

Moaz Bassam Yanes

Development of a secure, role-based password manager

Favn Password Manager

Bachelor's thesis in Computer Engineering

Supervisor: Donn Morrison

May 2023

Moaaz Bassam Yanes

Development of a secure, role-based password manager

Favn Password Manager

Bachelor's thesis in Computer Engineering

Supervisor: Donn Morrison

May 2023

Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering

Department of Computer Science



Norwegian University of
Science and Technology

Sammendrag

Som et resultat av den raske teknologiske utviklingen og økningen av digitalisering globalt, er det en stor økning i datamengden. Denne dataen krever skikkelig beskyttelse mot nettoangrep. Mer sensitiv digital data krever mer robuste sikkerhetsløsninger for å beskytte denne informasjonen. I denne moderne æraen blir passord ansett som sensitiv data som må sikres fordi de fungerer som det første forsvarslinjen i beskyttelsen av informasjon gjennom autentiseringsprosesser.

Dette prosjektet har som mål å utvikle et system for å beskytte disse passordene. Oppgaven i dette prosjektet er å bygge et passordhåndteringssystem med spesifikke tilpassede krav.

Produkteieren, Favn Software, er et programvareselskap i Trondheim, som opererer med en hybrid forretningsmodell som fokuserer på innovasjon og konsultasjon i programvareutvikling. Det har vokst raskt de siste årene og trenger mer robust intern sikkerhet.

De ber om at Favn Password Manager skal tilby et sikkert og enkelt lagringssystem for passordene deres med et tilpasset innholdsstyringssystem (CMS). Styringssystemet organiserer passordene i mapper med hierarkisk tilgang basert på roller. Systemet bør operere med en null-kunnskapsserver, altså mangler serveren informasjon om sensitiv data.

Gjennom hele prosjektet har sikkerhetsaspekter og -betingelser blitt utforsket i praksis. Derfor hadde kryptografiske applikasjoner en vesentlig rolle i utviklingen av dette produktet, ved å bruke krypteringsmetoder for å sikre sensitiv data. Systemutviklingsmetodikkene ble brukt for å organisere arbeidet og levere produktet. Generelt var prosjektet både utfordrende og interessant og utviklet seg gradvis for å implementere funksjonene.

Selv om systemet har noen mangler som må forbedres, er hovedfunksjonene implementert, og systemet er i en tilstand hvor det er klart for bruk.

Abstract

As a result of the rapid technological development and the rise of digitalization globally, there is a major increase in data amount. This data requires proper protection against cyberattacks. More sensitive digital data requires more robust security solutions to safeguard this information. In this modern era, passwords are considered sensitive data that need to be secured because they serve as the first line of defense in protecting information through authentication processes.

This project aims to develop a system to protect these passwords. The task in this project is to build a password management system with specific custom requirements.

The product owner, Favn Software, is a software company in Trondheim, operating with a hybrid business model that focuses on innovation and consultation in software development. It has been growing fast in the last few years, needing more robust internal security.

They request the Favn Password Manager to provide a secure and simple storage system for their passwords with a custom Content Management System (CMS). The management system organizes the passwords in folders with hierarchical access based on roles. The system should operate with a zero-knowledge server, thus the server lacks information about sensitive data.

Throughout the project, security aspects and terms have been explored in real life practice. Therefore, cryptographic applications had an essential role in developing this product, using encryption methods to secure sensitive data. System development methodologies were used to organize the work and deliver the product. Overall, the project was both challenging and interesting and gradually developed to implement the features.

Even though the system has some shortcomings that need to be improved, the core features are implemented, and the system is in a state where it is ready for use.

Preface

This report is authored by Moaaz Bassam Yanes, a student at NTNU, as a part of a bachelor's thesis in the course IDATT2900 Bachelor Thesis in Computer Engineering.

The assignment was chosen by the student due to his interest in cybersecurity, cryptography, and a desire to solve a problem involving the development of a product that uses cryptography for the bachelor's thesis. It was requested by the company, Favn Software AS, to focus on today's online world, where security risks resulting from password breaches make a persistent threat. For this reason, the assignment was undertaken, to create a solution and develop a simple and secure password management system for the company.

The purpose of this document is to clarify the underlying theoretical aspects and technologies related to the problem being solved, detail the development process of the product, and discuss the research results.

Acknowledgements

The student thanks his NTNU supervisor, Dr. Donn Morrison, for his excellent guidance in navigating challenges and finding the right solutions and for the support provided throughout the project. Additionally, appreciation is extended to the company representatives, William Chakroun Jacobsen and Sveinung Øverland, for their guidance on the task and valuable feedback during the development process.

Special thanks are also due to the student's family, for their support and encouragement during the academic journey. Warm thanks and gratitude are offered to the student's uncle, Dr. Bahaa Yanes, for his unwavering support in the pursuit of science.

Moaaz B. Yanes

Moaaz Bassam Yanes

Task

The task for this project is to develop a password management system with a custom Content Management System (CMS). Throughout the project, there will be various challenges representing real-world scenarios in which the student must determine what is necessary and what is "secure enough".

The solution allows for saving passwords in a highly secure manner with a controlled access. Password access should be role-based, where different roles are assigned to different folders containing passwords. It should also be possible for users to access specific folders hierarchically. Additionally, the method of storing passwords must be investigated, as the system itself should not have knowledge of the passwords. A robust and secure login approach is also required in the system.

The complete list of functional and non-functional specifications can be found in the vision document located in the appendices [Appendix A]. The list of user stories can also be found in the requirements document [Appendix C]. The product owner, FavN Software, has exclusively set functional requirements, without specific requirements for technology or developer environment.

Contents

List of Figures	viii
List of Tables	viii
1 Introduction	1
1.1 Background	1
1.2 Research question	1
1.3 Document structure	2
2 Theory	3
2.1 Password management	3
2.1.1 Bad Password Habits	3
2.1.2 Password managers	3
2.2 Encryption	4
2.2.1 Symmetric vs. Asymmetric Encryption	5
2.2.2 Key derivation	7
2.2.3 Hash and salt	8
2.3 Web technologies	9
2.3.1 Rest	9
2.3.2 HTTP	10
2.3.3 HTTP Rest API	10
2.3.4 HTTPS	11
2.4 Storage	11
2.4.1 Relational database systems	11
2.4.2 IndexedDB	12
2.5 Authentication and Authorization	12
2.5.1 Definition	12
2.5.2 Two factor authentication	13
2.6 Development and work methodology – RAD	13
3 Method	15
3.1 Process	15
3.2 Research methodology	15
3.3 Development methodology	16

3.3.1	Agile system development with RAD	16
3.3.2	GitLab	17
3.3.3	System architecture.....	17
3.3.4	Choice of technologies	19
3.3.5	Security	21
3.3.6	Testing.....	22
4	Results.....	23
4.1	Scientific results.....	23
4.1.1	Features.....	23
4.1.2	Encryption techniques.....	24
4.1.3	Hierarchical access	26
4.1.4	Zero-knowledge server architecture	29
4.2	Engineering results	30
4.2.1	Backend system development	30
4.2.2	Frontend Implementation	31
4.2.3	Updating Passwords (Synchronization)	35
4.2.4	Authentication	37
4.2.5	Security	39
4.3	Administrative results	40
4.3.1	Timesheet and activity.....	40
4.3.2	Methodology	42
5	Discussion	43
5.1	Scientific results.....	43
5.1.1	Features.....	43
5.1.2	Encryption techniques.....	43
5.1.3	Hierarchical access	44
5.1.4	Secure Zero-knowledge server architecture	46
5.2	Engineering results	47
5.2.1	Functional requirements	47
5.2.2	Non-functional requirements.....	48
5.2.3	System architecture.....	49
5.2.4	Backend.....	51
5.2.5	Frontend	52

5.2.6	Security	53
5.2.7	Authentication	54
5.3	Administrative results	55
5.3.1	Reflection	55
6	Conclusion and Further Work	57
	Social Impact.....	59
	References	60
	Appendices	65

List of Figures

Figure 1: Data Encryption Process (Bitdefender, n.d.)	5
Figure 2: Symmetric encryption (SSL2BUY, 2019)	6
Figure 3: Asymmetric encryption (SSL2BUY, 2019).....	7
Figure 4: Illustration of appending salt to passwords with comparing of the hash output.	9
Figure 5: A Visualization of the difference between traditional development methodologies and RAD	14
Figure 6: Business diagram illustrating the four-step password creation process.	18
Figure 7: System architecture diagram	18
Figure 8: Overview of encryption usage for passwords in the system	26
Figure 9: An example of tree data structure implementation in the database	27
Figure 10: An example of the tree data structure implementation for organizing roles and access control in the system.....	29
Figure 11: Workflow diagram illustrating the zero-knowledge process on client side.....	30
Figure 12: Modular structure of the custom CMS handling requests and responses.	31
Figure 13: Main Page of admin dashboard with system tree and roles	32
Figure 14: Adding new password within a selected node, which uses the encryption process.	33
Figure 15: Password list in admin dashboard.	34
Figure 16: Illustration of the password synchronization process for new users in the frontend	36
Figure 17: Sequence diagram of the user authentication process without 2FA	38
Figure 18: Work distribution pie chart	41
Figure 19: Project's Gantt diagram	41
Figure 20: Database tables diagram	45
Figure 21: Discrete logarithm problem attack illustration.....	51
Figure 22: Node management pop-up menu	53

List of Tables

Table 1: Feature comparison between Favn Password Manager, LastPass, and 1Password ..	23
Table 2: Functional requirements implementation status	47
Table 3: Status of user's needs implementation.....	47

1 Introduction

1.1 Background

Favn Software, a software engineering company located in Trondheim, Norway, was established in 2020 by students from NTNU (Norwegian University of Science and Technology). Operating with a hybrid business model that focuses on innovation and consultation in software development, it has a specialization in frequent development and production of MVP. Favn's assorted options for services include the creation of websites, back-end systems, mobile apps, and more.

Favn Software is a company that consistently aspires to quality and maintains modernity in all aspects of its operations, including internal procedures and infrastructure as well as the products and services that it offers to clients. The company thus requested the creation of a password management system in order to meet the need for a more effective and secure approach to managing their internal operations.

At present, the company does not use any password management solutions. They instead rely on a shared text file containing all passwords, which is available to the entire team. As the company has rapidly grown in recent years, there has been little opportunity to consider security improvements, possibly due to budget constraints. The essential purpose of the desired product is to securely store passwords, with secure sharing among users, and define access based on the user's role hierarchically. This product can optimize the internal security and provide the company with complete control over their data.

1.2 Research question

The project has aimed to develop a secure and simple password manager with role-based access control for hierarchical management of passwords. To achieve the main requirements, the focus has been on implementing and using secure choices and approaches.

The research question for this project has become:

What are the key design principles and best practices for developing a secure password manager with hierarchical access to passwords based on roles?

1.3 Document structure

A brief overview of the structure of the document:

- **Introduction** – background information and thesis question.
- **Theory** – summary of relevant background theory to understand the method, results and discussion.
- **Method** – explanation of the development process, research methodology, development methodology and technologies used.
- **Results** – the results demonstrate the work accomplished throughout the project duration. It include scientific results showing the efforts made to respond to the research question. Engineering results evaluate the product itself. Administrative results illustrate the student's work process.
- **Discussion** – a section explores the relationship between the achieved results and the research question.
- **Conclusion** – a brief answer to the thesis question using the results and the discussion, with suggestions for further work:
- **References** – a detailed list of the sources cited in this paper.
- **Appendices** – list of attachments provided along with the thesis.

2 Theory

2.1 Password management

2.1.1 Bad Password Habits

In the age of digital connectivity, strong and unique passwords play an essential role in safeguarding online security and protecting sensitive data and managing access to these data.

However, an enormous number of users have poor password habits, which can lead to unauthorized access and data breaches.

Poor password habits are widespread, and they contribute to security breaches. According to a survey conducted by Digital Guardian, a data protection platform (Digital Guardian, 2018):

- Many individuals tend to create weak passwords or reuse the same password across multiple accounts, exposing themselves to security risks.
- Among the survey participants, a large number admitted to using the same or very similar passwords for multiple online accounts.
- A large portion of participants confessed to memorizing their passwords or writing them down in plaintext.
- Only a small portion said they use a password manager to securely store and manage their login credentials.

Generally, the results show that password bad habits are widespread among people, and this confirms the importance of using better password habits and encouraging the use of password managers to improve online security.

2.1.2 Password managers

Password managers are software applications (web, desktop, and mobile applications) or browser extensions designed to store and manage login credentials for various online accounts, allowing users to save passwords securely. They help in decreasing the risks associated with poor password habits [1.1] by allowing users to maintain multiple complex passwords without the need to memorize or write them down.

By taking advantage of top tier encryption algorithms, password managers ensure that stored credentials are safe from unauthorized access (nordpass.com, n.d.). Additional features can be found in these applications, such as automatic form fill capabilities, which

make logging in easier for users. Password generators that create difficult to guess passwords are also included.

Online security can get a boost with the use of a password manager, which also lowers the risk of cyberattacks. As users often prefer weak or reused passwords to avoid the issue of memorizing multiple strong credentials, password managers alleviate this issue.

2.2 Encryption

Encryption is a process utilized to prevent unauthorized access and protect data confidentiality (Cloudflare, n.d.). This converts plaintext data into an encoded format called a cipher via a mathematical algorithm. Without the decryption key, the information becomes illegible. The encryption key is comprised of mathematical values, which must be agreed upon by both the message's sender and receiver.

Definition some key terms of encryption to understand the process better:

- **Plaintext:** The original, unencrypted state of the information.
- **Ciphertext:** The encrypted information.
- **Encryption key:** A set of mathematical values exported as a random string of numbers, letters, and symbols used by a cryptographic algorithm to encrypt and decrypt data. It is like a physical key, which only allows those with the correct key to lock and unlock. The most secure keys are unique and unpredictable.
- **Encryption algorithm:** The method that uses the encryption key to convert plaintext into ciphertext (random text). Decrypting the data is the process of using the correct key to convert the ciphertext into plaintext and make it readable again.

How data encryption works

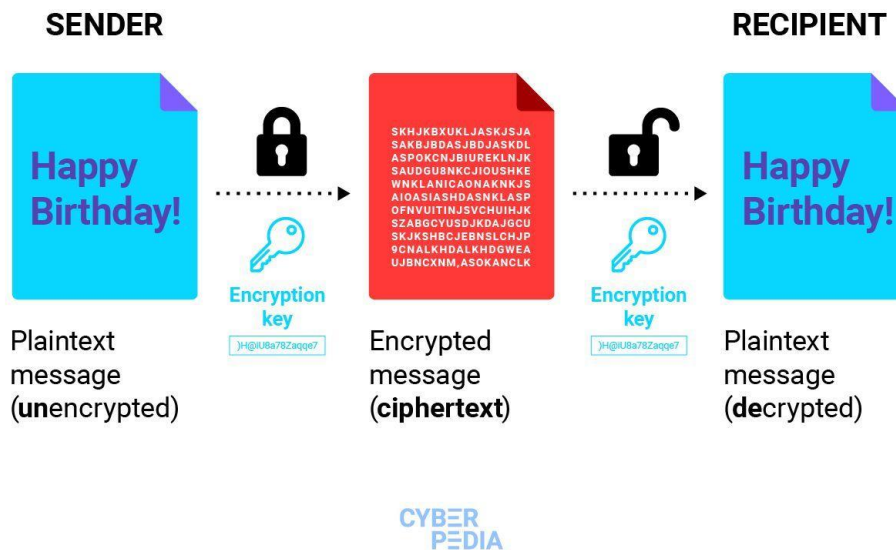


Figure 1: Data Encryption Process (Bitdefender, n.d.)

Encryption can provide protection for data during transit across the internet by applying encryption to transport protocols, such as HTTPS. Furthermore, security can be ensured by encrypting data stored in databases and servers.

Unauthorized access can be prevented, and security threats can be curbed with encryption in password management systems. The unapproved parties cannot get hold of the login credentials as the sensitive data gets encrypted by the password managers. It renders the data incomprehensible to anyone who doesn't have permission, even if there's a breach in the system (Micro Focus, n.d.).

There are two primary types of encryptions:

- **Symmetric encryption:** Uses a single key for both encryption and decryption processes.
- **Asymmetric encryption:** Uses a public-private key pair, allowing secure communication between parties without the need to share a common secret key (Cloudflare, n.d.).

2.2.1 Symmetric vs. Asymmetric Encryption

As mentioned in [2 Encryption](#), there exist two encryption techniques, symmetric encryption and asymmetric encryption, each with distinct advantages and limitations. Their differing use of encryption keys plays a role in securing data.

Symmetric encryption: Using a single key for encrypting and decrypting data is known as symmetric encryption or secret key encryption. This technique is faster than asymmetric encryption since the key has to operate less computationally. However, the key challenge of symmetric encryption is exchanging the key securely between parties communicating. Unauthorized access to the key risks the security of the encrypted data.

The most common symmetric encryption algorithms are Advanced Encryption Standard (AES), Data Encryption Standard (DES), and Triple DES (3DES). (Harmening, 2017).

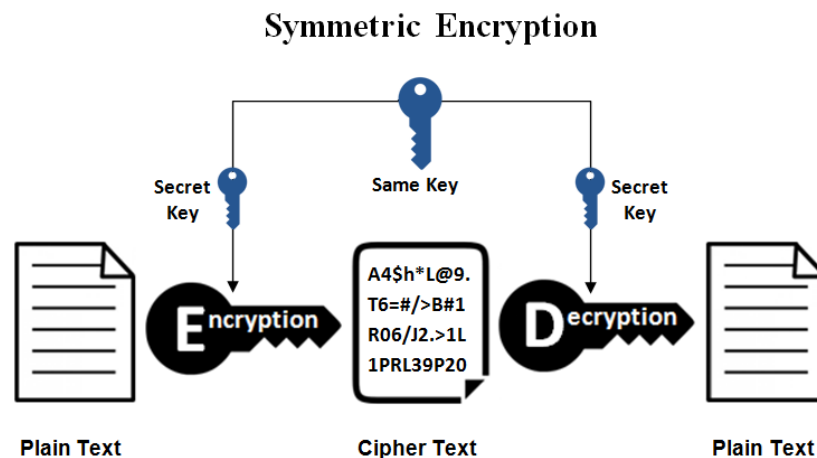


Figure 2: Symmetric encryption (SSL2BUY, 2019)

Asymmetric encryption: The encryption using public keys, commonly known as asymmetric encryption, involves a public key and a private key pair (Sciencedirect.com, 2017). The public key, provided on public for encryption, can be shared freely. The private key, utilized for decryption, is a closely guarded secret. With this technique, exchanging keys securely becomes unnecessary since only the public key is transmitted. Although more secure, asymmetric encryption is computationally more intensive and slower than symmetric encryption.

Encryption algorithms like Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) are frequently used for asymmetric encryption. (Sciencedirect.com, 2017).

Asymmetric Encryption

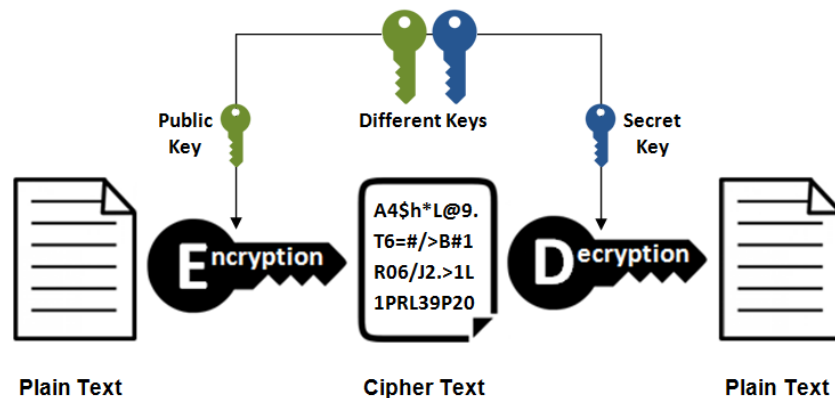


Figure 3: Asymmetric encryption (SSL2BUY, 2019)

2.2.2 Key derivation

From a master key, like a password or passphrase, key derivation is a cryptography technique that produces a secure key. To accomplish this, Key Derivation Functions (KDFs) utilize algorithms to develop a key derivation key. These KDFs were originally created to create robust, one-of-a-kind, and unpredictable keys for a range of cryptography tasks, from authentication to encryption and decryption (Sönmez and Paar, 2009). Passwords being easier to remember than binary keys is why cryptography benefits from key derivation.

Key derivation has many applications, but perhaps its most widespread use is in managing passwords. To protect sensitive data, like login credentials for multiple online accounts, a user's password is utilized to derive an encryption key. The key derived in this way can then be used to encrypt or decrypt the data as needed without compromising the user's login details. This is why key derivation is used by password managers to ensure that even if a user's password gets into the wrong hands, the related encryption key remains secure and unpredictable. This makes it extremely difficult for attackers to gain unauthorized access to the encrypted data.

Adding random data to the input value, otherwise known as salt, and using iteration are common techniques used in algorithms for performing the key derivation process. The functions responsible for performing this process are known as Key Derivation Functions (KDFs). In fact, KDFs often apply the algorithm multiple times to increase the complexity of the derived key according to Aggarwal (2021).

The Password-Based Key Derivation Function 2 (PBKDF2) is an often utilized KDF. It is great for creating strong keys from passwords.

2.2.3 Hash and salt

Hashing and salting are essential techniques in securely storing passwords.

Hashing is a *one way function* that takes an input and generates a *fixed size* string of characters. This output, known as a hash, is known as a “fingerprint” of the original data. Hash output *cannot be reversed* to obtain the original input. It is commonly used hash passwords for password storage, as it allows systems to verify passwords without storing the actual plaintext password. Some widely used hashing algorithms include MD5, SHA-1, SHA-2, SHA-3, and bcrypt (Arias, 2018).

Salting is a technique that adds an extra layer of security to the hashing process to safeguard passwords in storage. A salt is a random piece of data generated for each password, which is combined with the password before hashing. This combination makes it more difficult for attackers to use precomputed tables (e.g., rainbow tables) to crack hashes, as the salt increases the complexity and make passwords unique (Arias, 2018).

Adding salt to hash function input is important because it protects the hash (function output) against some types of attacks. Without using salt, two users with the same login password have identical hashes. But when adding a unique salt to each password, the resulting hash values are dissimilar, which complicates the process of cracking the passwords via techniques such as dictionary attacks and rainbow table attacks. (Arias, 2018)

In the following figure 4, an example is provided to illustrate how the salt is appended to the password. It demonstrates the output for two users with the same passwords in two cases: with and without salt. The hash function used in the example is SHA256.

Username	Password
User1	Qwerty123
User2	Qwerty123

Username	String to be hashed	Hashed value = SHA256
User1	Qwerty123	77aae185203edc6357676db95caa25d0f398d402c1723e6a7b42cfe8d2967f2e
User2	Qwerty123	77aae185203edc6357676db95caa25d0f398d402c1723e6a7b42cfe8d2967f2e

Without salt -> same output

Username	Salt	String to be hashed	Hashed value = SHA256
User1	D;%yL9TS:5PaIS/d	Qwerty123 D;%yL9TS:5PaIS/d	232e162dd2879f5fb4bf7611b17691a8fe9a4d0b6f63a3a33184be099d0cce50
User2)<,-<U(jLezy4j>*	Qwerty123 <,-<U(jLezy4j>*	7299c18b0bfb384d9080584841c7bc866f1e5188954369724b0f7f75fdd128d1

With salt -> different outputs

Figure 4: Illustration of appending salt to passwords with comparing of the hash output.

2.3 Web technologies

2.3.1 Rest

Representational State Transfer (REST) is a design architecture for distributed hypermedia systems. It is widely used for creating web services and APIs due to its simplicity and ease of use. The key characteristics of REST architecture are as follows (IBM, n.d.):

- **Separation of Client and Server:** By separating client and server, REST improves the portability of user interfaces, reduces complexity, and enhances the system's scalability.
- **Statelessness:** In REST architecture, each request must contain all the necessary information for the server to fulfill the request, making that the server doesn't need to store any client-specific context between requests.
- **Cacheability:** Responses to requests in REST can be labeled as cacheable or non-cacheable, allowing clients to cache data and reduce network load, which in turn leads to faster response times for the end-user.
- **Uniform Interface:** REST promotes a generalized architecture by using a consistent interface, simplifying communication between different components, and making it easier to understand and use.
- **Layered System:** REST architecture can be built using hierarchical layers, with each layer limited to interacting only with the resources it needs. This approach improves modularity and allows for better separation of concerns. For example, retrieving a user's information should only provide access to the user's properties, not the subjects the user is a part of.
- **Code on Demand (optional):** REST allows for extending client functionality by downloading and executing code, such as applets or scripts, on demand. This feature is optional and can be used to enhance the capabilities of clients when needed.

2.3.2 HTTP

The World Wide Web relies on Hypertext Transfer Protocol (HTTP), an application layer protocol devised by the Internet Engineering Task Force (IETF) to ferry data. This protocol is so fundamental to web communication that any URL using "HTTP" taps into it. HTTP, similar to REST, is stateless, meaning that every request is viewed as a singular instance without retaining any information from previous requests (R. Fielding, M. Nottingham and J. Reschke, 2022).

HTTP provides various methods or "**verbs**" for interacting with resources on the web:

- **GET**: Retrieves a resource without modifying it. GET requests should only be used for fetching data.
- **HEAD**: Returns the same response as a GET request but without the body. This is useful for obtaining metadata about a resource without actually downloading it.
- **POST**: Creates a new resource on the server.
- **PATCH**: Updates a single property of a resource.
- **PUT**: Updates one or more properties of a resource.
- **DELETE**: Removes a resource from the server.
- **CONNECT**: Establishes a network connection or tunnel to the server identified by the target resource. This is automatically handled by modern web browsers and typically doesn't require developer intervention.
- **OPTIONS**: Inquires about the allowed methods on the target resource.
- **TRACE**: Performs a message loop-back test, which is primarily used for debugging purposes.

Through the use of HTTP and its verbs, engineers and developers can implement multiple actions on the same endpoint. They can use CRUD operations (create, read, update, delete) to handle resources on the web, thus making simple and stable communication between clients and servers.

2.3.3 HTTP Rest API

Integrating HTTP with REST to develop an API enables the usage of HTTP verbs and the implementation of resources in a hierarchical structure, where each property within the resource must use unique identifiers. This integration permits the execution of multiple actions on the same endpoint, which better the versatility and functionality of the API.

2.3.4 HTTPS

The SSL/TLS protocols encrypt and authenticate the secure version of HTTP, giving what is known as HTTPS. It is a protocol defined in RFC 2818 of (May 2000) and uses the default port 443, as opposed to HTTP's port 80. In other words, HTTPS provides a more secure way to transmit data via the internet.

Adding security, encryption, integrity, and authentication to the HTTP protocol is the primary distinguishing factor between HTTPS and HTTP. HTTPS provides a safe way for users to transmit delicate data on the internet, including banking data, login credentials, and credit card information. HTTPS is quite important for securing online activities like shopping, banking, and remote work. HTTPS is rapidly becoming the standard protocol for all websites, irrespective of whether they exchange sensitive data with users or not.

HTTPS combines the [HTTP](#) protocol with [encryption](#), ensuring secure communication and data transfer between clients and servers.

2.4 Storage

2.4.1 Relational database systems

Relational database systems are a prevalent data storage method that organizes data in tables and establishes relationships between them. Each entry in a relational database has a unique primary key, allowing it to be distinguished from other entries (Oracle, 2021).

The act of separating related data into different tables and reducing data redundancy is called *normalization*. Normalization which reduces data redundancy and improves efficiency. By storing the primary key of a foreign table, known as a foreign key, connections between tables can be established. The implementation of normalization minimizes duplicate data storage and simplify data updates, as updating a single row implicitly updates all connected rows (Microsoft, 2022).

Consideration of relationship numbers and types is crucial when designing a successful relational database. Three key relationship types consist of one to one, many to one, and many to many. Special considerations are not necessary for a one to one relationship and the data in such a case may even be stored within the same table. In table A, one input can link to numerous entries in table B, but one entry in table B has only one correspondence with table A. To have multiple connections between the two tables, a connection table must be generated. It links various entries from table A and table B.

Most relational databases allow the use of *Structured Query Language (SQL)*, a domain-specific language specifically designed for managing these databases. Oracle (2022). SQL serves as a standard interface for interacting with databases. It provides the queries of creation, retrieval, updating, and deletion of data and more.

Relational database systems usually work alongside *Database Management Systems (DBMS)*, which provides a simple and perfect database management including data exploring, tables creations, tables relations and data queries.

Some examples of popular database software or DBMSs include Oracle, MySQL, Microsoft SQL Server, PostgreSQL (Statista, n.d.)

2.4.2 IndexedDB

IndexedDB is a web based database management system designed to handle the increasing complexity of modern browsers, which now support various multimedia applications, such as chatting, gaming, and internet banking. These applications are compatible with most browsers and can operate offline while retaining persistent data.

IndexedDB serves as a powerful solution for storing application data on the client's device, as it can store complex data types, such as objects, rather than just strings, which is the case for LocalStorage (developer.mozilla.org, n.d.). The ability to store more sophisticated data types enhances the functionality and efficiency of web applications, giving solutions for data storing problems in many use cases.

Key reasons for selection of IndexedDB include its support for object data types, support large data sets, indexing capabilities, and the ability to perform transactions. These features make it a suitable choice for applications that require robust data storage and retrieval on the client side, which ensure data persistence and better performance(developer.mozilla.org, n.d.).

2.5 Authentication and Authorization

2.5.1 Definition

Authentication and authorization are fundamental security principles which are often misunderstood as being synonyms. They both establish access control in software development and provide a mechanism to validate that users are indeed *who* they claim to be and have the *right access* to data and functionalities.

Authentication: It verifies user *identity*. This process required identifying who is accessing an application and validating their credentials, such as their username, email, or phone number, along with a password. This phase requires matching the visitor's used identity with the existing set of user identities in the system. (Auth0, n.d.)

Authorization: It involves granting specific *permissions* and *privileges* to a user. It works together with authentication, first verifying (authenticating) the user's identity, and then assigning permissions (authorizing) based on that confirmed identity. Within the system,

users are granted various permissions known as authorization that enable them to access certain resources like screens, data, or functionalities. Authorization process defined as verifying if a user has the required authorizations to carry out a task. (Auth0, n.d.)

2.5.2 Two factor authentication

Two-factor authentication (2FA) is a powerful tool used against online identity theft. It is an extra verification step to increase the security level by making that a compromised password alone doesn't result in an account breach. This added layer of security allows users to opt for their preferred method of secondary authentication.

2FA mechanism ensures that online users are indeed who they claim to be. After the standard authentication inputs (e.g., username and password), users must provide an additional piece of information before gaining access to the account. This second piece of information derives from one of these categories (Globalknowledge.com, 2018):

- Something you know: This might involve something a user knows, for instance, a personal identification number (PIN), a password, responses to certain security queries, or a unique pattern of keystrokes.
- Something you have: Typically, a user have a physical item used for verification. This could be a credit card, a smartphone, or a hardware token.
- Something you are: This advanced category includes biometric identifiers unique to the individual, like fingerprints, iris patterns, or voice characteristics.

2FA various types (VentureBeat, 2017):

- Hardware Tokens: This involves a physical device that generates a login code.
- SMS Text-Message and Voice-based: Users receive a code via SMS or voice call.
- Software Tokens: An application generates a code for the user, known as Time-based One-Time Password (TOTP)
- Push Notification: Users receive an alert on a trusted device and approve the login attempt.

2.6 Development and work methodology – RAD

Rapid Application Development (RAD) methodology is an approach to software development that provide the process with flexibility, adaptability, and efficient delivery. This methodology emerged as a response to traditional development models, for instance, waterfall model. This model can be inflexible, and it make modifying functions and features once the software was built is a challenge.

The concept of RAD has been in the industry for decades but has gained traction in recent years due to the rapid expansion of software development requirements. RAD was introduced by James Martin, a British computer scientist, in 1991 as a more flexible and efficient alternative to traditional software development methodologies (What is Rapid Application Development? i, n.d.).

Rapid Application Development methodology is designed to be more flexible and responsive to modifying and adding new features and functions at every step of the development process. The five main phases of RAD methodology are:

- Business modeling: Identifying and understanding the business processes and requirements.
- Data modeling: Defining the data structures and relationships within the system.
- Process modeling: Describing the processes and workflows that govern how data is managed and transformed.
- Application generation: Developing the actual software application using iterative techniques and rapid prototyping.
- Testing and turnover: Running thorough testing to check and ensure that the application meets the requirements and is ready for deployment. (What is Rapid Application Development? i, n.d.).

Benefits of RAD methodology in software development projects are numerous, including faster development cycles, improved collaboration between stakeholders, greater adaptability to change, and increased customer satisfaction due to better alignment with their needs.

In figure 5 below, the difference between traditional development methodologies and RAD is illustrated, showing the cyclical nature of the RAD phases.

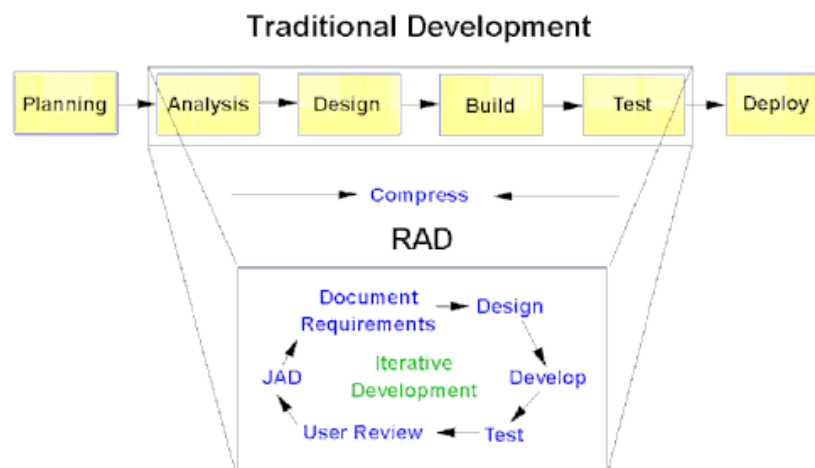


Figure 5: A Visualization of the difference between traditional development methodologies and RAD

3 Method

The method chapter is divided into process, research methodology, and development methodology. The research methodology describes the approaches used to gather existing knowledge or create new understanding, the development methodology relates to the choice of technology and how the product was built in practice. These methodologies were applied in the thesis to achieve Favn Software's desired product and to implement the requirements described in the vision document.

3.1 Process

In this project, the goal was to develop a secure, role based password management system for Favn. This system will be consumed by Favn to maintain full control over their passwords, rather than relying on a Software as a Service (SaaS) solution. The product will allow Favn to securely manage their passwords and grant access hierarchically.

This thesis was completed by one student without a team, which made popular team based agile methodologies unsuitable, such as Scrum. Consequently, the student decided to follow the Rapid Application Development (RAD) methodology. Throughout the project, the focus was on set up and identifying suitable solutions to implement the functional requirements of the system.

From the project's start, the main functionalities were defined clearly. As the first step in the project, a preliminary project plan document was created and provided the student with an understanding of the activities which the project consists of. A Gantt chart was developed for long-term activity planning and estimations.

A project vision document was created at the beginning of the system development process, offering a clearer understanding of the functional requirements specified by the product owner. The requirements and priorities in the document were determined and explained by Favn.

During development, multiple meetings were held with Favn to discuss potential solutions suitable for product development. Additionally, continuous feedback was received throughout the project via meetings and email correspondence with both Favn and NTNU supervisor. Favn was cooperative and understanding and realizing that the student did not have a team. As a result, the sprints and updates did not follow strict fixed periods but were instead determined through mutual agreement between Favn and the student.

3.2 Research methodology

A study approach focused on engineering and practice was taken to understand the concepts and technologies involved in password management and encryption. Key elements of the research methodology include:

- ❖ Researching the nature of password managers, their features, and their role in enhancing user security.
- ❖ Studying fundamental encryption concepts such as symmetric and asymmetric encryption, hashing, salting, and key exchange.
- ❖ Studying various authentication methods and techniques, such as two-factor authentication (2FA).
- ❖ Exploring secure password storage, encryption standards, and best practices for protecting user data.
- ❖ Understanding the concept of "tokens" and their use in authentication and authorization processes.
- ❖ Reviewing existing password management solutions and identifying their strengths and weaknesses.
- ❖ Exploring the applicability of tree data structures in relational databases and examining methods for their implementation and usage in managing hierarchical access control.

In addition, informal case studies of popular password managers, such as 1Password, LastPass, and the mail service Tutanota. They have been explored to gather inspiration and assist the development of the password manager's architecture. Based on the knowledge acquired through this research methodology, the requirements specification and architecture for the custom Favn Password Manager have been developed, implemented, and tested, with applying best practices for security.

3.3 Development methodology

3.3.1 Agile system development with RAD

As mentioned in the [process](#) [3.1], the Rapid Application Development (RAD) methodology was used to develop the product. At the beginning of the project, it was planned that design and development would take place in open iterations, with open sprints (non-fixed size sprints). The company was informed about every important update. There was also a focus on working continuously and having regular meetings with the client and supervisor throughout the project. The main goal was to finish developing and programming the system as soon as possible since the student was working alone and needed backup time for unexpected situations. This extra time could also be used to improve documentation. The workflow during the procedure was organized and managed using a physical task board, a Kanban board, with sticky notes.

3.3.2 GitLab

GitLab was used as a DevOps tool for collaboration and version control of source code. Although working alone, this tool was used for organizing and managing the workflow as desired. Since there were no conflicts to resolve with other team members, branching was not necessary. The code was pushed to the repository only after local testing. As stated in the vision document's section on non-functional criteria, continuous integration and deployment (CI/CD) were not necessary for Favn. However, they mentioned that ensuring the quality of the system was the main concern throughout the development process.

3.3.3 System architecture

Designing the system architecture was an important aspect of the project to implement the desired functionalities, particularly achieving the main two properties which are **security** and ensuring the **server has no knowledge** of the passwords. The design was refined multiple times before reaching the final result. Inspiration was drawn from other complex password managers and secure services providers such as Tutanota (a secure email service), but the design was modified to reduce complexity and make it more applicable.

The architecture relied on implementing encryption on the client side by the administrator in order to achieve the key features of the system—security and server not knowing the passwords. There are four steps in this process:

1. The admin creates the password.
2. The password is encrypted using symmetric encryption, which uses the same key for encryption and decryption.
3. The symmetric key is encrypted with asymmetric encryption using the public keys of users who have access to the password.
4. Two requests are sent to the server: one for the encrypted password and another for the encrypted copies of the key.

During decryption, the user needs the private key to decrypt the keys of the passwords and uses these keys to decrypt the corresponding passwords. To enhance security, the private key is stored in the browser's indexedDB, encrypted with user's master password using [key derivation](#). [2.2.2]

In the requirements document [Appendix C], sequence diagrams for the two main operations, encryption and decryption of passwords, are provided. In addition, domain model diagram is provided in the same document giving an overall understanding of the entity system.

The following Figure 6 presents the business diagram illustrating the four steps of password creation. Figure 7 showcases the system architecture diagram.

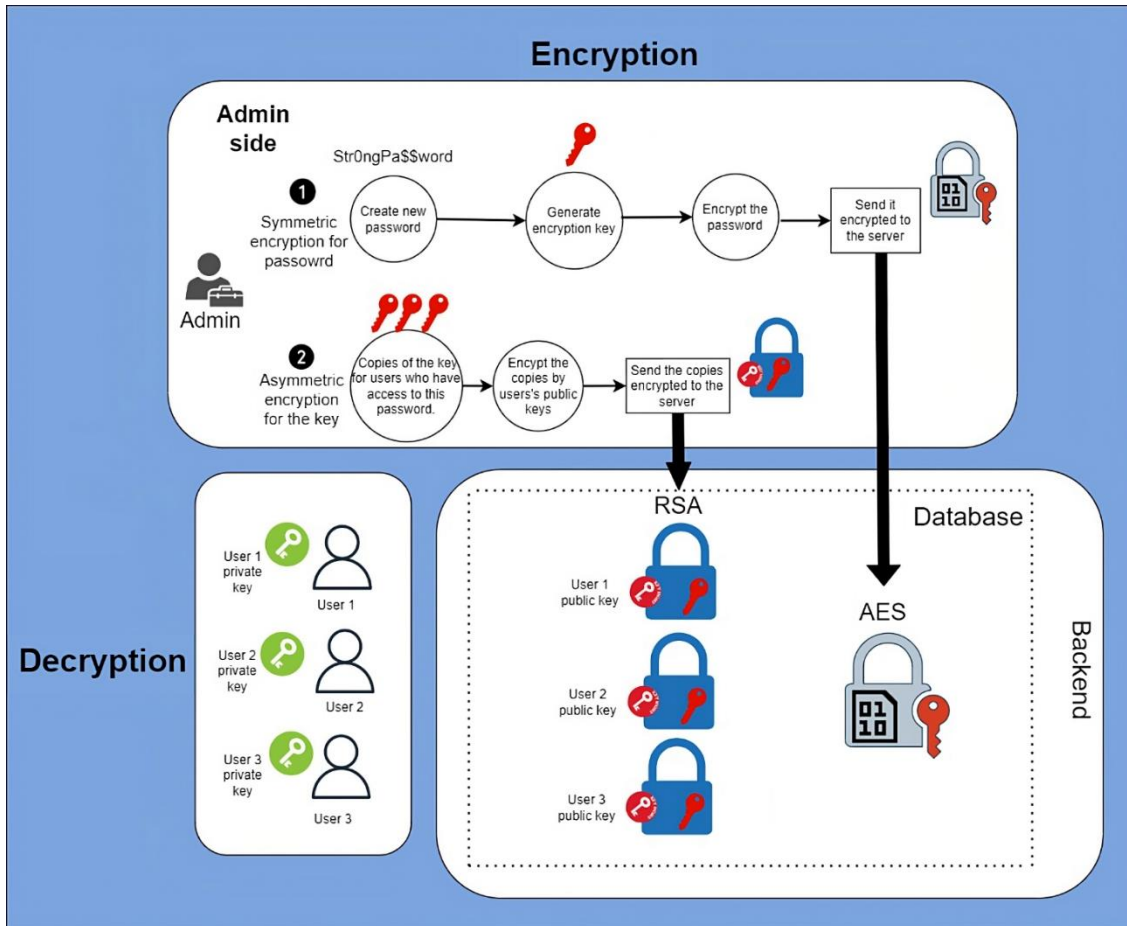


Figure 6: Business diagram illustrating the four-step password creation process.

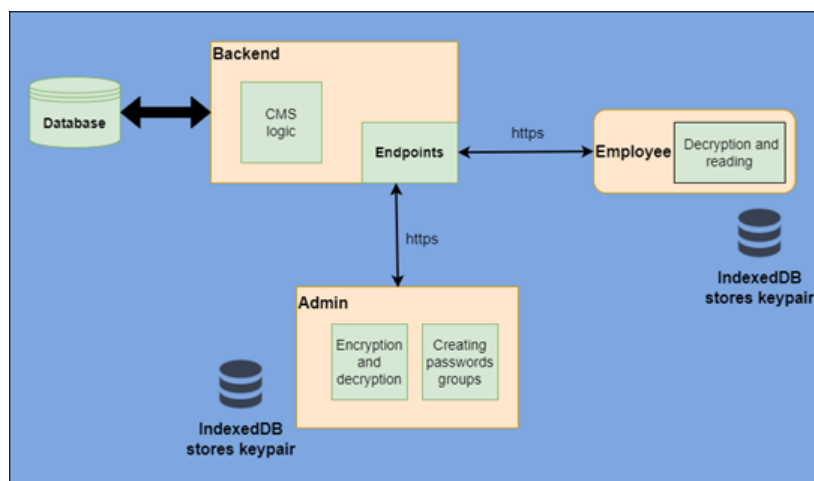


Figure 7: System architecture diagram

3.3.4 Choice of technologies

3.3.4.1 *Frontend: Vue*

In the project vision document, the company FavN made it clear that the interface is not essential and is considered secondary compared to the core of the system and its functionality. This reasoning led to the frontend choice being open without limitations. Therefore, the frontend solution, including the logic and encryption, could be implemented using vanilla JavaScript (pure JavaScript) without the need for special packages or frameworks.

The client side application is lightweight and does not require many resources or optimal refresh performance. The core of the client side focuses on handling passwords and the encryption and decryption processes. Security is an essential aspect to be considered on the client side, which can be achieved through good practices in encryption, authentication, and other security concepts.

The web application features two dashboards for users, the admin dashboard and the user dashboard. The admin dashboard displays the system tree, where nodes represent password "folders" and their contents of corresponding password lists. Additionally, it includes necessary option buttons for creating and editing items such as users, roles, nodes, passwords, and more within the system.

The user dashboard includes only the passwords the user has access to. The system also provides the main pages needed for registration and login processes.

Although the flexibility in frontend choices, the Vue3 framework was used in the project to build the product. Framework is used to create a familiar structure and makes the project easier to hand over after its completion. Vue3 is a popular and modern framework with good documentation. It was chosen to simplify the development process for the product owner or any bachelor students who may want to further develop the solution in the future. To take advantage of its capabilities as well, particularly those related to security, including its default prevention of Cross-Site Scripting (XSS). n.d. (Vue.js).

3.3.4.2 *Backend: Flask*

As mentioned in the frontend section[3.3.4.1], the core functionality involved encrypting passwords by the admin on the client side, which meant that the backend was not

responsible for password encryption. This design choice aligns with the system requirement that the server must not know the contents of the passwords. The server acted as a custom CMS (Content management system) to implement the requirements and manage desired operations. The majority of the logic, including CMS logic, authentication, authorization, and item management, was implemented from scratch without relying on third-party services or packages.

Many of the SQL queries were complex due to the hierarchical access feature for passwords, which required storing a tree structure in the database. This complexity, especially when selecting passwords and public keys using recursive queries, made these queries had to be written manually and ORM (Object Relational Mapping) solutions unnecessary for this project.

Flask was chosen as the backend framework because it is lightweight and straightforward for building REST APIs. Its minimalistic nature allows for greater control over the implementation, which is useful when developing such a secure password manager. Flask provides a solid foundation for creating secure applications when combined with best practices for security and proper configuration.

3.3.4.3 Persistence

A relational database (SQL Database) [2.4.1] was used to persist the data, including tables for managing authentication and authorization operations. The chosen DBMS [2.4.1] was PostgreSQL, according to the company's preference and as one of the non-functional requirements specified in the vision document. The design of the Entity-Relationship (ER) diagram was continuously updated and modified throughout the project to align with the evolving solution and features.

In addition to backend persistence, the system also uses frontend persistence by utilizing IndexedDB [2.4.2] for storing asymmetric keypairs, as described in the System Architecture. On decryption, users require the private key to unlock the encrypted keys associated with the passwords, which are then used to decrypt the passwords. The keypair is kept in the browser's IndexedDB and encrypted using the user's master password utilizing key derivation in order to increase security. With this client-side storage strategy, sensitive data is handled securely while yet allowing for quick access.

3.3.4.4 Docker

Docker was used in this project to run the PostgreSQL database and the Flask backend in separate containers. This decision was driven by the company's preference and as one of the non-functional requirements specified in the vision document. Docker Compose was used to manage and configure the communication between these containers, simplifying the

development process and ensuring a consistent environment across different stages of development.

Using Docker provided several benefits, such as simplifying deployment, ensuring consistency between environments, and isolating application components. This approach made it easier to manage dependencies and eased a smoother development experience. Additionally, Docker containerization capabilities will help the company Favn simplify the deployment process, making it easier to deploy and maintain the application across different environments and platforms. This can lead to reduced deployment issues and improved overall system reliability.

3.3.5 Security

As a password management system, security is a primary concern and must be completely considered throughout all aspects of the system. The research methodology's [3.2] best practices have been applied as much as possible in real life scenarios. The system contains a number of security aspects, such as:

- Encryption of all sensitive data, including passwords and keys as described in the system architecture, and other data elements that play a role in security, for instance, two-factor authentication secrets.
- Careful selection of encryption algorithms, with AES for symmetric encryption and RSA for asymmetric encryption, following recommended best practices.
- Hashing of login passwords on both the client side and server side to protect against interception and unauthorized access.
- Implementation of a cookie based authentication system that includes two-factor authentication for enhanced security.
- The adoption of recommended procedures for token expiration times. Refresh tokens are set to expire after 7 days, whereas access tokens and password setup tokens are set to expire after 15 minutes. These times limit risks and create a balance between use and security.
- Using the refresh token rotation approach to add an additional layer of security and prevent compromised tokens from being used again.
- Validation of user input during login and registration processes on the client side to prevent any potential security vulnerabilities due to malicious input.
- The implementation of a role-based authorization system, which makes sure that users can only access passwords in accordance with the roles and permissions. This feature supports the maintenance of control over who may see, edit, or remove particular data, further increasing the system's security and privacy. Additional

authorization procedures applied on both server and client side to protect pages and api requests that need admin permission.

- As part of following the OWASP Top 10 list guidelines, focus on preventing SQL injection by utilizing prepared statements with placeholders of the type %s when reading values in the backend.

The system's integration of these various security mechanisms aims to deliver a stable and safe environment for managing sensitive password data.

3.3.6 Testing

In the backend testing, the focus was on implementing unit tests while continuous integration and continuous deployment were not prioritized based on Favn's recommendation to concentrate on the system itself. In accordance with the non-functional requirement specified in the vision document, more than 80% of the methods in the manager.py file were covered by unit tests. This file, which serves as the core of the management system, contains all the essential methods for management, covered with a total of 28 tests.

Some SQL queries were complex, as mentioned in [3.3.4.2 backend]. To verify the correctness of these queries, their output was tested using an SQL script file in DBeaver, a popular database management tool, with dummy data. Then, manual tests for the methods in manager.py that incorporated these queries to ensure their proper functioning in the context of the overall system.

On the frontend, Favn did not prioritize the implementation of unit tests as a non-functional requirement in the vision document. Instead, it was used console logs and DOM manipulation for testing, alongside a simple page for displaying the encryption and decryption operations output. This temporary page was removed at the end of the project.

4 Results

The results demonstrate the work accomplished throughout the project duration. The scientific results present the efforts made to respond to the research question, while the engineering results evaluate productivity in light of the goals set at the beginning of the project. Administrative results illustrate the student's work process through timesheets, meetings, and similar activities.

4.1 Scientific results

This section presents the work made to answer the research question, hence the key findings from the research on the design principles, encryption techniques, tree data structures, and zero-knowledge server architecture for the secure password manager.

4.1.1 Features

Comparison of researched password manager features and implemented features in Favn Password Manager, with reference to the most common password managers, LastPass (lastpass.com n.d. 1&2&3) and 1Password (1Password n.d. 1&2).

For the tables below, it used the following abbreviations.

Encryption technique: Technique used to encrypt the passwords, not for authentication process.

Zero knowledge: Means that no one has access to the user's master password, or the data stored in the database, except the user. Not even the server.

End-point encryption: Encryption happens exclusively at the device level before syncing to the server for safe storage, so only users can decrypt their data.

Table 1: Feature comparison between Favn Password Manager, LastPass, and 1Password

Feature	Favn Password Manager	LastPass	1Password
Passwords encryption technique	Symmetric & Asymmetric	Symmetric	Symmetric
Zero knowledge	Yes	Yes	Yes
End-point encryption	Yes	Yes	Yes

Hashing	Yes	Yes	Yes
Key Derivation with master password	Yes	Yes	Yes
Hierarchical Role-based Access	Yes	No	No
Custom Role Creation	Yes	Limited	Limited
Encrypted Private Key Storage	Client-side	No	Server-side
Two-factor Authentication (2FA)	Yes	Yes	Yes
Authentication approach	Client-side & server-side hashing for master password	Unknown	Secure Remote Password (SRP)

4.1.2 Encryption techniques

4.1.2.1 *Evaluation of encryption techniques*

Evaluation of encryption techniques, including symmetric and asymmetric encryption, and their role in the system's overall security.

In the system both techniques symmetric and asymmetric used in the architecture described in [\[3.3.3\]](#).

Symmetric encryption is applied to the password that has been created, and asymmetric encryption is used on the key. First the password being encrypted with symmetric encryption AES (256). Then, the encryption key that is being used is encrypted using RSA asymmetric encryption. Following this, they are sent to the server in separate requests.

In Favn Password Manager system, a combination of symmetric and asymmetric encryption techniques has been applied to enhance the security and protect the sensitive data. The architecture incorporates these techniques to ensure the confidentiality of the passwords and to achieve the controlled access.

Confidentiality of the passwords: Symmetric encryption, specifically AES-256, is used to encrypt the password before it is stored or transmitted.

Controlled access: Asymmetric encryption is applied to the encryption key used in the symmetric encryption process.

Authentication: Symmetric encryption (Fernet) is used to encrypt and decrypt two factor authentication secrets, more details in engineering results[4.2.4].

The usage of both symmetric and asymmetric encryption techniques in the system provides a good security mechanism that safeguards the passwords and their associated keys. In addition, by sending the encrypted password and the encrypted symmetric key in separate requests to the server, the system reduces the risk of unauthorized access to the data.

4.1.2.2 *RSA, AES and Fernet*

AES and RSA

AES and RSA are used in Favn Password Manager because they are recommended and use encryption algorithms, known for their security, efficiency, and strong standing in the field of cryptography.

AES: A symmetric encryption algorithm with key lengths of 128, 192, or 256 bits. Established as a standard by NIST in 2001, AES is widely adopted in various industries for securing sensitive data. Its efficiency makes it suitable for a range of applications, from low-power devices to high-performance servers (Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES), 2001)

RSA: An asymmetric encryption algorithm introduced in 1977 that relies on the mathematical properties of large prime numbers for its security. RSA provides both encryption and digital signature capabilities, making it suitable for secure communication and authentication. Its security is based on the computational difficulty of factoring large composite integers (web.archive.org, 2021).

Fernet: A symmetric encryption method uses AES as its underlying encryption algorithm. It can be viewed as a more advanced encryption system that offers extra capabilities to increase encryption's security and usability. It uses AES in CBC mode with a 128-bit key and HMAC with SHA-256 for authentication. Developed for Python's cryptography library, it simplifies encryption and offers many benefits containing ease of use, built in integrity checks, secure key management, and timestamp-based token revocation.

These encryption algorithms have been analyzed and proven to be effective against cryptographic attacks, making them ideal choices for secure systems, such as password managers. (Kjellemo, 2022)

Figure 8 provides an overview of the encryption usage for passwords in Favn Password Manager.

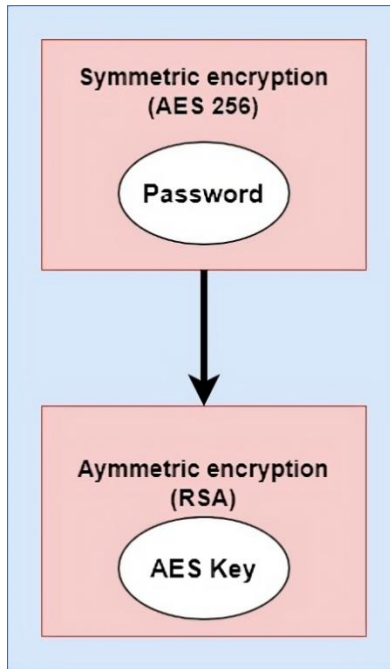


Figure 8: Overview of encryption usage for passwords in the system

4.1.3 Hierarchical access

Tree data structures are used in the system for its advantage for representing hierarchical relationships. In the context of storing a tree structure in a relational database for the password manager, the desired feature is to create a hierarchy based on user roles, where a user with a specific role can access the content of a node and its children nodes.

4.1.3.1 *Tree data structure implementation in the database*

Implementing tree data structure for role-based access control:

Tree data structure has been implemented in the relational database by creating a table named 'file' represents the tree's nodes (as seen in the domain model in requirements document C). This table refers to itself and has a 'parent_id' attribute to structure the tree. The root node has a parent_id with value 0, and when creating a new node, the parent_id is needed to build it under the existing node. Each node has a list of roles. The table could be named 'node', but due to modifications during the project and time constraints, it was not renamed.

The following figure provides an example of how the tree data structure is implemented in the database.

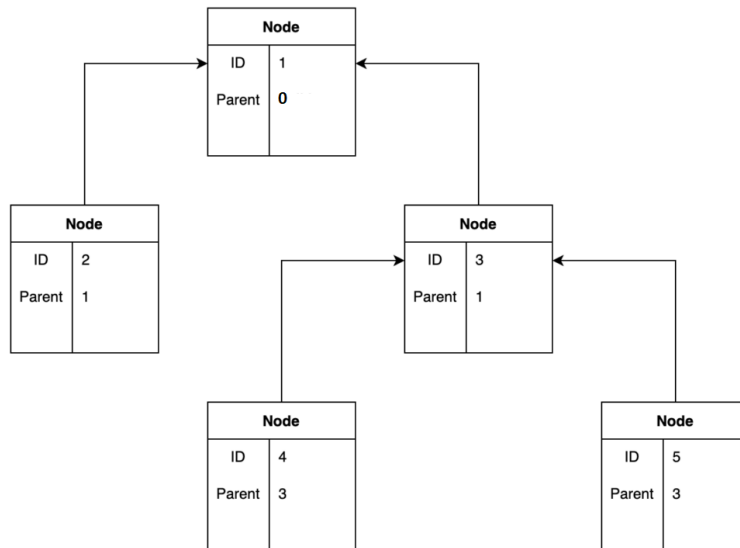


Figure 9: An example of tree data structure implementation in the database

4.1.3.2 How does it work and its role in security.

Enhancing efficiency and security of access control with tree data structures and recursive queries:

The tree data structure and recursive queries improve the efficiency and security of access control in the password manager compared to alternative approaches. They are used during password creation and when fetching passwords for a user has access to.

Password creation: When an administrator creates a password, they need the list of users with their public keys to make copies of encryption key. These users are both the users with roles connected to the parents of the chosen node and the users with roles connected to the chosen node itself.

Fetching passwords: When a user request password has access to, the passwords are the content of the node associated with the user's role and the content of its children nodes.

The main idea is to retrieve a node and its children (subtree), making it easy to obtain users with associated roles. The used PostgreSQL database engine supports Recursive Common Table Expressions, which enables the use of the **WITH RECURSIVE** clause to query the described structure. (PostgreSQL Documentation, 2023). This involves writing a query that

joins the table back onto itself and informs the database engine to recursively do this until it runs out of rows. The following code is an example of using WITH RECURSIVE to fetch tree using node id.

```
WITH RECURSIVE rectree AS (  
    SELECT *  
        FROM tree  
        WHERE node_id = (THE REQUESTED node id)  
UNION ALL  
    SELECT t.*  
        FROM tree t  
        JOIN rectree  
            ON t.parent_id = rectree.node_id  
) SELECT * FROM rectree;
```

The role based access control was successfully implemented by using the "file" table in the database and its "parent_id" attribute, along with Recursive Common Table Expressions.

The following illustration, figure 10, is an example of the tree data structure implementation for organizing roles and access control in the system. In this figure, for instance, a user has a role "Software engineer" has access to passwords associated with node "Tech" and all passwords associated with "Tech" children nodes (Development, Web, Frontend and other children...)

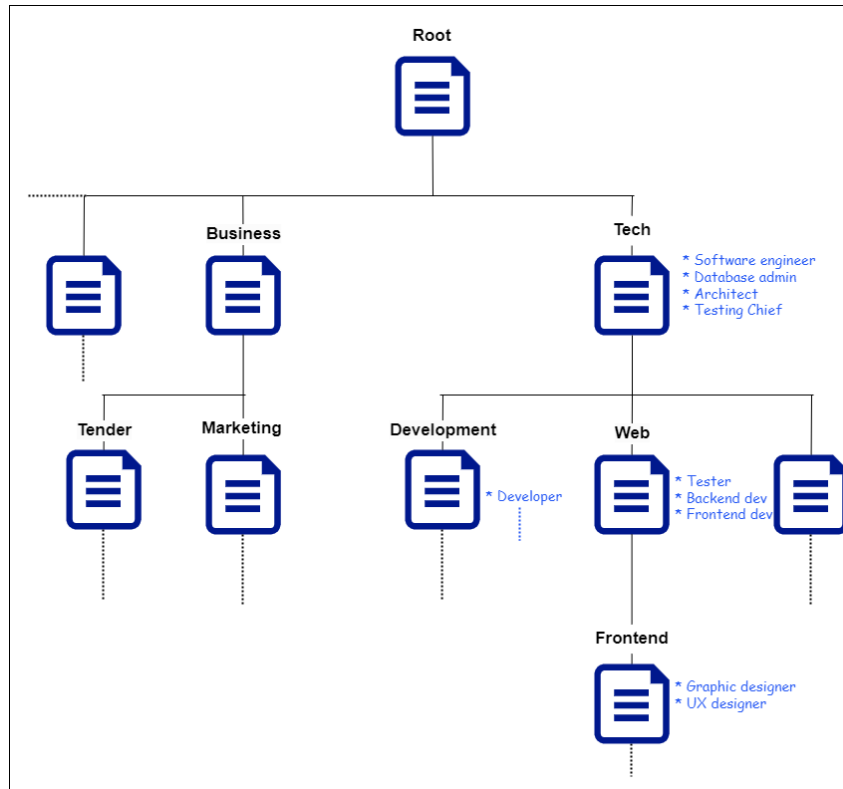


Figure 10: An example of the tree data structure implementation for organizing roles and access control in the system.

4.1.4 Zero-knowledge server architecture

Zero-Knowledge Server Architecture Implementation: As described in the System architecture [3.3.3] Favn Password Manager adheres to the zero-knowledge server architecture concept. The server is unaware of the user's sensitive information under this design, including their master password and any passwords saved in the database.

The implemented approach makes sure that the server only receives *hashed* master passwords and *encrypted data* (passwords and keys) by performing all encryption and hashing operations on the client side before sending the user's passwords to the server.

In the figure 11 below, the steps involved in the zero-knowledge server architecture for Favn Password Manager are illustrated as a workflow diagram.

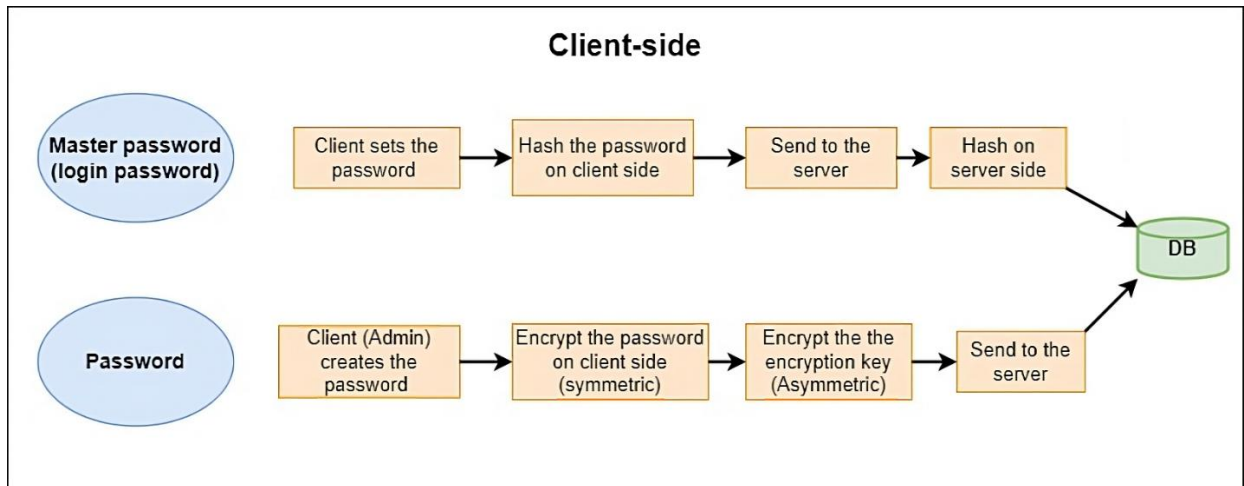


Figure 11: Workflow diagram illustrating the zero-knowledge process on client side.

4.2 Engineering results

This section covers the practical results obtained through the development of the custom Content Management System (CMS), frontend, and backend systems, as well as the security and authentication techniques used.

4.2.1 Backend system development

Custom Content Management System (CMS) architecture

The custom CMS for the product was developed using a modular approach, with three main files in the backend. The main files are `app`, `manager`, and `sql_statement`. Each file has a specific job and functions cohesively with the others to form the core of the system. The implementation of the tree data structure, as discussed in the [Scientific Results](#), was incorporated into the CMS to develop access control and user management. Additional service files for 2FA, encryption, and mail services were also implemented to support the system's functionality.

App (API): The `app` file is the launcher for the Flask application and houses the REST API. It includes all the endpoints with simple logic, focusing on handling request parameters, calling the relevant function from the `manager` file, and sending the appropriate response. Exceptions to this straightforward process include registration and authentication endpoints, which require specific logic within the `app` file.

Manager (Core): The `manager` file is the heart of the system, containing all methods that handle the main tasks. These methods are responsible for error handling, establishing

database connections for CRUD operations (create, read, update, and delete), and returning the desired values when called by the endpoints in the app file. The manager file relies on the sql_statement file for executing SQL queries and incorporates the tree data structure for efficient access control and user management.

SQL_Statement (SQL Queries): The sql_statement file contains all the SQL queries used in the system, as no ORM was needed. An ORM was deemed unnecessary due to the complexity of many queries that needed to be written manually. This file's primary purpose is to return the requested values to the methods in the manager file. These methods then open connections and send parameters without additional complexity.

Together, these three files act as the engine of the custom CMS, ensuring efficient handling of user management, access control, and other core functionalities of the system.

The following figure 12 illustrates the modular structure of the custom CMS and the flow of requests and responses between its main components.

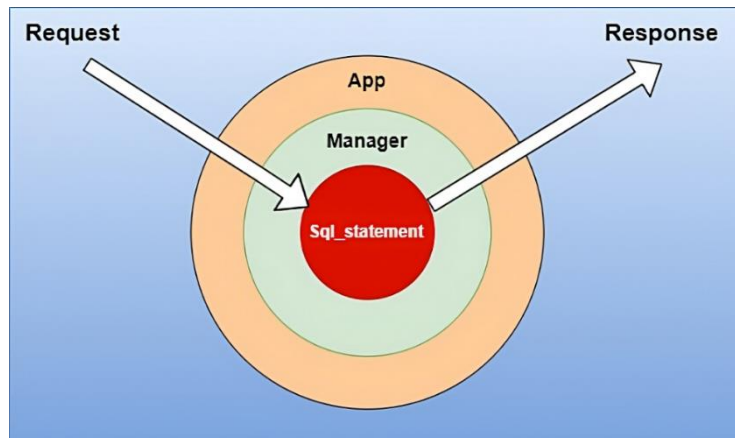


Figure 12: Modular structure of the custom CMS handling requests and responses.

4.2.2 Frontend Implementation

The frontend implementation of the password manager relies on the Web Crypto API, a JavaScript-based cryptographic API that is supported by modern web browsers (developer.mozilla.org, n.d.). This API allows the encryption of passwords on the client side, as described in the System Architecture [3.3.3]. In this section presents main functionalities implemented in frontend.

4.2.2.1 System main view

The System Tree is populated and structured using data from the backend, which uses the custom Content Management System (CMS) [4.2.1] and a tree data structure in the database. This configuration facilitates hierarchical management of password folder "nodes" and roles.

Figure 13 displays the main page of the admin dashboard, featuring the primary System Tree and roles (both available and unavailable).

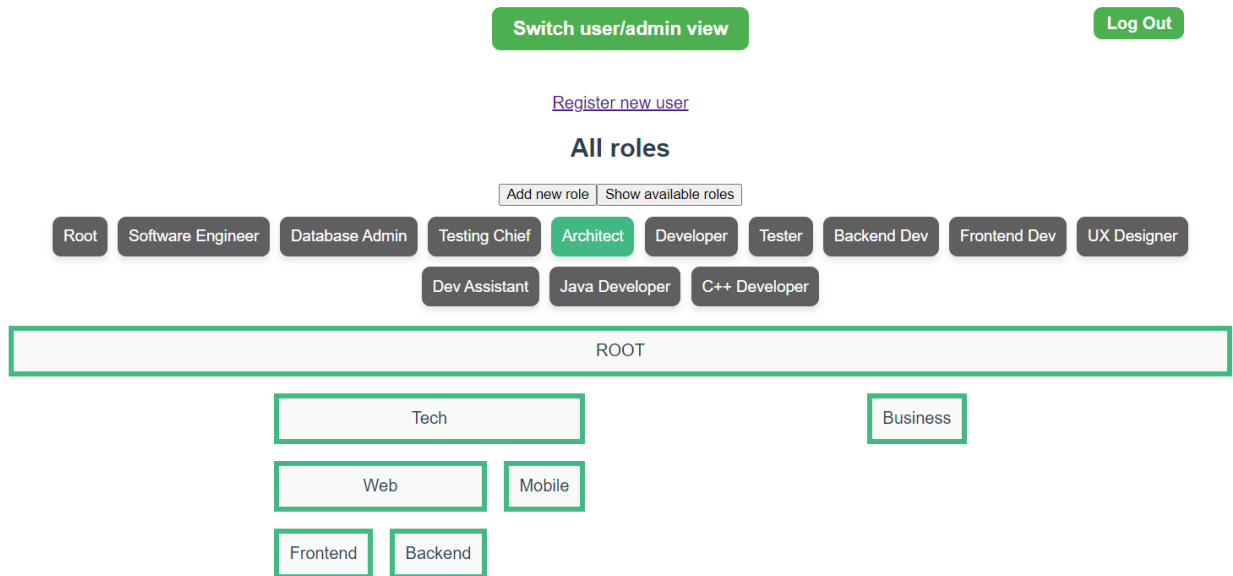


Figure 13: Main Page of admin dashboard with system tree and roles

4.2.2.2 Password encryption

The admin role is placed at the top of the system tree. During the registration of a new admin, the root role is automatically assigned, with no need to choose a specific role. The password creation process applied in the "InsideNode" page, where the admin adds a new password associated with the node.

The following steps outline the process of submitting a new password (a sequence diagram found in requirements document [appendix C]):

Click on the "Add New Password" button.

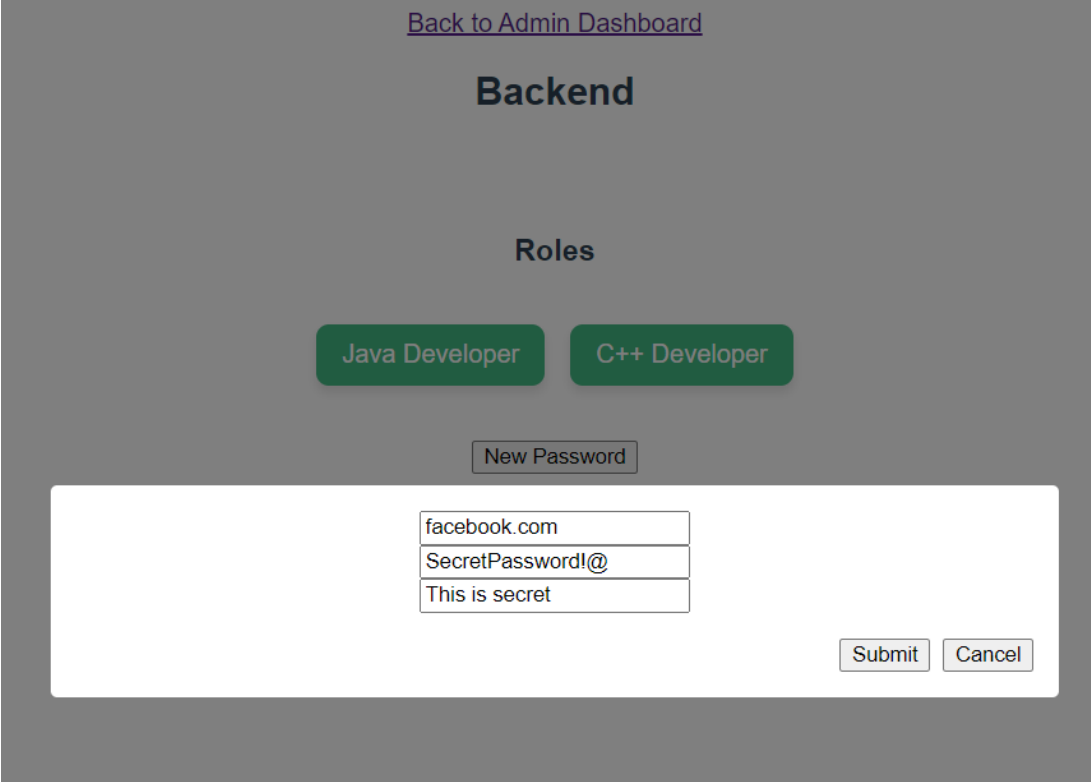
Call the `submitNewPassword(url, password, note)` function, which includes the following logic steps:

1. Generate an AES key.

2. Stringify the password data json object
3. Encrypt the password using the generated key.
4. Send the encrypted password to the server and wait for the response, which includes the added password ID.
5. Export the key to a Base64-encoded string.
6. Request public keys (this retrieves IDs for users who have access to the created password based on users' roles, and their public keys).
7. Loop through the users to make copies of the encryption key with their RSA public keys:
 - a. Import the user's public key from a PEM-encoded Base64 string.
 - b. Encrypt the key with the user's imported public key. The returned value is an ArrayBuffer.
 - c. Convert the ArrayBuffer into a Base64 string.
 - d. Add the user ID, password ID, and encrypted AES key to the list.
8. Send the list to the server.

By implementing these steps, the frontend ensures that passwords are encrypted on the client side and securely communicated with the backend.

Figure 14 below illustrates adding a new password by admin within a selected node (named "backend" in this example), employing the password encryption process steps.



The screenshot displays a web interface for managing a node named "Backend". At the top, there is a link "Back to Admin Dashboard". Below it, the word "Backend" is prominently displayed. Underneath, the "Roles" section contains two buttons: "Java Developer" and "C++ Developer". A "New Password" button is positioned below the roles. A white modal form is open, containing three input fields with the following text: "facebook.com", "SecretPassword!@", and "This is secret". At the bottom right of the form, there are "Submit" and "Cancel" buttons.

Figure 14: Adding new password within a selected node, which uses the encryption process.

4.2.2.3 Password decryption

The password decryption process is identical for all users, including admins and regular users. Both user types (admin and non-admin) use the same approach to view passwords. Additionally, admins use the approach to update data for the new registered users by creating copies of keys.

The following steps outline the process of viewing passwords (a sequence diagram can be found in the requirements document [appendix C]):

1. Fetch the RSA encrypted private key from the IndexedDB, which is decrypted during login with the master password.
2. Request the list of passwords that the user has access to, along with their IDs.
3. Request the list of keys that the user has access to, along with their associated password IDs.
4. Merge both lists based on the password ID attribute.
5. Loop through the merged list to decrypt the passwords:
 - a. Convert the encrypted key from a string into an ArrayBuffer.
 - b. Decrypt the key using the RSA private key.
 - c. Import the decrypted key.
 - d. Decrypt the password using the decrypted imported key.
 - e. Parse the data into JSON.
 - f. Add the decrypted password data to the list.
6. Return the list, which includes password data as JSON objects.

By following these steps, the password decryption process ensures that all users can securely view the passwords they have access to.

Figure 15 below displays the password list in the admin dashboard, accessible by clicking the "Switch User/Admin View" button. This button retrieves all passwords from different nodes into a single list, offering a complete overview of all passwords and making it easier for the admin to navigate them.

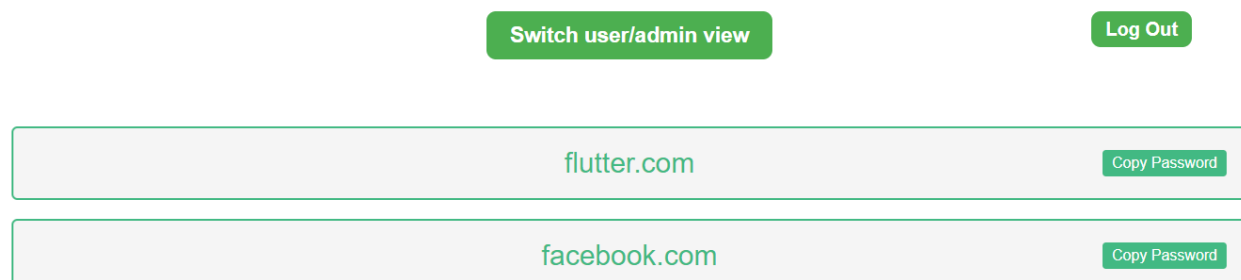


Figure 15: Password list in admin dashboard.

The user dashboard (non-admin dashboard) displays a similar view, but only includes passwords the user has access to, and does not have the “Switch User/Admin View” button.

4.2.3 Updating Passwords (Synchronization)

In the implementation of the system architecture, a specific issue needs to be solved. As previously described, during the password creation process and in the system architecture, the admin fetches public keys for users who have access to the password to manage the access. This implies that only users who are registered and have uploaded their public keys can receive their copies of the keys. Consequently, newly registered users who have not yet logged in and uploaded their public keys still need access to the passwords.

To resolve this issue, a database table called “Update” is used to track new users. During the backend registration process, when a user is registered and granted access to passwords, their status in the Update table is set to False, indicating that the user requires an update.

On the frontend, the admin fetches the Update list and updates the users by creating copies of the keys encrypted with the users' public keys. Once this is done, the admin sets the user's status to True.

Figure 16 below illustrates the Update process flowchart on the frontend.

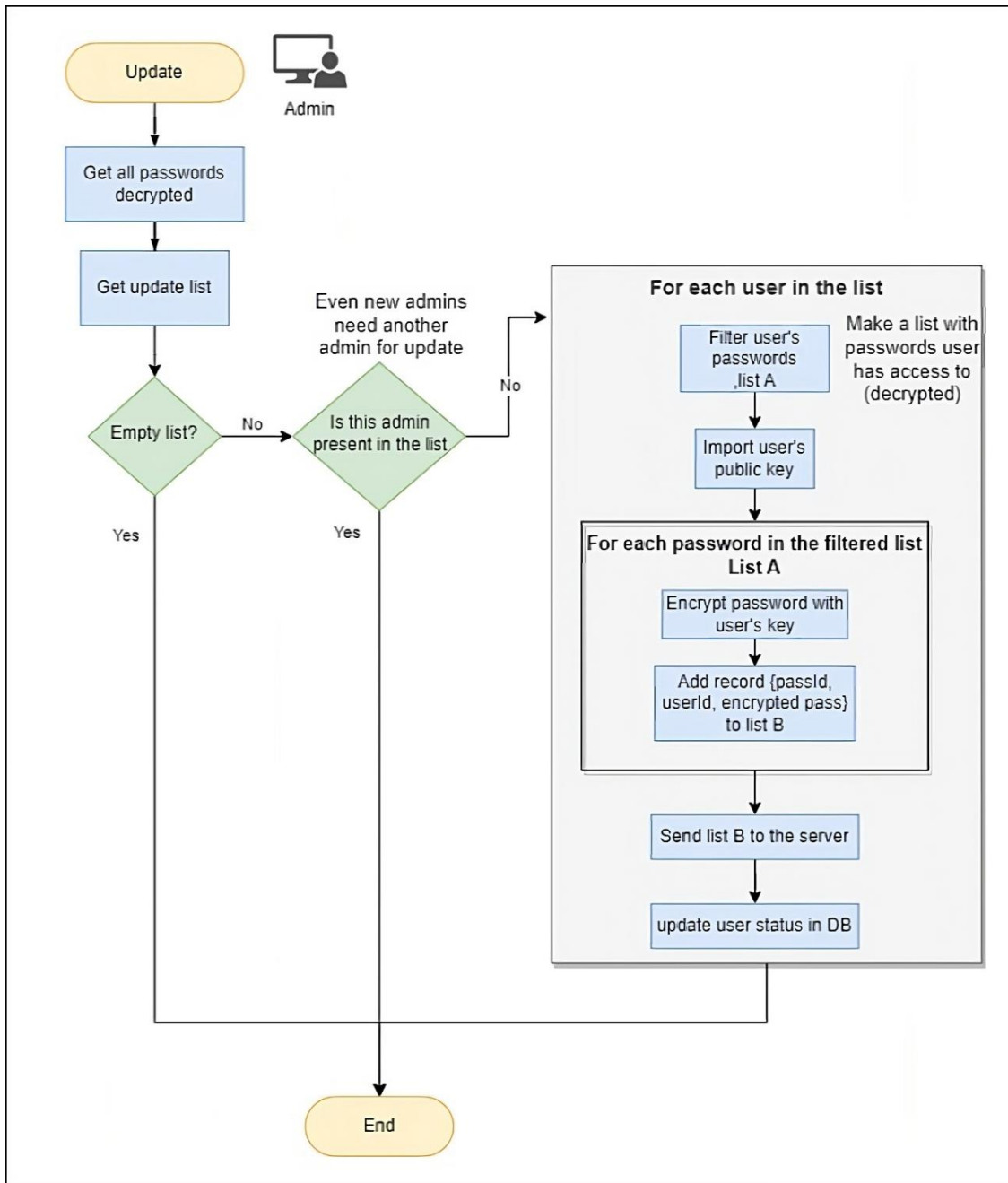


Figure 16: Illustration of the password synchronization process for new users in the frontend

4.2.4 Authentication

Authentication process:

The system's user authentication procedure, without the usage of two-factor authentication (2FA), is described in detail below. After the user has been successfully authenticated with the procedure described here, the 2FA implementation to verify the user will be covered in the subsequent section:

- 1- The user enters their email and password in the login form.
- 2- The client-side code hashes the entered password using PBKDF2 with SHA-512, 1000 iterations, and a key size of 64. The salt is a combination of strings with user's email, the used one currently is ("localhost" + "signup" + email), and it can be decided and modified by FavN on deployment.
- 3- The email and hashed password are sent to the server.
- 4- The server retrieves the stored user information (email, hashed password, and salt) from the database based on the provided email. The stored salt is generated using `werkzeug.security.gen_salt` with length 64.
- 5- The server-side code hashes the received hashed password from the client-side again using PBKDF2 with SHA-512, but this time with 4096 iterations and a key size of 64, using the stored salt.
- 6- The server compares the resulting hash with the stored hash in the database.

If the hashes match, the server grants access to the user, otherwise, access is denied.

Figure 17 illustrates the sequence diagram for the user authentication process without two-factor authentication (2FA) in place, showcasing the interactions between the client, server, and database.

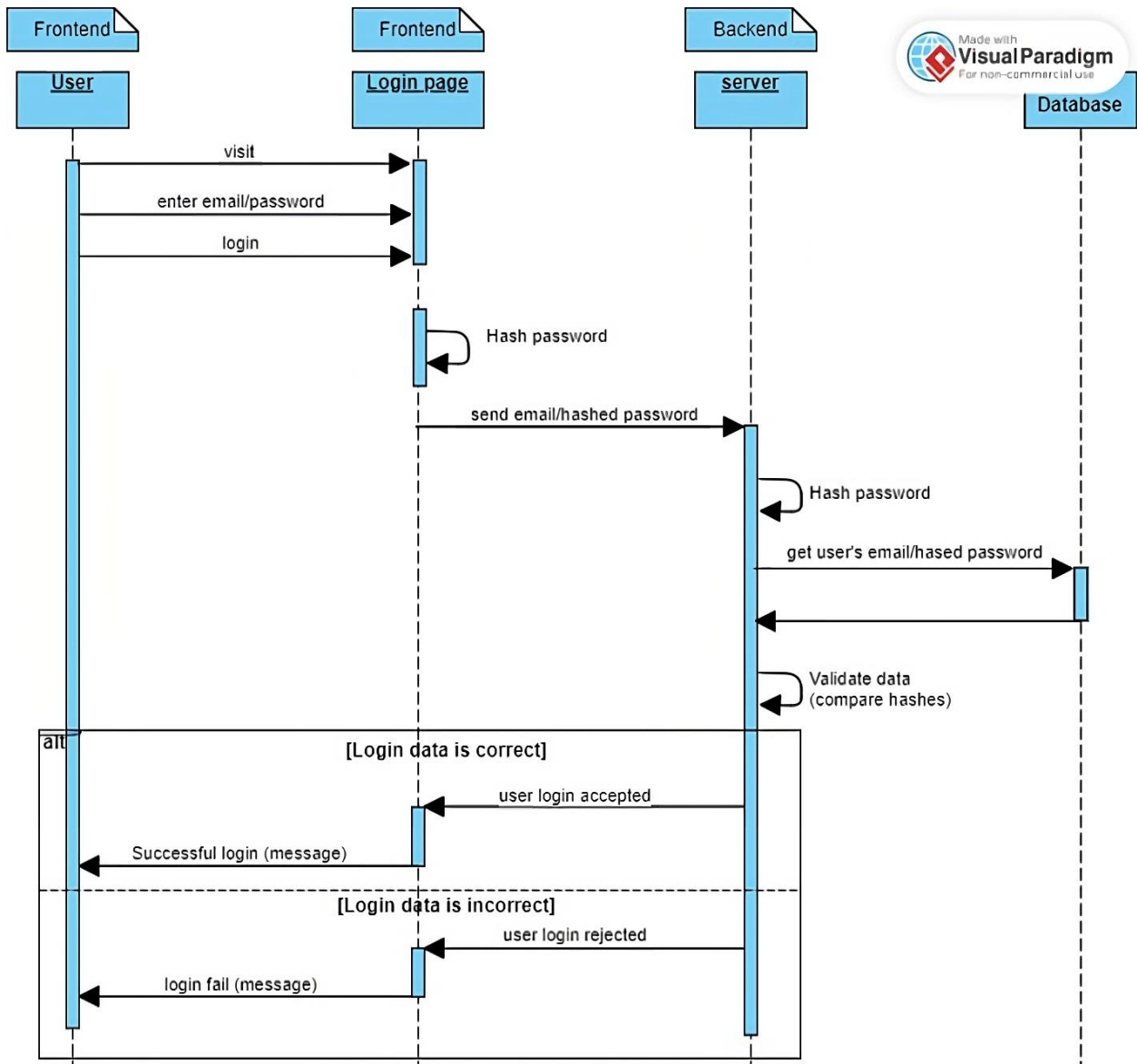


Figure 17: Sequence diagram of the user authentication process without 2FA

Two-Factor Authentication (2FA)

The authentication method implemented in the system is Time-based One-Time Password (TOTP) [2.5.2]. The authentication process consists of two main phases, registration and login.

User Registration:

1. When the admin registers a new user, the server generates a 2FA secret (base32-encoded string) for the user within the 'create_user' method in the 'manager' file.

2. The secret is encrypted using Fernet [4.1.2.2] imported from the known python package 'cryptography'.
3. The 'generate_qr_code(user)' method from the 'two_factor_authentication' service file is called to create a QR code containing the user's 2FA secret (decrypted within this method) for use with an authenticator app. This function employs the 'pyotp' and 'qrcode' libraries to generate the QR code, which includes the user's email and 2FA secret in the provisioning URI. The QR code is then sent to the user's registered email.
4. The user scans the QR code with an authenticator app (e.g., Google Authenticator) to set up 2FA for their account.

User Login:

1. On the client-side, the user retrieves the 6-digit token (valid for 30 seconds) from their authenticator app and enters it into the login page's 2FA input field.
2. The token is sent plain alongside the user's email and hashed password, as it is not considered secret information.
3. On the server-side, after validating the user's email and password (as described in the authentication section above), the server retrieves the user's information, including the encrypted 2FA secret. Next, it calls the 'is_2fa_valid(user, token)' method from the 'two_factor_authentication' service file.
4. This method decrypts the user's 2FA secret and verifies if the provided token is valid. It initializes a TOTP object with the user's 2FA secret and uses the object's 'verify()' method to confirm the token's validity for the given time. The 'verify()' method internally calls 'TOTP.now()', which returns a token that matches the provided token. The authenticator app on the user's phone uses the same underlying logic of calling 'TOTP.now()', ensuring a successful match.

4.2.5 Security

The project focuses on security best practices along with handling the OWASP Top 10 vulnerabilities:

- SQL Injection: Handled both on the frontend and backend.
 - Frontend: Input validation using the vee-validate package with specific rules for each input field on the login and set password pages (the only pages accessible to non-users). (vee-validate.logaretm.com, n.d.)
 - Backend: Prepared statements (%s) are used when executing SQL queries to prevent SQL injection.
- Cross-Site Scripting (XSS): The system is protected against XSS through the standard use of Vue, which automatically escapes any user input rendered in the DOM, preventing the execution of malicious scripts. No use of setInnerHTML is made (Vue.js, n.d.).

- Authentication: Best practices are followed by using hashing on both the frontend and backend, along with two-factor authentication (2FA).
- Sensitive Data Encryption: Appropriate system architecture is employed to encrypt sensitive data before being stored in the database. In addition, no sensitive data is saved on the client-side, except for the RSA keypair, which is encrypted using the user's master password.
- Authorization: Is handled in the backend, with all endpoints protected by a decorator wrapper function (@token_required) and with use of cookies to check if user has admin permissions by user's id. Authorization is also processed in frontend with Vue router conditional rendering of pages to protect the access to the administration pages, such as admin dashboard, inside node and register page.
- Cookie-Based System:
 - The server uses a cookie-based system to store refresh tokens and access tokens, instead of local storage, which is less secure.
 - Cookies are also used for the authorization process. The user ID is extracted from the token and checked for admin privileges, protecting endpoints that should only be accessible by admins.
- Force users to use a strong login password with recommendations from Microsoft when they set it up, in set-password page.(support.microsoft.com, n.d.)

By focusing on these security aspects, the project provides a secure environment for users and their sensitive data.

4.3 Administrative results

4.3.1 Timesheet and activity

At the end of the project, a total of 534 work hours were invested. The pie chart below provides a visual representation of the work distribution across various tasks and components [Figure 18].

All schoolwork requirements (preliminary project plan, poster and poster preparation etc.) counted with Documentation.

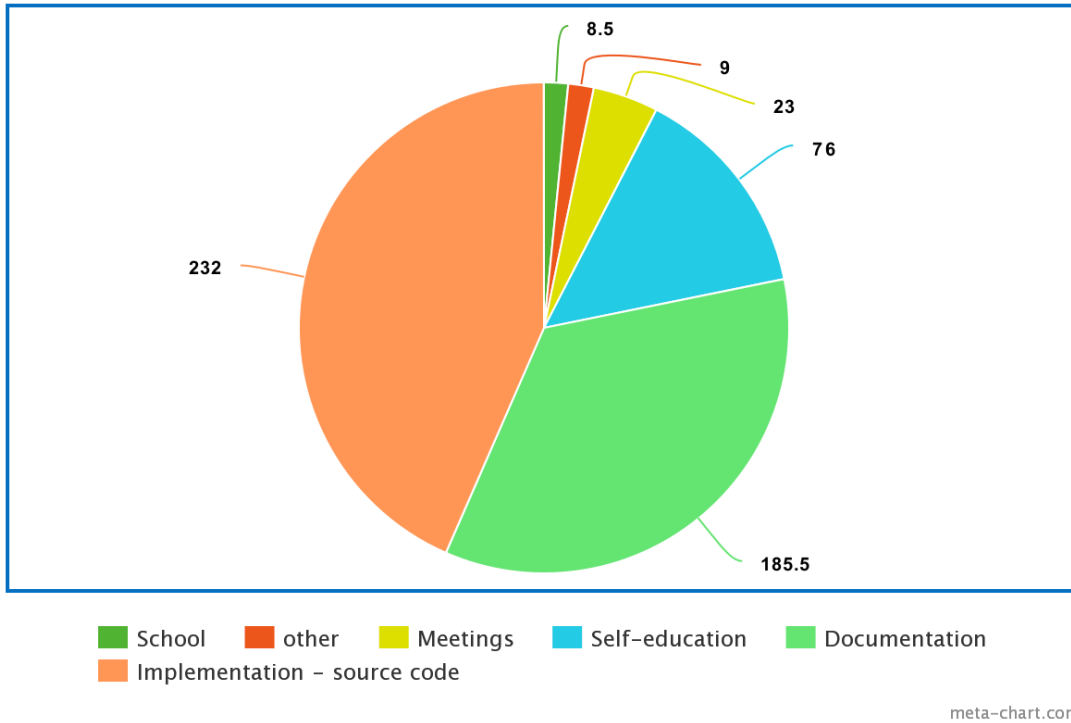


Figure 18: Work distribution pie chart

Figure 19 displays the actual schedule plan, represented as a Gantt chart.

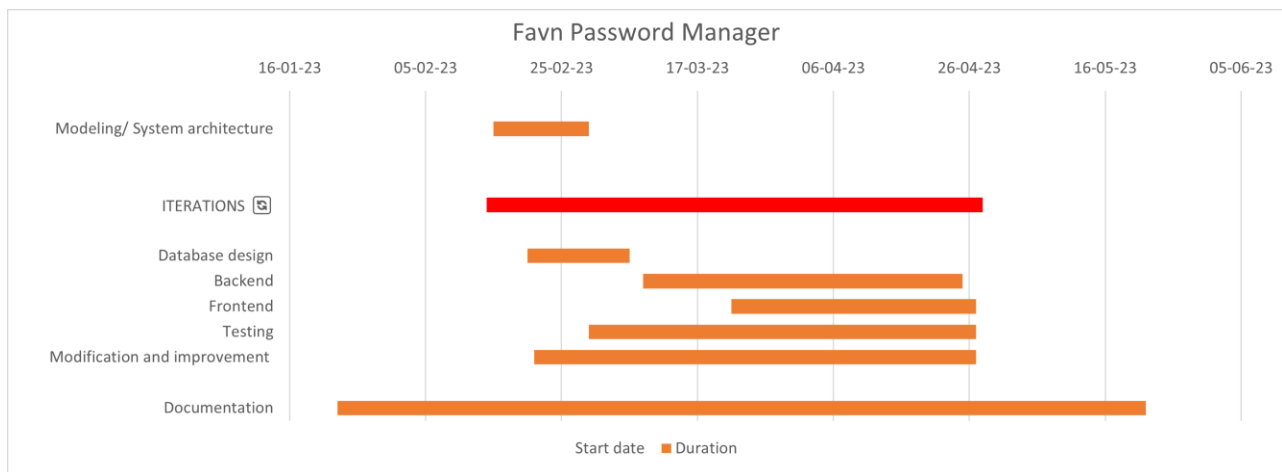


Figure 19: Project's Gantt diagram

4.3.2 Methodology

Throughout the project, the Rapid Application Development (RAD) methodology was chosen to ensure efficient and flexible progress. By working alone, the focus was on following an organized workflow, which was achieved within the estimated specified times using a traditional task board with sticky notes. Notes were written and updated at any time and from any location, even using a note-taking app on a smartphon. This approach allowed for prioritization and tracking of tasks while adapting to changes as needed.

By following the RAD methodology and engaging in open iterations without fixed sprint sizes, the project remained agile and adaptable. Regular meetings with the client and supervisor were held to discuss updates and maintain clear communication. The primary goal was to complete the development and programming while ensuring high-quality results and leaving enough time for handling any unexpected situations or refining documentation.

The implementation of this methodology turned out to be effective by achieving the desired results in terms of product functionality, process management, and documentation. The product was as expected and gained the approval from both the product owner and the supervisor. All necessary documentation was finalized and resulted efficiency of the selected strategy.

5 Discussion

The discussion section explores the relationship between the achieved results and the research question. It provide an analysis of the strengths, weaknesses, and implications of the implemented system.

5.1 Scientific results

In this subsection, the focus is on understanding the results in relation to the research question, evaluating the developed password manager's success in handling the key design principles, and best practices for hierarchical access based on roles.

5.1.1 Features

The features implemented in the Favn Password Manager collectively shape the final product, aligning it with the expectations of the product owner, Favn Software. These features gained from extensive research into existing password managers and their functionalities. They are important to fulfilling custom requirements and securing data, which is the main principle in any password management system.

The research focused on well-known password managers like LastPass and 1Password. The research also had extra exploration on secure services such as Tutanota, which used encryption within a unique architectural framework. The information gleaned from these services helped in establishing a robust and secure system, incorporating custom requirements like role-based hierarchical access to passwords, password sharing among users, and unlimited flexibility in role management (adding and using).

However, an expanded research scope encompassing more password managers like NordPass(nordpass.com, n.d.) and DashLane(Dashlane, 2019), could have potentially given additional ideas to improve system robustness. The challenge here, however, lay in navigating the complex architectures and limited available information of these systems.

The implemented approach in authentication served its purpose, but the Secure Remote Password protocol used in 1Password could have been a better choice if it was feasible to use.

5.1.2 Encryption techniques

The strategic application of both symmetric and asymmetric encryption within the system was essential. Their combined usage contributed towards the fulfillment of the main requirements specified by the product owner.

Symmetric encryption was widely used in various parts of the system, especially in the encryption of passwords and other important data like 2FA secrets. In this context, the selection of the AES algorithm was informed by its recommendation for such scenarios. Similarly, Fernet, with its benefits, was identified as the optimal choice for encrypting 2FA secrets.

Asymmetric encryption was applied into the system to achieve controlled access password sharing, the main feature. The application of best practices in the usage of algorithms and approaches was ensured as far as possible, which included considerations such as key length, hash, among others.

Implementing encryption at the frontend proved to be a tough task, mainly because of the complexity involved in debugging. The debugging process was tricky due to factors like byte-level detail, variance in input/output types, length, and more. Furthermore, the use of the Web Crypto API in the frontend added to the challenge as it is a low-level interface(developer.mozilla.org, n.d.). This required a solid knowledge of cryptography and encryption for good handling.

Weaknesses:

Even though alternative approaches involving encryption processes may reduce overhead and enhance system performance, the main focus was on ensuring the security and integrity of the system, rather than its performance. Thus, the trade-off was considered acceptable.

5.1.3 Hierarchical access

The implementation of a tree data structure within the database, supplemented with appropriate server logic and SQL queries, played a main role in establishing the hierarchical access feature. The data, transmitted from the backend to the frontend, arrived in a ready-to-use structured format, simplifying the building of the tree system and its display on the admin dashboard.

The decision to use a relational database was appropriate, as it helped keep data organized and in order, thereby simplifying operations and allowing for smooth implementation of requirements.

The table "File" could have been more accurately named "Node" to better describe its purpose. However, due to the already established SQL queries, methods, and endpoints,

along with the time required to make the necessary changes, the table was not renamed, more details in engineering results discussion section [5.2.3].

The overall design of the database architecture was normalized and highly effective without data duplicates, and the final version was neat. The table diagrams were clear, and the tables looked like a simple closed circuit with all tables connected providing organized data fetching.

The following figure 20, depicts the database tables diagram.

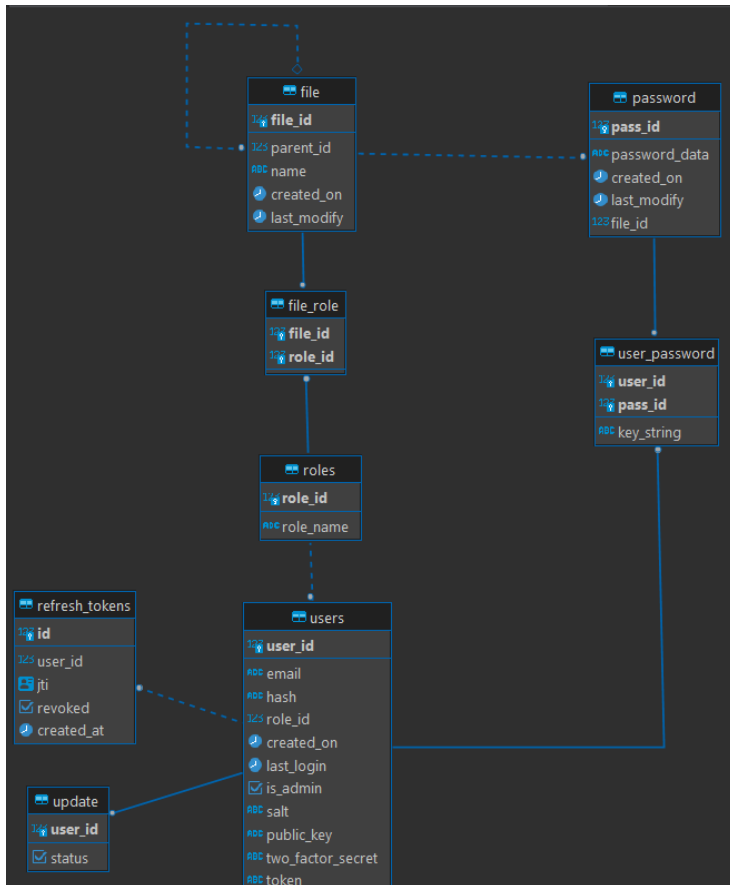


Figure 20: Database tables diagram

The hierarchical access leveraged a tree data structure in the database along with Recursive Expressions for a full solution. This well-integrated approach functioned effectively across various processes, from password creation to fetching passwords. The problem of implementing hierarchy within the system was solved through the use of the "file" table referring to itself and the adept application of the parent_id attribute and Recursive Expressions. This simplicity and strategic execution made the desired task achievable.

5.1.4 Secure Zero-knowledge server architecture

The term "Zero-Knowledge server" refers to a design approach where the server lacks any knowledge of the user's sensitive data. In this context, the sensitive data pertains to the users' master passwords (used for login) and the passwords that require management. It also refers to the fact that all encryption-related processes must occur on the client-side.

Thus, the term "Zero-Knowledge server architecture" does not refer to the server's architecture per se but to the overall system architecture designed to keep the server ignorant of sensitive information.

In the developed product, this architecture is applied using the described system architecture. All encryption and decryption processes occur on the client side. This indicates that no password data is accessible to the server. In order to prevent the server from knowing the login credentials, master passwords are hashed on the client side before being submitted to the server.

The client-side solution's encryption of the RSA keypair with the user's master password is a crucial security feature. The private key will remain safe even in the worst case scenario of a breach.

Since two-factor authentication (2FA) is required upon login, access token handling was added to the system architecture to boost user convenience by removing the requirement to log in each time a website is accessed. This led to a bit of conflict between the desire for easier login and the need for keypair encryption. The system requires the master password once to decrypt the keypair. For admins, this process can be annoying as the system prompts for the master password after every main operation on the main page (like creating, editing, or deleting nodes) that require a page refresh. Upon each refresh, the system again asks for the master password. Future improvements could aim to fix this inconvenience while maintaining stringent security measures.

5.2 Engineering results

This subsection reflects on the system's implementation, challenges faced, strengths and weaknesses, and provides recommendations for future improvements and research.

5.2.1 Functional requirements

The system's development was mainly driven by a range of functional requirements detailed in the project requirements document. The primary focus was on realizing the main features.

Table 2: Functional requirements implementation status

Functional requirements	Implemented	Not implemented
19	15	4

These requirements corresponded with 11 user needs detailed in the vision document. The following table represents the status of implementation prioritized according to these needs.

Table 3: Status of user's needs implementation

Priority	Implemented	Not implemented	Partially implemented
high	6	0	1
Medium	1	1	0
low	0	2	0
TOTAL	7	3	1

Due to time constraints, the implementation of three requirements was not possible.

However, six high priority needs, and one medium priority need are implemented, as shown in the above table. They led to the creation of a product with the desired features such as *secure* and *hierarchical access* for password management.

On the other hand, several requirements are not implemented. As mentioned in the table above, these were one medium priority and two low priority needs. The system could work well without these components.

Further information about the **implemented requirements** can be found in appendix C of the requirements document.

The **unimplemented requirements**, though not overly complicated, could still be completed. These include:

- User promotion from user to admin: Theoretically, this can be simply accomplished by modifying the 'is_admin' attribute in the users table of the database to grant access to the admin dashboard. Also, the user's role would need to be linked to a role associated with the root node, allowing access to all passwords from the top of the system tree. However, this theory must be thoroughly tested and enhanced to ensure its efficacy in solving the problem.
- Edit login password: This could be similar to the 'set-password' method already implemented. With backend procedures for temporary token creation and handling already in place, password editing could follow a similar approach. Again, this solution needs testing and refinement to ensure its functionality.
- Edit password: Theoretically, this solution might follow a similar process to password creation by the admin. However, it would involve updating instead of creating. This could be achieved by sending the updated item and its id to an endpoint accepting a PUT http method. The additional step would require the admin to decrypt the password, re-encrypt it, and send it back to the server in its encrypted form.

Regarding the **partially implemented requirement** – the ability to delete passwords – time constraints prevented its full implementation. As a temporary solution, a cascading delete operation was added to the foreign key linked to the relevant node in the database. The product owner accepted this solution as sufficient, stating that in a worst-case scenario where a password contained incorrect information, they could either add a new one with the correct data, or delete the entire node, effectively removing all associated passwords.

5.2.2 Non-functional requirements

- ✓ The system's non-functional requirements, as detailed in the vision document, have been successfully met. Over 80% of the central file (the manager file) in the backend has been subject to unit testing, with each test covering multiple cases.
- ✓ Security, the main requirement was achieved in the system. Encryption technologies were applied alongside a system architecture with a zero-knowledge server to optimize security and product's safety.

- ✓ Regarding data persistence, the system used an SQL relational database with the PostgreSQL DBMS. Both the database and server operate in separate containers that are interconnected, with Docker Compose facilitating their management.
- ✓ The system's frontend was thoroughly tested on both Chromium (Chrome) and Firefox using the latest versions, specifically Chrome Version 113.0.5672.92 (Official Build) (64-bit), and Firefox Version 113.0.1 (64-bit). Testing was not conducted on Safari for Mac users.
- ✓ Version control was maintained throughout the project using Git, and the project being uploaded to GitLab in two separate repositories for the backend and frontend.

5.2.3 System architecture

The chosen system architecture robustly supported the project's central objectives: security, hierarchical controlled access, and implementation of a zero-knowledge server. This architecture has been detailed in various sections throughout this paper. It's worth discussing how this solution has evolved and improved over time, particularly after multiple suggestions led to critical modifications.

Initially, the concept was to consider a node as an actual file, hence the term "File" was employed in the backend and database (this is the reason "File" word exists in several parts in the core). It was envisioned that the admin would create JSON files, insert passwords into these files, and then store a copy of these files on the client side. The idea also included generating the RSA key pair directly from the client's machine, for instance, using the `ssh-keygen` command. Due to the limited access a browser has to the client-side file system, both of these approaches were later determined to be unsuitable. The strategy also featured a security flaw, which is if the key to one file was stolen, all the passwords in that file would be at risk. The final solution, however, secures every password with a unique key, so if a key is compromised, only one password is affected.

One further idea was to only use RSA's asymmetric encryption along with password encryption. In this case, instead of producing copies of the key when an admin creates a password, they would retrieve the users' public keys and encrypt the password directly. However, this strategy was also dropped.

The final approach, which uses symmetric encryption (AES), was chosen as it offers two advantages:

- ✓ It minimizes the risk of data loss in the event of a key compromise.
- ✓ This approach, which first uses AES to encrypt the password and then uses the RSA public keys to encrypt the symmetric key, is ideal for the functionality of editing of

password by the admin. The admin can easily edit the password and update it in the database. In this case, the symmetric key remains unchanged, which means that the users' copies also stay unchanged. Thus, there is no need for the admin to update keys after editing a password.

The architecture has been checked for possible attacks from many sides, it found that there is a possible cryptographic attack:

The robustness of the system architecture has been tested against potential security breaches, including cryptographic attacks. One possible scenario entails an internal assault from a user seeking unauthorized access to elevated credentials within the system tree. This could potentially be achieved through a cryptographic attack known as the "Discrete Logarithm Problem Attack". It is a known attack on asymmetric RSA encryption.

While this remains a complex and mathematically problem (Bach, n.d.), mathematically details will not be discussed in this paper.

To illustrate how this attack might be executed on this system, consider the following scenario:

When two users decrypt their copies of a key, they get the same key (symmetric key used with password encryption)

- A system user, User1, has access to a key X encrypted with his public key (Y1: his encrypted copy of key).
- A second user, User2, has access to the same key but encrypted by his public key (Y2: his encrypted copy of key).
- Both keys decrypted (X1 and X2) are the same, symbolized as X, $X=X1=X2$.

In a situation where User2 gains unauthorized access to the database and fetch the encrypted key Y1, they could theoretically launch a discrete logarithm attack to uncover User1's private key. This would enable User2 to decrypt all passwords accessible to User1.

This means: User2 has both X1 and Y1 -> User2 could, in theory, employ the discrete logarithm problem attack to crack Y1 and obtain User1's private key.

Again, while this is purely hypothetical and such attacks is extremely difficult to execute in reality nowadays, this vulnerability should not be overlooked in the overall discussion of the system's security (Bach, n.d.; Cryptography Stack Exchange, n.d.).

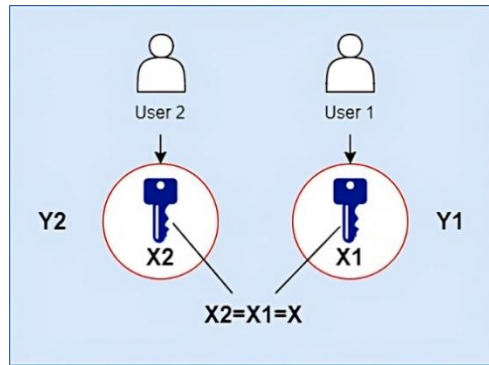


Figure 21: Discrete logarithm problem attack illustration.

5.2.4 Backend

The Flask framework served its purpose in this project and enabled the development of a straightforward server with a REST API. Main features include the development of endpoints and logic methods, all implemented using plain Python. While Flask supported packages could have been used for tasks like form handling, the decision was made to implement processes such as registration, login, and token handling manually. This approach allowed for a more hands-on application of research findings and simplified the debugging process. Encryption was implemented through specific packages, while Flask extensions were used for additional services, such as a mail service to send credentials to registered users (a link to set the password and 2FA QR code).

In this project, the backend operates as a small-scale system, without the need for extensive production systems or frameworks. Flask offers simplicity, a lightweight structure, and ease of use that provide rapid development. The straightforward Flask documentation and its supportive community of developers make the development process easier. They provide helpful resources like tutorials and guides.

The implementation of the CMS architecture detailed in section [4.2.1] within the Flask framework offered an efficient solution. The simplicity of this core structure makes it easy to follow and further develop if needed.

Although the development of unit testing in Python posed some challenges due to unfamiliarity with the language, the principle of unit testing was well understood.

5.2.5 Frontend

The selection of the Vue framework was motivated by the project's specific requirements as well as personal preferences. Its popularity and robust documentation made it an appealing choice, considering potential future development of the system by the product owner or other bachelor's students.

Vue is a solid framework from a security standpoint, it provides essential security features through its packages. These features prevent common attacks such as SQL injections and Cross-Site Scripting (XSS) by default, as outlined in [4.2.5]. Since the system could be developed using Vanilla JavaScript "plain JS without framework", the use of Vue was justified, given the small number of views and components. Vue simplicity counts as a bonus in the project.

The frontend implementation delivered favorable results, successfully fulfilling the key goals of the project. Specifically, it allowed for the creation and encryption of passwords by the admin, and decryption by authorized users. This setup is fundamental to achieving a zero-knowledge server, with encryption being carried out on the client side.

The usage of Web Crypto API for cryptography and IndexedDB for data storage proved to be good choices because both are supported by browsers, and no extra packages needed. All functionalities, including role assignment, node addition, node editing, password creation, and user registration, are easy to use and user friendly.

However, certain shortcomings were identified in the frontend solution:

- **Performance:** While safety was prioritized over speed in the project requirements, the performance could be improved. Some processes, such as encryption and tree building, were somewhat heavy and could be optimized. The tree building process, in particular, was quite heavy as it required reformatting data retrieved from the backend into a hierarchical structure (set children for each node) for display.
- **Node Naming:** When creating or renaming a node, the system should prevent name duplication. As a temporary solution, the UNIQUE constraint was removed from the "name" attribute in the File table to avoid database conflicts. This does not affect the main solution as node IDs are used for identification.
- **User Interface:** Although the interface is not the primary focus of the system, efforts were made to make it as user-friendly as possible. The Universal Design 7 Principles were considered to an extent. Even though the WCAG 2.0 principles were not strictly adhered to, the text content is readable and understandable, and the system is easy to use. For instance, admin functionalities are clearly visible and accessible with few

buttons, and system tree management is straightforward with options readily available with node selection.

The following figure displays the pop-up menu that is triggered when a node is clicked.

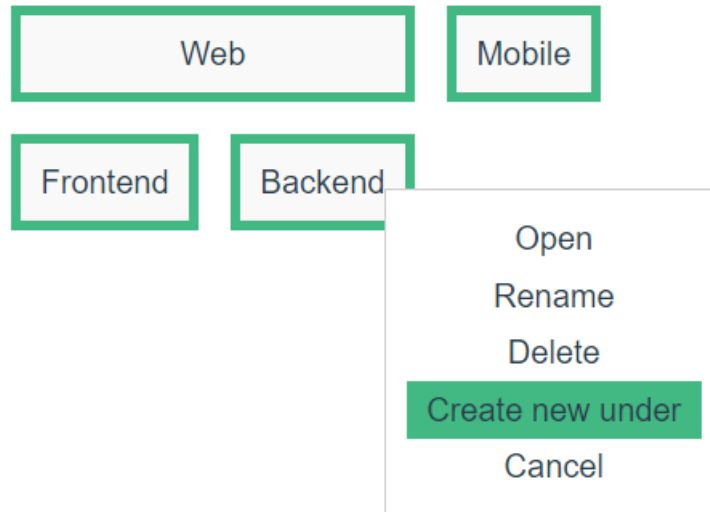


Figure 22: Node management pop-up menu

5.2.6 Security

Security takes center stage in the design and implementation of this system, with the execution of all principles enumerated in the [Research Methodology](#) [3.2]. The system architecture was custom-designed to provide security measures and to ensure the safekeeping of sensitive data. Further details regarding security considerations can be found in the engineering results section [4.2.5].

As mentioned in the vision document, system deployment is beyond the scope of this project. Therefore, secrets for local hosting are stored in a (.env) file in the backend. In a real-world scenario, FavN could use GitLab's environment variables feature during deployment. Similarly, while the current system uses HTTP, HTTPS with SSL/TLS can be implemented by FavN's DevOps team during the system's deployment to apply real world security considerations.

Moreover, considering the pivotal role of the admin in the system, it is important to ensure their security. A reasonable theoretical solution is the use of a hardware key for admin authentication. Such devices, like Google's Titan Security Key (Google Cloud, n.d.), offer an additional layer of security by requiring a physical object for user authentication.

5.2.7 Authentication

Authentication in this system forms an essential security layer within the implemented solution. It follows recognized standards and best practices, deploying hashing on both client and server sides. This approach contributes to achieving a zero-knowledge server and in increase the overall system's security.

The current system applies robust hashing mechanism with SHA-512 combined with PBKDF2, an algorithm known for its strong output, on both the client and server sides. However, in light of the growing attempts to break hashes using large hash tables, a higher level of security might be achieved by introducing a slow password hashing algorithm such as Argon2, bcrypt, or scrypt on the server side. These algorithms enhance security defenses against a variety of attacks, including brute-force and rainbow table attacks (openwall.info, n.d.).

Due to the main role of security in this project, user authentication forms a non-negotiable requirement. Therefore, the two-factor authentication feature is mandatory for all users. Although this requirement might be viewed as less user-friendly and slightly annoying, it's a necessary procedure considering the importance of safety in password management systems.

5.3 Administrative results

The selection and implementation of the project's methodology proved to be effective in achieving the desired results. Since this project was conducted by an individual student without a team, the challenge was to pick an approach that promotes agility and organization. Scrum was not considered an ideal fit due to its team-based structure. On the other hand, the Waterfall methodology, with its rigid progression, can pose difficulties in refactoring and modifying solutions. This led to the choice of the Rapid Application Development (RAD) methodology which provided the required agility and dynamism for modification and improvement. As noted previously, the system structure was changed several times to meet all the main goals and features.[5.2.3]

Implementing RAD from the outset proved to be an excellent decision. The project's focus, as detailed in the task description and vision document, was on the system's desired functionalities, such as security and controlled access, rather than on web design or the graphical user interface. This required a parallel development approach for the backend, database, and frontend, as reflected in the Gantt diagram in the administrative results section [4.3.1]. Initial efforts focused on the database and backend to provide the base CMS features, but then, each feature was implemented fully, involving modifications to the database, addition of feature logic in the backend, and implementation in the frontend.

The approach helped the timely delivery of all project components. With a total of approximately 534 work hours distributed appropriately over the project period, the process unfolded smoothly without pressure on the student [4.3.1]. Therefore, the product was delivered on time, leaving extra time for focus on the main report (this paper). The final were well received by both Favn and the project supervisor.

5.3.1 Reflection

My interest in cybersecurity motivated me to choose this project, as I wanted my bachelor's thesis to focus on a cybersecurity-related issue.

I did not have a team and I worked alone on my thesis project the entire time, facing the difficulties and enjoying the challenges that came with it. Given that I was able to deliver the product as expected, I am quite pleased with the outcome. The encryption aspect was particularly challenging, as it required extensive research and the implementation was more complex than the theoretical aspects. However, this challenge also made the project interesting.

Each task was finished on schedule, and the entire process was well-organized. Many skills and knowledge acquired during my data engineering studies were applied in this project, such as pre-work, documentation, various methods, information gathering, project administration, modeling, designing, and diagram creation. I also took advantage of many technologies that I had learned in the study. This thesis was for me as an approval that software project management and engineering is more than coding.

Working alone on this project was a good experience, as I followed a consistent approach throughout the semester. My passion for the project drove me to deliver the product I wished. I believe that the project management methodologies I used worked well and made the development process smooth. Generally, I'm satisfied with the hard work, techniques in problem solving, and the final results of this project.

6 Conclusion and Further Work

The project delivered successfully fulfills all core features requested in the assignment and answered the research question [\[1.2\]](#). By adhering to best practices in the security field, the sensitive data is well protected, and a system core is implemented to control access to passwords hierarchically. The main functional and non-functional requirements mentioned in the vision document have been implemented, with some minor deviations.

Should the project be expanded or developed further, this paper will serve as a valuable guide, even if the research question changes, as it focuses on password management, which inherently involves cryptography for securing sensitive data. The final product could be ready for deployment after fixing some issues discussed in the discussion section and undergoing testing by the product owner.

This paper provides a comprehensive answer to the question: "What are the key design principles and best practices for developing a secure password manager with hierarchical access to passwords based on roles?"

Suggestions for **Further work**:

Features to be completed:

To fulfill the unimplemented user requirements described in the engineering results section with brief suggested solutions for each, the following features should be considered:

- Change a user's permission from user to admin.
- Edit login password.
- Edit existing passwords.
- Continue and improve the process of deleting passwords.

Updates:

- The current solution stores the RSA keypair encrypted at the client side for enhanced security, limiting its use to a specific browser. An alternative solution would be to save the keypair in the database, but the current approach was preferred due to the potential risks of a database breach and subsequent decryption.
- A minor inconvenience in the frontend when the admin modifies the system tree (add node, rename node, delete node) since the code refreshes the page and prompts the admin to re-enter their master password to decrypt the stored keypair for encryption processes.

Extra features:

To improve the solution, the following additional features (not required by FavN) could be considered:

- Users management: A dedicated page for the admin featuring a table with user details, email, role, and more.
- Users management: The ability to edit or delete users.
- Interface and styling improvements, focusing on WCAG 2.0 principles.

Social Impact

This project has varied impacts on the community can be considered. The developed product Favn Password Manager has multifaceted impact which can be understood by evaluating its influence on three areas: the economy, society, and sustainability.

Economic Perspective:

In a world that's increasingly digital, businesses, organizations, individuals are making major investments in data security, and more data being digitalized. More digital data requires more robust security solutions to safeguard this information.

This project effect positively to the economy by providing a secure and effective password management solution. Companies that adopt this solution could minimize their cybersecurity risks, leading to fewer instances of costly data breaches, and thus, potentially save substantial financial resources.

Societal Perspective:

The Favn Password Manager contributes to our society's digital resilience. By providing an effective solution for password management, it supports individuals and businesses in their effort to maintain security in their digital interactions and transactions. This is crucial, especially in light of the rise in the amount of cybercrimes. The software's role-based hierarchical access feature ensures a responsible and ethical practice in data handling, permitting only those with the necessary authorizations to gain access to specific information and functionalities. This not only upholds professional standards but also adds a layer of protection against potential misuse of access privileges.

Sustainability Perspective:

The project aligns with sustainable goals as it promotes the responsible use of technology for secure digital data management. As a software solution, its environmental footprint is less compared to physical data storage methods.(Podder et al., 2020)

In addition, this project helps achieve global targets for smart and responsible technological growth as per sustainable development goal 9 (The Global Goals, 2022). It does this by providing a trustworthy, secure, and sustainable data management solution. This highlights how Information Technology can boost economic development and help keep our online communities secure in the digital age.

References

Lord, N. (2017). *Uncovering Password Habits: Are Users' Password Security Habits Improving? (Infographic)*. [online] Digital Guardian. Available at: <https://digitalguardian.com/blog/uncovering-password-habits-are-users-password-security-habits-improving-infographic>

nordpass.com. (n.d.). *Industry-Leading Features*. [online] Available at: <https://nordpass.com/features/>

Cloudflare (2022). What is Encryption? | Types of Encryption | Cloudflare. *Cloudflare*. [online] Available at: <https://www.cloudflare.com/learning/ssl/what-is-encryption/>

www.microfocus.com. (n.d.). *What is Encryption and How Does It Work? | Micro Focus*. [online] Available at: <https://www.microfocus.com/en-us/what-is/encryption>

Bitdefender. (n.d.). What is Data Encryption? [online] Available at: <https://www.bitdefender.com/cyberpedia/what-is-data-encryption/> [Accessed (04.2023)].

Harmening, J. (2017). *Symmetric Encryption - an overview | ScienceDirect Topics*. [online] www.sciencedirect.com. Available at: <https://www.sciencedirect.com/topics/computer-science/symmetric-encryption>

Sciencedirect.com. (2017). *Asymmetric Encryption - an overview | ScienceDirect Topics*. [online] Available at: <https://www.sciencedirect.com/topics/computer-science/asymmetric-encryption>

SSL2BUY (2019). *Symmetric vs. Asymmetric Encryption – What Are differences?* [online] SSL2BUY Wiki - Get Solution for SSL Certificate Queries. Available at: <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences>

Sönmez, O. and Paar, C. (2009). *Symmetric Key Management: Key Derivation and Key Wrap Chair for Communication Security*. [online] Available at: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b4114e8eafcc6a9114be5bba2401b6cb8ba2cd49> [Accessed 04.2023].

Aggarwal. (2021). *Key Derivation Function - an overview* | ScienceDirect Topics. [online] Available at: <https://www.sciencedirect.com/topics/computer-science/key-derivation-function> [Accessed 04.2023].

Arias, D. (2018). *Adding Salt to Hashing: A Better Way to Store Passwords*. [online] Auth0. Available at: <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>

IBM (n.d.). *What is a REST API?* | IBM. [online] www.ibm.com. Available at: <https://www.ibm.com/topics/rest-apis>

R. Fielding, M. Nottingham and J. Reschke (2022). *RFC 9110 HTTP Semantics*. [online] rfc-editor.org. Available at: <https://www.rfc-editor.org/rfc/rfc9110.html> [Accessed 04. 2023].

Rescorla, E. (2000). *RFC 2818, HTTP Over TLS*. [online] IETF. Available at: <https://datatracker.ietf.org/doc/html/rfc2818>

Oracle (2021). *What is a relational database?* [online] Oracle.com. Available at: <https://www.oracle.com/database/what-is-a-relational-database/>

Statista. (n.d.). *Most popular relational database management systems 2020*. [online] Available at: <https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/>.

Microsoft (2022). *Database normalization description - Office*. [online] learn.microsoft.com. Available at: <https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description>

developer.mozilla.org. (n.d.). *IndexedDB key characteristics and basic terminology - Web APIs | MDN*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Terminology [Accessed 05.2023].

Auth0 (n.d.). *Authentication vs. Authorization*. [online] Auth0 Docs. Available at: <https://auth0.com/docs/get-started/identity-fundamentals/authentication-and-authorization>

Globalknowledge.com. (2018). *The Three Types of Multi-Factor Authentication(MFA)*. [online] Available at: <https://www.globalknowledge.com/us-en/resources/resource-library/articles/the-three-types-of-multi-factor-authentication-mfa/>

VentureBeat. (2017). *A guide to common types of two-factor authentication*. [online] Available at: <https://venturebeat.com/security/a-guide-to-common-types-of-two-factor-authentication/>

What is Rapid Application Development? i. (n.d.). Available at: https://www.iro.umontreal.ca/~dift6803/Transparents/Chapitre1/Documents/rad_wp.pdf

www.lastpass.com. (n.d.) 1. *Transparent Security & Customer Data Protection - LastPass*. [online] Available at: <https://www.lastpass.com/security> [Accessed 05.2023].

www.lastpass.com. (n.d.) 2. *Our Zero-Knowledge Security Model | LastPass*. [online] Available at: <https://www.lastpass.com/security/zero-knowledge-security>

www.lastpass.com. (n.d.) 3. *Share Passwords Effortlessly | LastPass*. [online] Available at: <https://www.lastpass.com/features/password-sharing>

1Password. (n.d.) 1. *About the 1Password security model*. [online] Available at: <https://support.1password.com/1password-security/#encryption> [Accessed 05.2023].

Password Security Design 1Password Memberships. (n.d.) 2. Available at: <https://1passwordstatic.com/files/security/1password-white-paper.pdf>

developer.mozilla.org. (n.d.). *Web Crypto API - Web APIs | MDN*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API

Federal Information Processing Standards Publication 197 Announcing the
ADVANCED ENCRYPTION STANDARD (AES). (2001). Available at:
<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>

web.archive.org. (2021). *Wayback Machine*. [online] Available at:
<https://web.archive.org/web/20210617144558/https://people.csail.mit.edu/rivest/Rsapaper.pdf> [Accessed 05.2023].

Kjellemo, D.O. (2022). *Cyber security and cryptography*. [online] NTNU. Available at:
ntnu.no [Accessed May 2022].

PostgreSQL Documentation. (2023). *7.8. WITH Queries (Common Table Expressions)*.
[online] Available at: <https://www.postgresql.org/docs/current/queries-with.html>
[Accessed 05.2023].

vuejs.org. (n.d.). *Security | Vue.js*. [online] Available at: <https://vuejs.org/guide/best-practices/security.html>

Dashlane. (2019). *Never forget another password | Dashlane*. [online] Available at:
<https://www.dashlane.com/>

nordpass.com. (n.d.). *Password Manager: Autosave & Autofill Passwords*. [online]
Available at: <https://nordpass.com/>

Google Cloud. (n.d.). *Titan Security Key Fido U2 F Usb C Nfc Ble*. [online] Available at:
<https://cloud.google.com/titan-security-key>

vee-validate.logaretm.com. (n.d.). *VeeValidate*. [online] Available at: <https://vee-validate.logaretm.com/v3/> [Accessed 05.2023].

support.microsoft.com. (n.d.). *Create and use strong passwords*. [online] Available
at: [https://support.microsoft.com/en-us/windows/create-and-use-strong-
passwords-c5cebb49-8c53-4f5e-2bc4-
fe357ca048eb#:~:text=A%20strong%20password%20is%3A](https://support.microsoft.com/en-us/windows/create-and-use-strong-passwords-c5cebb49-8c53-4f5e-2bc4-fe357ca048eb#:~:text=A%20strong%20password%20is%3A)

Bach, E. (n.d.). *Discrete Logarithms and Factoring*. [online] Available at: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/1984/5973.html> [Accessed May 2023].

Cryptography Stack Exchange. (n.d.). *number theory - Would the ability to efficiently find Discrete Logs have any impact on the security of RSA?* [online] Available at: <https://crypto.stackexchange.com/questions/802/would-the-ability-to-efficiently-find-discrete-logs-have-any-impact-on-the-secur> [Accessed 22 Jun. 2022].

openwall.info. (n.d.). *Differences Between Fast Hashes and Slow Hashes [Openwall Community Wiki]*. [online] Available at: <https://openwall.info/wiki/john/essays/fast-and-slow-hashes>

Podder, S., Burden, A., Singh, S.K. and Maruca, R. (2020). *How Green Is Your Software?* [online] Harvard Business Review. Available at: <https://hbr.org/2020/09/how-green-is-your-software>

The Global Goals (2022). *Goal 9: Industry, innovation and infrastructure*. [online] The Global Goals. Available at: <https://www.globalgoals.org/goals/9-industry-innovation-and-infrastructure/>

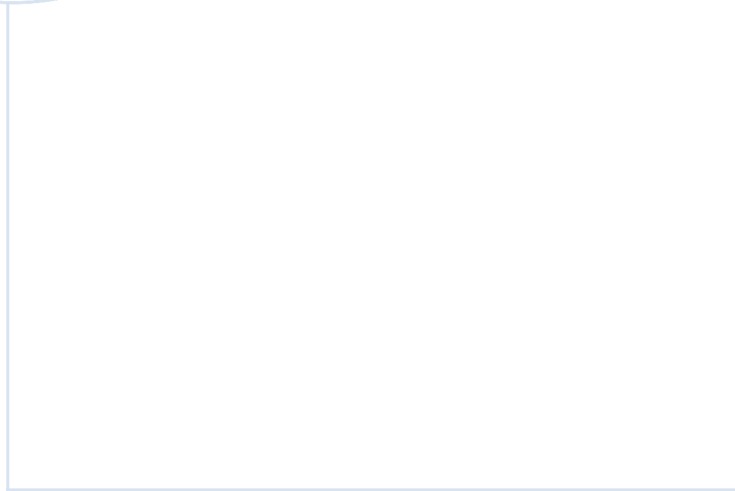
Appendices

Appendix A – Vision Document

Appendix B – System Document

Appendix C – Requirements Documentation

Appendix D – Project Handbook



 **NTNU**

Norwegian University of
Science and Technology