit.org/junit5/>. Latest access: 21.06.2023.
- [21] AssertJ - fluent assertions java library. <https://assertj.github.io/doc/>. Latest access: 21.06.2023.
- [22] PrimeNG. <https://primeng.org/>. Latest access: 21.06.2023.
- [23] Bootstrap. <https://getbootstrap.com/>. Latest access: 21.06.2023.
- [24] Stepan Suvorov. ngx-cookie-service. <https://github.com/steevermeister>. Latest access: 21.06.2023.
- [25] DaiDH. ngx-scanner-qrcode. <https://github.com/id1945/ngx-scanner-qrcode>. Latest access: 21.06.2023.
- [26] Burt Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. <https://www.rfc-editor.org/info/rfc2898>, September 2000. Latest access: 17.06.2023.
- [27] Colin Percival and Simon Josefsson. The scrypt Password-Based Key Derivation Function. <https://www.rfc-editor.org/info/rfc7914>, August 2016. Latest access: 17.06.2023.
- [28] Alex Biryukov, Daniel Dinu, Dmitry Khovratovich, and Simon Josefsson. Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications. <https://www.rfc-editor.org/info/rfc9106>, September 2021. Latest access: 17.06.2023.
- [29] DCODE. ASCII85 encoding. <https://www.dcode.fr/ascii-85-encoding>. Latest access: 20.06.2023.
- [30] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. <https://www.rfc-editor.org/info/rfc4648>, October 2006. Latest access: 18.06.2023.

- [31] Joseph A. Salowey, David McGrew, and Abhijit Choudhury. AES Galois Counter Mode (GCM) Cipher Suites for TLS. <https://www.rfc-editor.org/info/rfc5288>, August 2008. Latest access: 17.06.2023.
- [32] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. <https://www.rfc-editor.org/info/rfc8017>, November 2016. Latest access: 17.06.2023.
- [33] Hrithik Saini. 8 Strongest Data Encryption Algorithms in Cryptography. <https://www.analyticssteps.com/blogs/8-strongest-data-encryption-algorithms-cryptography>, 2022. Latest access: 17.06.2023.
- [34] IBM. What is REST API? <https://www.ibm.com/topics/rest-apis>. Latest access: 17.06.2023.
- [35] WHATWG community. Fetch Living Standard - CORS protocol. <https://fetch.spec.whatwg.org/#cors-protocol>, June 2023. Latest access: 18.06.2023.
- [36] VMware Tanzu. Cross Site Request Forgery (CSRF) - Synchronizer Token Pattern. <https://docs.spring.io/spring-security/reference/features/exploits/csrf.html#csrf-protection-stp>. Latest access: 18.06.2023.
- [37] Eric Rescorla. HTTP Over TLS. <https://www.rfc-editor.org/info/rfc2818>, May 2000. Latest access: 20.06.2023.
- [38] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. <https://www.rfc-editor.org/info/rfc5246>, August 2008. Latest access: 20.06.2023.
- [39] Patrik Fältström, Fredrik Ljunggren, and Dirk-Willem van Gulik. The Base45 Data Encoding. <https://www.rfc-editor.org/info/rfc9285>, August 2022. Latest access: 20.06.2023.
- [40] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. <https://www.rfc-editor.org/info/rfc8259>, December 2017. Latest access: 18.06.2023.
- [41] Postman. Postman. <https://www.postman.com/>. Latest access: 22.06.2023.
- [42] Comparitech. Password Strength Test and Strong Password Generator Tool. <https://www.comparitech.com/privacy-security-tools/password-strength-test/>. Latest access: 22.06.2023.
- [43] University of Illinois at Chicago - Academic Computing and Communications Center. Password strength test. <https://www.uic.edu/apps/strong-password/>. Latest access: 22.06.2023.



- [44] Wireshark Foundation. Wireshark. <https://www.wireshark.org/>. Latest access: 25.06.2023.
- [45] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. <https://www.rfc-editor.org/info/rfc8446>, August 2018. Latest access: 24.06.2023.
- [46] Krupa Patil. Why Is TLS 1.3 Better And Safer Than TLS 1.2? <https://www.appviewx.com/blogs/why-is-tls-1-3-better-and-safer-than-tls-1-2/>, March 2022. Latest access: 24.06.2023.
- [47] PortSwigger. Burp Suite. <https://portswigger.net/burp>. Latest access: 24.06.2023.
- [48] Duyet Le. Github - bruteforce-database. <https://github.com/duyet/bruteforce-database>. Latest access: 24.06.2023.
- [49] KirstenS. Cross Site Scripting. <https://owasp.org/www-community/attacks/xss/>. Latest access: 24.06.2023.