Roshan Upreti

Secure Sharing of Files

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master’s Thesis

19th March 2024

Abstract

Author(s):	Roshan Upreti
Title:	Secure Sharing of Files
Number of Pages:	38 pages
Date:	19 March 2024
Degree:	Master of Engineering
Degree Programme:	Information Technology
Specialisation option:	Networking and Services
Instructor(s):	Ville Jääskeläinen, Principal Lecturer

The thesis explores the development and analysis of a secure file-sharing system within Enterprise Resource Planning (ERP) systems, focusing on Two-Factor authentication (2FA) to enhance security. It addresses the need for secure data exchange in business environments, especially in sensitive Business to Business (B2B) contexts.

A proof of concept was designed, implemented and evaluated. The created system demonstrated that it effectively ensures data integrity and confidentiality. The research highlights the balance between security measures and user accessibility and provides some useful insights regarding information technology, cybersecurity, and data protection.

Keywords: Secure File Sharing, ERP Systems, Two-Factor Authentication, Data Security, Information Technology, Cybersecurity, Data Protection.

Contents

List of Abbreviations

2FA	Two-factor authentication.
API	Application Programming Interface.
B2B	Business-to-Business.
CCPA	California Consumer Privacy Act of 2018.
CRUD	Create, Read, Update and Delete.
DC2.0	DomaCare 2.0.
eID	Electronic Identification.
eIDAS	Electronic Identification and Trust Services.
email	Electronic Mail.
ERP	Enterprise Resource Planning.
EU	European Union.
FINEID	Finnish Electronic Identity Card.
FTN	Finnish Trust Network.
GDPR	General Data Protection Regulation.
HTTP	Hypertext Transfer Protocol.
ID	Identification.
JAR	Java ARchive.
jOOQ	Java Object Oriented Querying.
JPA	Jakarta Persistence API.
JWT	JSON Web Token.
LAN	Local Area Network.
MFA	Multi-factor Authentication.
ORM	Object-relational mapping.
PIN	Personal Identification Number.
PoC	Proof of Concept.
RAI	Resident Assessment Instrument.
REST	Representational state transfer.
SMS	Short Message Service.
SQL	Structured Query Language.
SSN	Social Security Number.
TRAFICOM	Liikenne-ja Viestintävirasto.
TUPAS	Tunnistuspalvelu Standardi.

UI	User Interface.
USB	Universal Serial Bus.

1 Introduction

Enterprise Resource Planning (ERP) systems, in the simplest of terms, can be defined as business process management software used to manage various aspects of a company’s day-to-day operations, including finances, supply chain, human resources, logistics, manufacturing, and customer relationship management. An ERP system comprises an integrated set of various software applications and tools that bring together information from all the business units into a single platform, thus rendering a centralized source of information for a business or an enterprise. ERP systems, through their data-driven operations, produce quantifiable data, which is crucial for a business. For example, they enable running financial analyses and predicting future stock levels to maintain a healthy inventory.

ERP systems are now considered the price of entry for running a business and offer a variety of deployment options, such as cloud-based, on-premises or hybrid. Despite having the word ‘Enterprise’ in its name, ERP systems serve companies of all sizes: startups, small, mid-sized and large. ERP systems, by unifying several units of a business, promote a free flow of information sharing which can be beneficial to a business in terms of cost-saving, planning, productivity, internal communications, and customer service. Examples of popular ERP systems include Oracle ERP cloud, SAP, and BizAutomation.

Invian Oy is a leading provider of ERP solutions in Finland in the social and care sector. The company commenced operations in 2005 in Oulu and is based in its Helsinki and Oulu offices. DomaCare is a popular ERP system used by many companies associated with the social and care services in the Business-to-Business (B2B) context. Services include secure customer data management, booking and billing services, RAI performance assessments as part of the system, a range of useful statistics, and reports among many others, and are available as desktop, web and mobile applications. Figure 1 shows DomaCare as a web app and a desktop app, side by side.

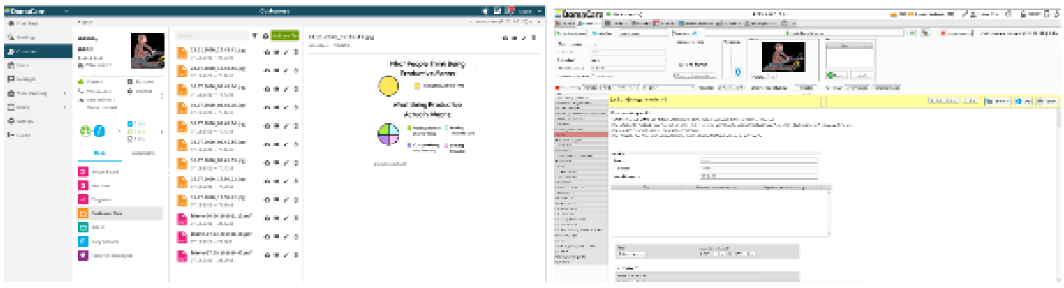


Figure 1: DomaCare web and desktop app.

As the clients of DomaCare are other B2B providers, management of their customer data is one of its core features, and one of the recurring scenarios in this context is the need to share this sensitive customer data with outside agencies and entities (both public and private) for various reasons, including adherence with existing laws. For example, in the event of a death, a nursing home may be required to send the deceased’s paperwork to local municipalities or other government agencies. The current practices of sharing this highly sensitive information include using mail, e-mail attachments, physically carrying it to the recipient in a memory stick and more. To make this sharing process seamless, secure and most importantly centralized, Invian wants to implement secure file sharing as a feature in the existing ERP system, as a solution to the business challenge.

The objective here is to ensure that the information is promptly delivered to the right person in the right organization by validating the recipient's identification through some sort of authorization process—2FA, authorizing via bank credentials, for example. Various processes and stages involved in the design and implementation of this so-called feature will be discussed in detail, as the thesis progresses.

This thesis aims to serve as a research paper focused on the design and implementation of a software solution that enables users to securely share files with third-party entities using a secure link. This implementation is intended to serve as one of many functionalities in an existing ERP system, DomaCare, a product of Invian Oy.

The thesis comprises six chapters: Chapter One provides a brief introduction to the context and the company. Chapter Two focuses on project specifications. Chapter Three delves into the technical background, covering research methodologies and existing mechanisms. Chapter Four explains the solution in terms of Proof of Concept (PoC) design and implementation. Chapter Five provides an overview of the solution's analysis, whereas the final chapter, Chapter Six, comprises the conclusion.

2 Project Specifications

File sharing is an integral part of modern business endeavours, with no exceptions in the context of ERP systems dealing with a huge number of various kinds of data daily. These files can range from simple invoices to employees and customers information, necessitating treatment based on the sensitivity of the contained information. Therefore, file sharing should be handled with caution, based on the sensitivity of the information they carry. For instance, it might be acceptable to share low-sensitivity files, such as general business invoices and simple presentations, over Local Area Network (LAN), e-mail, or Universal Serial Bus (USB) storage devices. However, this may not be the case for files containing highly sensitive information intended for specific recipients outside the organization. This section outlines the requirements for a secure file sharing system which should enable users to securely upload, store and share files with other authorized users, while ensuring the confidentiality and integrity of the files. A brief overview of the current state analysis, along with a synopsis of the requirements in terms of functional and non-functional types, will be discussed as follows.

2.1 Current State Analysis

Figure 2 represents a simplified version of the system diagram for 'DomaCare,' divided into four sections that encompass components and usability. The first section depicts various clients accessing the software systems via different platforms: Mobile, Desktop, and Web, as illustrated in the second section. A dedicated mobile server handles all requests sent by mobile clients. The legacy Java desktop application, still widely and actively used by many customers, and the web client requests are managed by the REST server, with the frontend implemented as a React App running on an app server. All these platforms utilize common database servers for persisting and retrieving data, as seen in the third section. Google Cloud Storage, used for the storage of files and other objects, is shown in the fourth section.

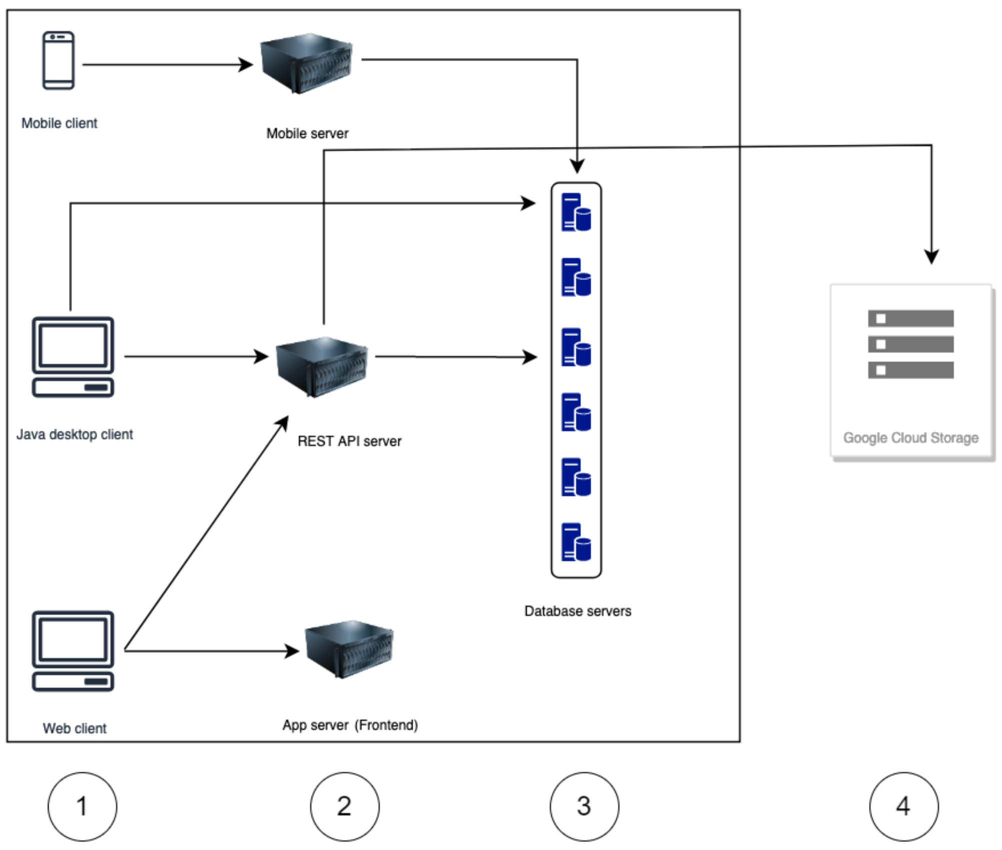


Figure 2: DomaCare System Overview.

2.2 Functional Requirements

The functional requirements can be discussed under the following four topics.

- **User Authentication and Authorization**: As with any web-based software system, user authentication and authorization are key requirements. The system must authenticate and authorize users before granting access to the system. Users should be able to sign up for an account and create their own credentials, and the system should allow for Multi-factor Authentication (MFA) to improve security. Administrators should be able to manage user accounts, including assigning roles and managing permissions.

- **File Upload and Storage**: The system should allow users to upload files to the system and store them in a secure and encrypted manner. The system should also have the capability to scan the uploaded files for possible viruses and malware, and support a wide range of file types and sizes.

- **File Sharing**: Users should be able to share files with other authorized users. The system should allow users to set access controls and permissions for each file they share, along with a mechanism for sharing them securely, such as through password-protected links or encrypted Electronic Mail (email) notifications, while tracking all the file sharing activities such as the sender, recipient and the date.

- **File Access and Retrieval**: Authorized users should be able to access and download the files shared with them, along with features like previewing the file, version control for files, which allows access to previous versions if needed. The system should have a mechanism for notifying users when the shared files are accessed or downloaded.

2.3 Non-Functional Requirements

The Non-Functional Requirements can be discussed under the following three topics.

- **Security**: The system should employ industry-standard encryption protocols to protect the confidentiality and integrity of the files, with the provision of robust security controls and measures to prevent unauthorized access and ensure data privacy. The system should also comply with relevant data privacy and protection regulations such as General Data Protection Regulation (GDPR) and/or California Consumer Privacy Act of 2018 (CCPA).

- **Scalability and Performance**: The system should be able to handle many users and files without compromising performance, with high availability and fault tolerance to ensure that it is always accessible.

- **User Experience**: The system should have an intuitive and user-friendly interface with easy navigation. The system should have a fast response time and minimal latency to ensure a smooth user experience, in addition to a good support and assistance via a help desk or knowledge base.

In the context of Invian, majority of the requirements are implemented in its product, DomaCare 2.0 (DC2.0), a browser based modern web-application that provides various features, examples include booking, payroll systems, Resident Assessment Instrument (RAI), calendar system etc., as a part of an ERP system, via an intuitive User Interface (UI). These features vary upon the nature of services they offer, and are available as Representational state transfer (REST) Application Programming Interface (API). RESTful API is a popular architectural style that uses Hypertext Transfer Protocol (HTTP) requests to access and use data. It was defined by Roy Fielding in his 2000 doctoral dissertation, and has almost become the de-facto industry standard in the present times. DC2.0 utilizes JSON Web Token (JWT), a token based stateless authentication mechanism to authenticate its users. The term stateless signifies the fact that the REST server doesn’t have to rely on or store any client related information, and all the required information must be sent to the server on every request. JWTs can be encrypted, encoded, signed and set to expire after a certain period.

3 Technical Background

This chapter provides a comprehensive overview of the existing technologies in Finland for authenticating identity online. The bedrock of digital security, especially in the context of a technologically advanced society like Finland, is reliable online authentication. The act of confirming one’s identity online is foundational to maintaining the integrity and confidentiality of digital interactions, whether for accessing financial services, government portals, or personal communication. Finland’s digital landscape has been profoundly shaped by its approach to Electronic Identification (eID) verification. The Finnish government introduced the Finnish Electronic Identity Card (FINEID) in 1999, marking the country’s initial steps towards a digital identification system. The FINEID card was a multi-functional smart card that provided a physical identity document with digital authentication and signing capabilities. It was designed to be a secure and versatile tool for citizens to access various e-services offered by the government.

In parallel with the government’s efforts, Finnish banks developed the Tunnistuspalvelu Standardi (TUPAS) protocol, a bank-based identification system that quickly became the standard for online services. TUPAS allowed individuals to use their bank credentials to authenticate their identity across various platforms, from government services to private sector transactions. Its widespread adoption was driven by the existing trust in the banking system and the convenience it offered to users (1). TUPAS was instrumental in the digitalization of Finnish society, providing a unified standard for strong customer authentication. However, as the digital landscape evolved, so did the regulatory environment. The introduction of the European Union (EU)’s Electronic Identification and Trust Services (eIDAS) regulation brought new requirements for interoperability and security, which TUPAS, in its original form, was not designed to meet. These regulatory changes necessitated a revaluation of the TUPAS protocol and its compliance with the new standards (1). This is where Liikenne- ja Viestintävirasto (TRAFICOM), the Finnish Transport and Communications Agency, played a crucial role. As the regulatory authority overseeing electronic communications in Finland, TRAFICOM was instrumental in ensuring that the national digital identification systems adapted to meet the new EU standards. Their responsibilities included setting and enforcing standards for electronic identification and digital security, certifying and approving eID technologies, and promoting secure eID practices.

Moreover, TRAFICOM’s role extended beyond regulatory compliance. The agency actively collaborated with other government bodies and private sector entities to develop and implement eID systems. This included supporting innovation and development in the field of electronic identification and ensuring data privacy and security in eID systems. TRAFICOM also played a key role in consumer protection and education, informing the public about the safe use of electronic identification and protecting consumer rights in the digital domain. In response to the eIDAS regulation, TRAFICOM guided the establishment of the Finnish Trust Network (FTN), a framework designed to standardize and secure eID services in Finland. This initiative marked a transition from TUPAS to a more regulated and advanced digital identification system. TRAFICOM’s involvement was critical in ensuring compliance with both national and EU regulations, thereby maintaining the security and reliability of digital services in Finland (2).

3.1 Two-Factor Authentication(2FA)

2FA is a critical security process in which users are required to provide two distinct authentication factors to verify their identity. This approach is a key element in layered defence strategies, significantly increasing the difficulty for unauthorized entities to access targets such as physical locations, computing devices, networks, or databases. Essentially, 2FA introduces an additional layer of security to the authentication process, bolstering its defence against a variety of cyber threats (3). The two factors in 2FA typically involve:

1. **Something the User Knows (Knowledge Factor)**: This could be a password, a Personal Identification Number (PIN), or answers to secret questions. It represents a unique piece of information known to the user.

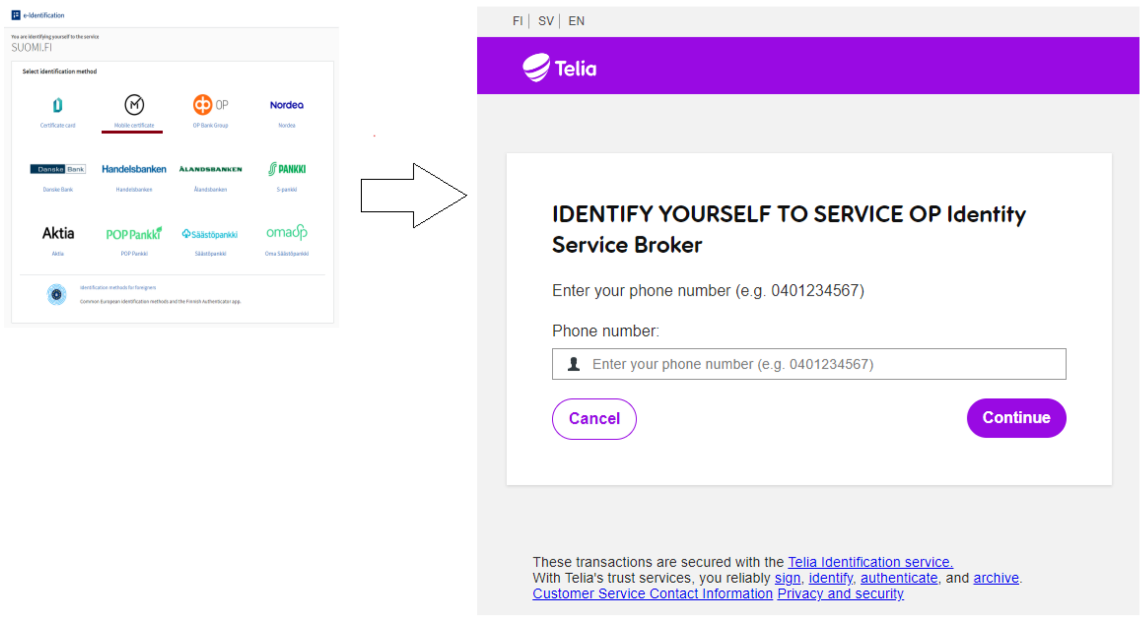
2. **Something the User Has (Possession Factor)**: This factor usually involves an object the user physically possesses, such as a mobile phone or a security token.

By integrating these two factors, 2FA significantly reduces the likelihood of unauthorized access to files. Even if the knowledge factor is compromised, the absence of the possession factor ensures the continued security of the data (4).

Following are some popular methods of electronic identity verification in Finland.

3.2 Mobile ID

Mobile Identification (ID) in Finland is a sophisticated eID solution that utilizes a secure application on a PKI-enabled SIM card. This application stores the user’s digital credentials, which are used for authentication purposes. When a user attempts to access a service, they are prompted to enter a PIN code on their mobile device. This action, combined with the possession of the device itself, constitutes a two-factor authentication process, ensuring a high level of security. The system’s design prioritizes user privacy by allowing for attribute queries without revealing unnecessary personal information (5). Figure 3 represents the interface for verifying identity using the mobile certificate, via suomi.fi.


e-Identification

You are identifying yourself to the service

SUOMI.FI

Select identification method

Certificate card | Mobile certificate | OP Bank Group | Nordea

Danske Bank | Handelsbanken | Ålandsbanken | S-pankki

Aktia | POP Pankki | Säästöpankki | Oma Säästöpankki

Identification methods for foreigners

Common European identification methods and the Finnish Authenticator app.

FI | SV | EN

Telia

IDENTIFY YOURSELF TO SERVICE OP Identity Service Broker

Enter your phone number (e.g. 0401234567)

Phone number:

Enter your phone number (e.g. 0401234567)

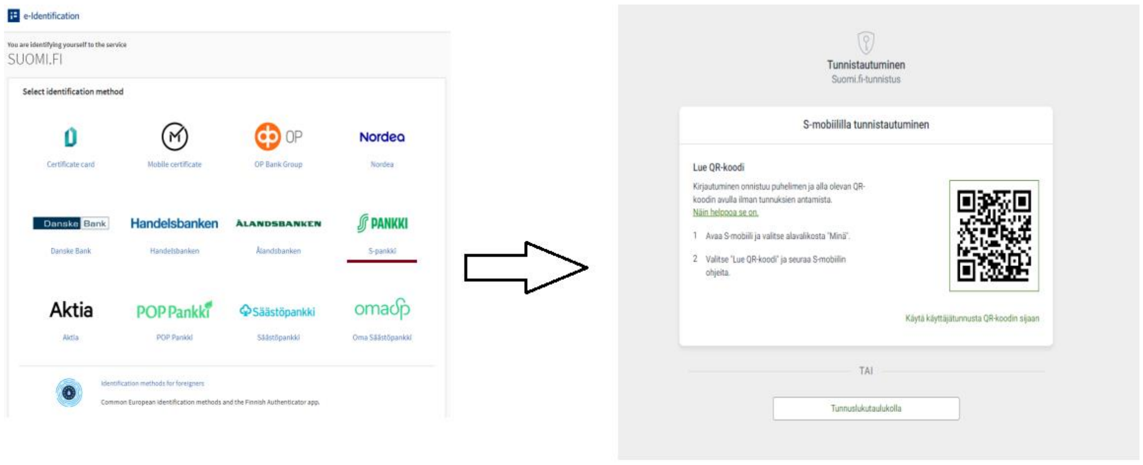
Cancel | Continue

These transactions are secured with the Telia Identification service.
With Telia's trust services, you reliably sign, identify, authenticate, and archive.
Customer Service Contact Information Privacy and security

Figure 3: Suomi.fi mobile certificate verification interface (mobile certificate option selected)

3.3 Bank ID

Bank ID is an integral part of the Finnish eID ecosystem, particularly for financial services and government portals. It uses online banking credentials to verify identity, capitalizing on the inherent trust in the banking system. Despite its widespread adoption, the process of obtaining Bank ID credentials has been criticized for potential barriers to access and concerns over data privacy (6). To use a mobile bank ID, one must first have a bank account and a mobile bank ID with a Finnish bank. A code can then be used to activate the mobile bank ID on the user's smartphone. Mobile bank IDs are very secure. They use strong encryption to protect personal data, and they are regularly audited by independent security experts. Figure 4 represents the interface for verifying identity using the bank credentials, via suomi.fi.


e-Identification

You are identifying yourself to the service

SUOMI.FI

Select identification method

Certificate card · Mobile certificate · OP Bank Group · Nordea

Danske Bank · Handelsbanken · Ålandsbanken · S-pankki

Aktia · POP Pankki · Säästöpankki · Oma Säästöpankki

Identification methods for foreigners

Common European identification methods and the Finnish Authenticator app.

Tunnistautuminen

Suomi.fi-tunnistus

S-mobiililla tunnistautuminen

Lue QR-koodi

Kirjautuminen onnistuu puhelimen ja alla olevan QR-koodin avulla ilman tunnuksien antamista.

Näin helppoa se on.

1 Avaa S-mobiili ja valitse alavalikosta "Minä".

2 Valitse "Lue QR-koodi" ja seuraa S-mobiilin ohjeita.

Käytä käyttäjätunnusta QR-koodin sijaan

TAI

Tunnuslukutaulukolla

Figure 4: suomi.fi bank id verification interface (s-pankki selected for reference)

3.4 Suomi.fi

Suomi.fi is the official digital service platform for citizens, providing a gateway to various government services. It acts as an eID broker, allowing users to authenticate their identity using various methods, including Mobile ID and Bank ID. Suomi.fi is part of the Finnish Trust Network and is compliant with eIDAS regulations, ensuring interoperability across EU member states. The platform streamlines the process of accessing government services, tax information, and more, by providing a single digital access point for citizens and residents in Finland (7). Figure 5 represents the identification interface on suomi.fi, showcasing various authentication methods.

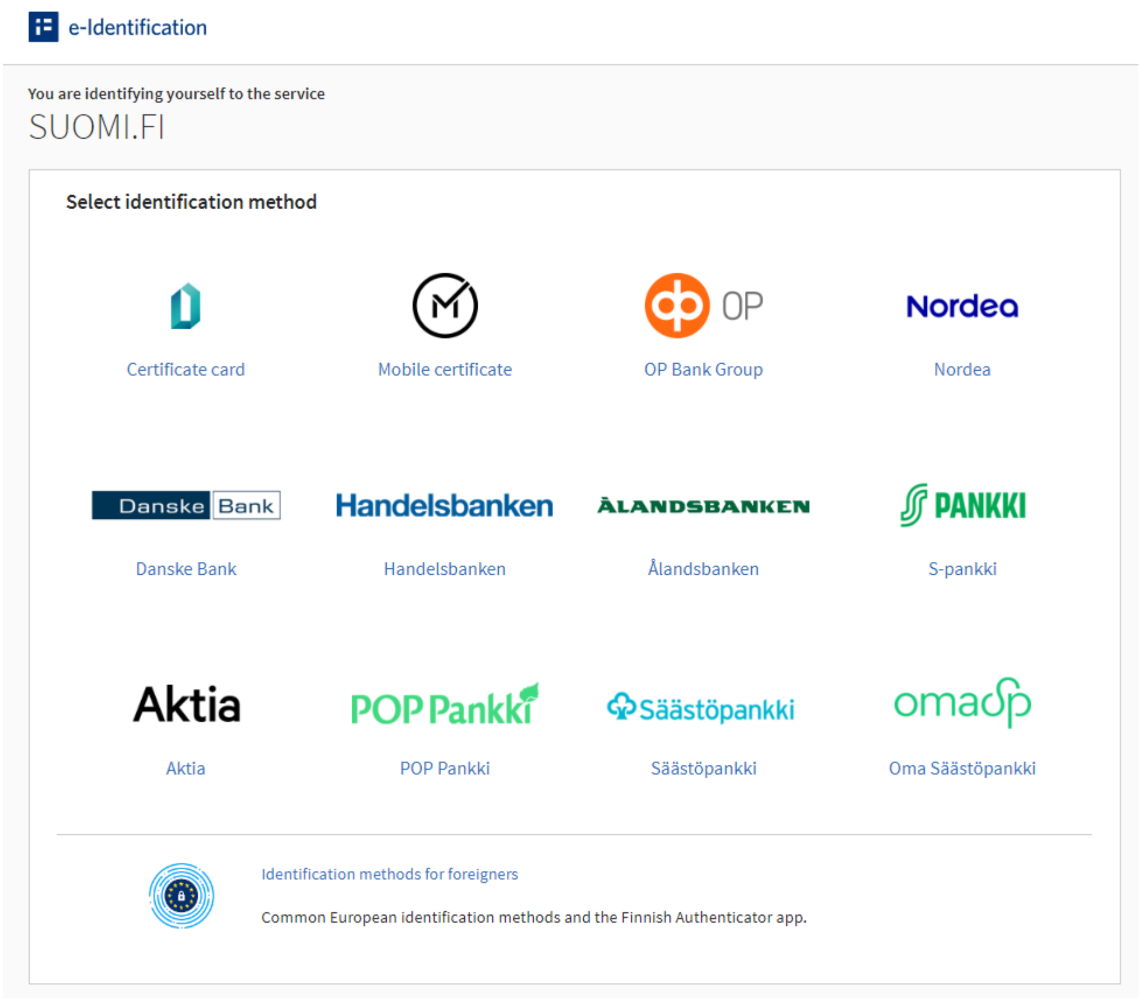

e-Identification
You are identifying yourself to the service
SUOMI.FI
Select identification method
Certificate card
Mobile certificate
OP Bank Group
Nordea
Danske Bank
Handelsbanken
Ålandsbanken
S-pankki
Aktia
POP Pankki
Säästöpankki
Oma Säästöpankki
Identification methods for foreigners
Common European identification methods and the Finnish Authenticator app.

Figure 5: suomi.fi identification interface.

In today's digital landscape, the security of data transmission and file sharing stands as a crucial concern. Traditional identity verification methods, such as Bank ID or mobile certification, often entail complex prerequisites and involve the handling of sensitive personal information. For example, sending a secure email might require the sender to know the recipient's Social Security Number (SSN), a highly sensitive piece of information typically accessible only through specific government permissions and bureaucratic processes (8). To address these challenges, this thesis proposes a secure file sharing system that leverages two-factor authentication (2FA) for enhanced security, particularly suitable for scenarios where a user needs to send a download link via email to a third party.

3.5 Proposed Secure File Sharing System

Figure 6 represents the proposed system, in terms of workflow.

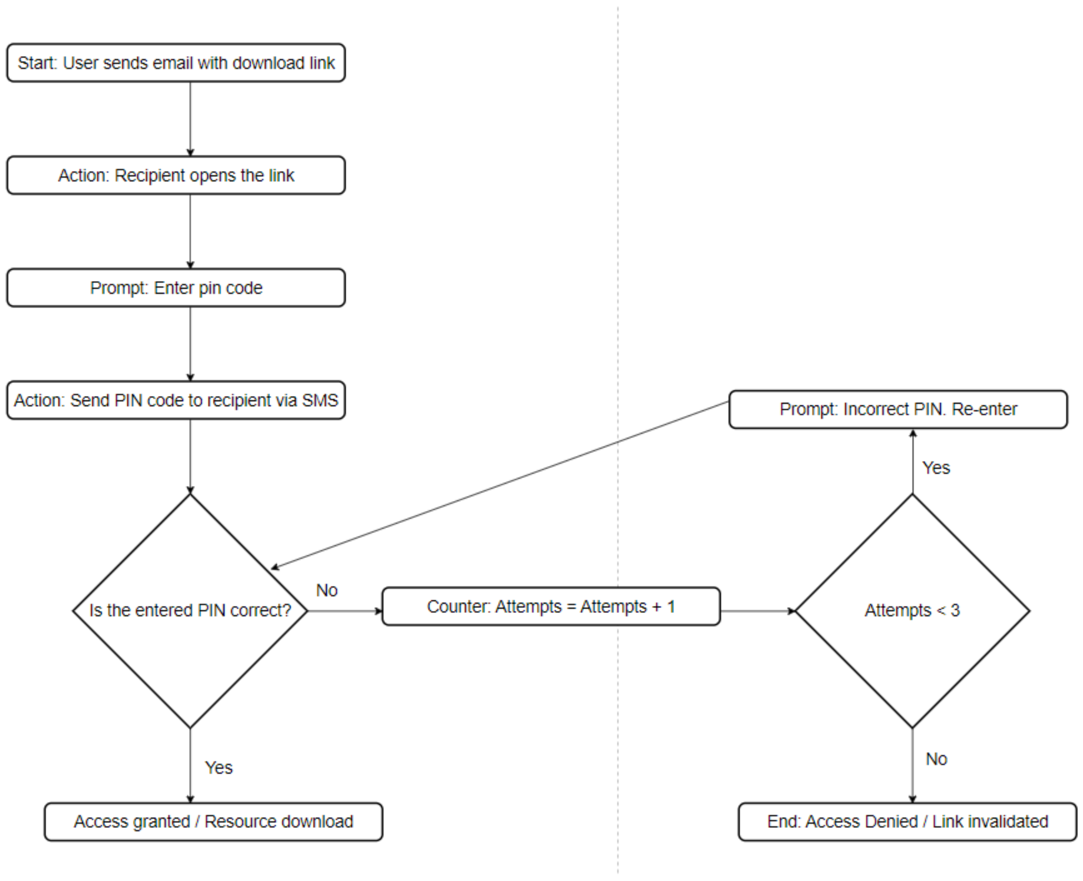

Start: User sends email with download link
Action: Recipient opens the link
Prompt: Enter pin code
Action: Send PIN code to recipient via SMS
Is the entered PIN correct?
No
Yes
Access granted / Resource download
Counter: Attempts = Attempts + 1
Attempts < 3
Yes
Prompt: Incorrect PIN. Re-enter
No
End: Access Denied / Link invalidated

Figure 6: Proposed workflow.

The workflow involves the following steps:

1. A user sends an email with a download link.

2. The recipient opens the link.

3. The recipient is prompted to enter a PIN code.

4. A PIN code is sent to the recipient via SMS.

5. The recipient enters the PIN.

6. If the correct PIN is entered, access to the download link is granted, and the resource is downloaded.

7. If an incorrect PIN is entered, access is denied, and the process loops back to the PIN prompt, providing a maximum of three attempts, and invalidating the link given an incorrect PIN is provided in the last attempt.

In the proposed secure file sharing system, the process begins when a user sends an email containing a download link, initiating the first phase of the authentication process where the email and link serve as the initial access point (8). Upon clicking the download link in the received email, the recipient triggers the second phase of authentication. At this point, the system prompts the recipient to enter a PIN code, marking a critical step that introduces the second factor in the 2FA process and thereby enhances the security of the transaction (3). Concurrently, a one-time PIN code is generated and sent to the recipient’s mobile phone via SMS, adhering to the 'something the user has' principle of 2FA, and effectively using the recipient’s mobile phone as a physical token of authentication (4). The recipient then enters this PIN into the system. If the PIN is correct, access to the download link is granted, allowing the recipient to download the resource and ensuring that the file is accessed solely by the intended recipient. Conversely, if an incorrect PIN is entered, the system denies access and re-prompts for the PIN, a vital security measure to prevent unauthorized access. This ensures that only an individual with access to both the link (knowledge factor) and the correct PIN (possession factor) can access the file (4).

In conclusion, the implementation of 2FA in the proposed secure file sharing system offers a robust solution to digital data security challenges. Although it introduces an additional step to the file sharing process, the enhanced security it provides justifies its consideration in light of increasing cyber threats.

4 Design and Implementation

Transitioning from the theoretical underpinnings and design principles delineated in the initial chapters, this chapter delineates the crucial phase of translating these concepts into a tangible reality through the implementation of a PoC for secure file sharing. This chapter represents a pivotal shift from abstract ideas to their practical execution, highlighting the process of bringing to life a standalone secure file sharing system. The focus here diverges from integrating with specific ERP products like DomaCare to evaluating the PoC as an independent entity, capable of demonstrating the effectiveness and robustness of the security measures, especially the 2FA, in a controlled environment.

The journey undertaken in this chapter is essential for substantiating the theoretical models and security protocols elaborated upon in Chapter 3 through a real-world application, albeit in a standalone manner. This approach allows for a focused evaluation of the PoC's design, development, and security features without the complexities of integration into existing systems. It is a strategic decision aimed at isolating the PoC's performance and security attributes, ensuring a thorough assessment of its capabilities and limitations.

In this narrative shift, the chapter transitions from discussing "what" needs to be accomplished to "how" it is achieved, emphasizing the step-by-step process of developing, deploying, and critically evaluating the PoC. This standalone assessment strategy is not only pivotal for validating the proposed secure file sharing model but also serves as a crucial step towards understanding the practical challenges and opportunities in enhancing data security. Through a detailed exploration of implementation challenges, innovative solutions, and the PoC's adherence to security best practices, this chapter aims to contribute significant insights into the field of secure file sharing, independent of specific ERP integrations.

4.1 PoC Design

This PoC for secure file sharing leverages the strengths of Spring Boot, Hibernate, and Java Object Oriented Querying (jOOQ) to create a robust and efficient application. Spring Boot facilitates rapid application development with its convention-over-configuration approach, while Hibernate and jOOQ work in tandem to manage data access and manipulation, combining the ease of Object-relational mapping (ORM) with the precision and control of Structured Query Language (SQL).

Spring Boot is a powerful framework for building Java applications quickly and easily. It simplifies the development process by providing default configurations for building Spring-powered applications. With Spring Boot, developers can create stand-alone, production-grade applications smoothly, without the need for extensive configuration. It supports a range of Spring projects with features that include embedded servers, security, metrics, and health checks, making it easier to develop web applications, microservices, and more. Spring Boot’s philosophy of convention over configuration and its ability to automatically configure Spring and third-party libraries make it an ideal choice for developers looking to efficiently develop and deploy applications. What follows is a set of key components of the proposed web application.

- **Application Layer**: Powered by Spring Boot, this layer establishes the application infrastructure, including embedded web servers and RESTful endpoints for file sharing functionalities. Spring Boot’s auto-configuration capabilities streamline the setup and development of the application.

- **Model Layer**: Utilizing Hibernate for Object-Relational Mapping (ORM), this layer defines the entities involved in file sharing, such as users and files. jOOQ complements Hibernate by enabling type-safe SQL operations for executing complex queries, thereby enhancing data access and manipulation capabilities.

- **Repository Layer**: Integrating Spring Data Jakarta Persistence API (JPA) and jOOQ, this layer offers a comprehensive data access strategy. Spring Data JPA facilitates basic Create, Read, Update, and Delete (CRUD) operations through repositories, simplifying the implementation of the data access layer. For more complex database interactions and queries, jOOQ provides detailed control and efficiency, ensuring robust data handling.

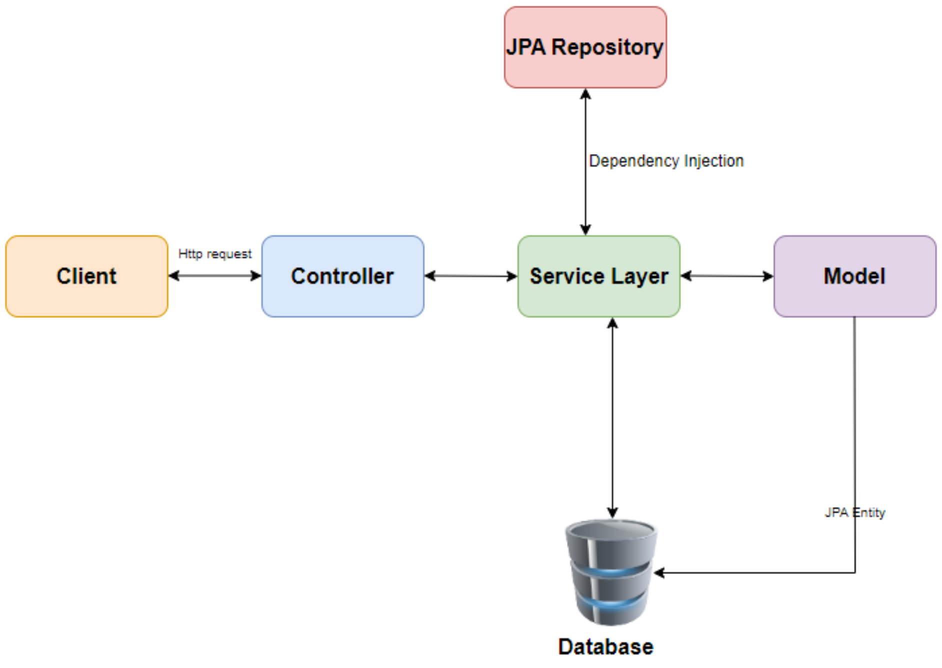


Figure 7: Architectural Diagram visualizing the flow and structure of the application.

Figure 7 represents a simplified architectural depiction of general Spring Boot applications (9), followed by a brief outline of the components:

- **Client**: Represents the users interacting with the system through various front-end interfaces.

- **Controller**: Manages HTTP requests, directing the flow of data between the client and service layer, and ensuring the appropriate response is returned to the user.

- **Service Layer**: Encapsulates the business logic of the application, handling the orchestration of data transfer between the controller and repository layers.

- **Model**: Defines the domain model, which directly maps to the database entities, ensuring data integrity and providing a structured data model for the application.

- **Repository (JPA Repositories)**: Interfaces with the database, abstracting the underlying data persistence mechanism and providing a clean separation of concerns.

- **Database**: The underlying storage mechanism for the system.

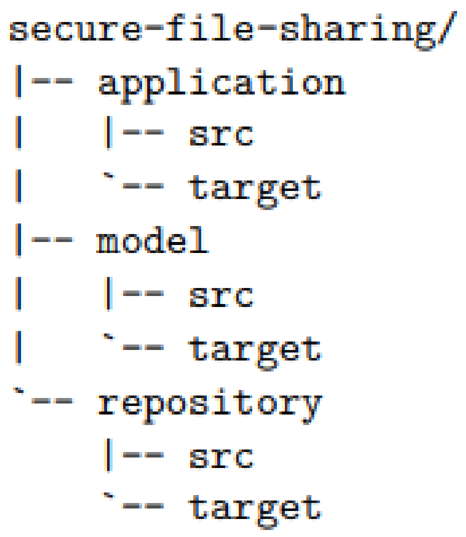
Adhering to the architectural paradigm, in the context of the Proof of Concept (PoC), the application module implements the controller and service layers, whereas the model and repository reside in their own respective modules. The database layer, in the context of the PoC, is managed by Hibernate and jOOQ to facilitate Object-Relational Mapping (ORM).

4.2 Implementation

In the implementation of the PoC, the Spring Boot framework is leveraged to streamline the creation and management of web applications. Offering an extensive infrastructure, Spring Boot supports both development and deployment phases, ensuring a seamless workflow. This section aims to delve deeper into the specific implementation facets of the application, focusing on the entry point, controller layer, and service layer functionalities. These components are elucidated under the following topics.

4.2.1 Structure

The application adopts a multi-module Spring Boot architecture, with each component implemented as a distinct module. The core structure of the application, named 'secure-file-sharing', is depicted below.



```
secure-file-sharing/
|-- application
|   |-- src
|   `-- target
|-- model
|   |-- src
|   `-- target
`-- repository
    |-- src
    `-- target
```

Figure 8: Tree structure representing the project core.

Figure 8 illustrates the organized separation of the application into modules, facilitating modular development and maintenance. The application module houses the controller and service components, while the model and repository modules contain the entities and the data access layers, respectively.

4.2.2 Entry Point

The entry point of the application is facilitated by Spring Boot's `@SpringBootApplication` annotation, which encapsulates component scanning, auto-configuration, and the setup of an embedded server. This streamlined approach to initiating a Spring Boot application is exemplified in the main class.

```
@SpringBootApplication
@EnableJpaRepositories(basePackages = "org.example.jparepository")
public class SecureShareApplication {
    public static void main(String[] args) {
        SpringApplication.run(SecureShareApplication.class, args);
    }
}
```

Figure 9: Application entry point.

4.2.3 Controller Layer

The controller layer plays a pivotal role in processing HTTP requests and steering the application's flow. Key controllers such as `HomeController` and `SharedLinkController` are designed to fulfil specific functions:

- **HomeController**: This controller facilitates navigation to the application's home page, providing a user-friendly interface for accessing the primary features of the application.

```
@Controller
public class HomeController {

    @GetMapping("/")
    public String home(Model model) {
model.attribute("title", "Link Request Form");
        return "link-request";
    }
}
```

Figure 10: Home controller renders the link request form.

- **SharedLinkController**: Dedicated to the creation and management of shared links, this controller is integral to enabling secure file sharing amongst users.

```
@RestController
@RequestMapping("api/v1/share")
@RequiredArgsConstructor
@Slf4j
public class SharedLinkController {

    private final ISharedLinkService sharedLinkService;

    @PostMapping("/new")
    public ResponseEntity<?> shareNewLink(/* parameters */) {
        // Implementation details
    }
}
```

Figure 11: SharedLinkController is responsible for the shared-link management, via calls to service layer.

4.2.4 Service Layer

The service layer, as represented by `SharedLinkService`, encapsulates the business logic essential for the management of shared links. Using Spring's `@Service` annotation and the `@RequiredArgsConstructor` annotation from Project Lombok, this layer ensures a clear delineation of responsibilities, thereby enhancing the application's scalability and maintainability.

```
@Service
@RequiredArgsConstructor
public class SharedLinkService implements ISharedLinkService {
    // Service methods implementation
}
```

Figure 12: Interface service responsible for shared link related operations.

4.2.5 Dependency Injection

The Spring Framework's dependency injection mechanism, particularly through constructor injection enabled by Project Lombok's @RequiredArgsConstructor, is instrumental in interlinking the application's layers. This method improves code readability, simplifies testing procedures, and aligns with clean architecture principles. Project Lombok is a Java library that automatically plugs into editors and build tools, spicing up the Java language with a series of annotations to reduce boilerplate code, such as getters, setters, and constructors, thereby streamlining the development process. Constructor injection is a technique where the required dependencies are provided to a class at the time of its construction, eliminating the need for setter methods or property injection, and fostering immutable object creation, which is conducive to cleaner and more robust code.

Through this formal exposition, the section provides a comprehensive overview of the application's design and the interplay between its various components, shedding light on the underlying technologies and methodologies that contribute to its efficiency and security.

4.3 Database Design

The application's database is structured to support the core features of secure file sharing, encompassing share logs, pin logs, and access logs. This design aims to ensure data integrity, facilitate efficient data retrieval, and support the application's security requirements:

- **share_log**: Central to the application, this table records details of each shared file, including recipient information, hashed links, sender details, and the share's validity. It acts as the primary entity, linking to other tables through relational integrity.

- **pin_log**: Tied to the share log through a foreign key, this table stores hashed pins and client fingerprints, along with pin expiration timestamps. This design enforces access control, ensuring that shared files are accessed securely and within designated time frames.

- **access_log**: Logs each successful access, capturing the timestamp and client fingerprint. This table is crucial for monitoring and auditing file access, thereby enhancing the application's security posture.

An entity-relationship diagram visually representing these relationships aids in understanding the database’s structure, focusing on the one-to-many relations from share log to pin log and access log, delineated by foreign key constraints, as seen in the figure below.

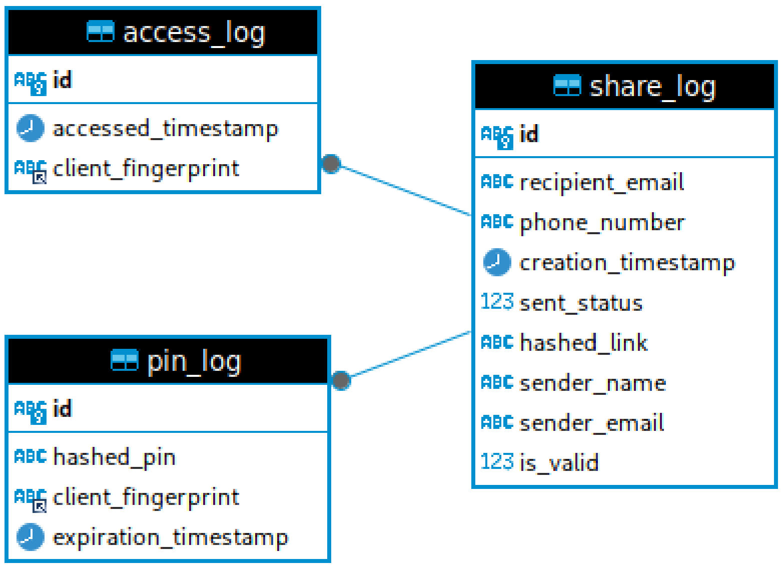
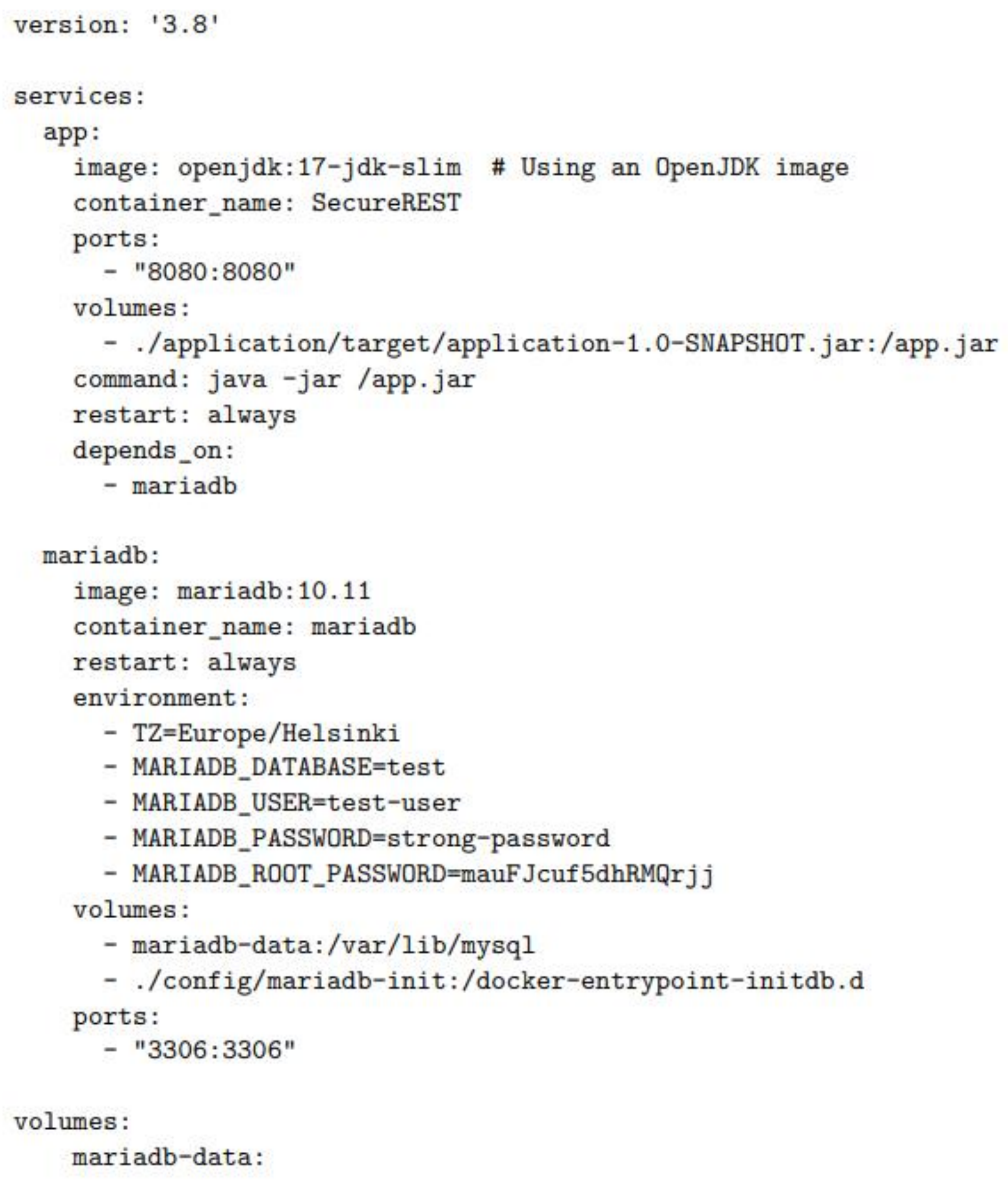


Figure 13: Diagram visualizing entity relationship among the tables used.

4.4 Containerized Local Deployment with Docker Compose

Upon a successful build, running the secure file-sharing application can be as straightforward as executing the compiled Java ARchive (JAR) file, provided an accessible database with the required schema and other essential prerequisites are in place. The management of these prerequisites is greatly simplified through Docker, which packages applications and their dependencies into containers. These lightweight, executable packages include everything needed to run an application: code, runtime, system tools, system libraries, and settings. By isolating the application from its surroundings, containerized software ensures consistent operation across various environments. This addresses the common challenge of applications working in one environment but failing in another, such as moving from development to production, thereby enabling predictable and scalable deployments.

Provided below is the project’s Docker compose configuration snippet.



```
version: '3.8'

services:
  app:
    image: openjdk:17-jdk-slim  # Using an OpenJDK image
    container_name: SecureREST
    ports:
      - "8080:8080"
    volumes:
      - ./application/target/application-1.0-SNAPSHOT.jar:/app.jar
    command: java -jar /app.jar
    restart: always
    depends_on:
      - mariadb

  mariadb:
    image: mariadb:10.11
    container_name: mariadb
    restart: always
    environment:
      - TZ=Europe/Helsinki
      - MARIADB_DATABASE=test
      - MARIADB_USER=test-user
      - MARIADB_PASSWORD=strong-password
      - MARIADB_ROOT_PASSWORD=mauFJcuf5dhRMQrjj
    volumes:
      - mariadb-data:/var/lib/mysql
      - ./config/mariadb-init:/docker-entrypoint-initdb.d
    ports:
      - "3306:3306"

volumes:
    mariadb-data:
```

Figure 14: YAML file containing docker configuration for local deployment.

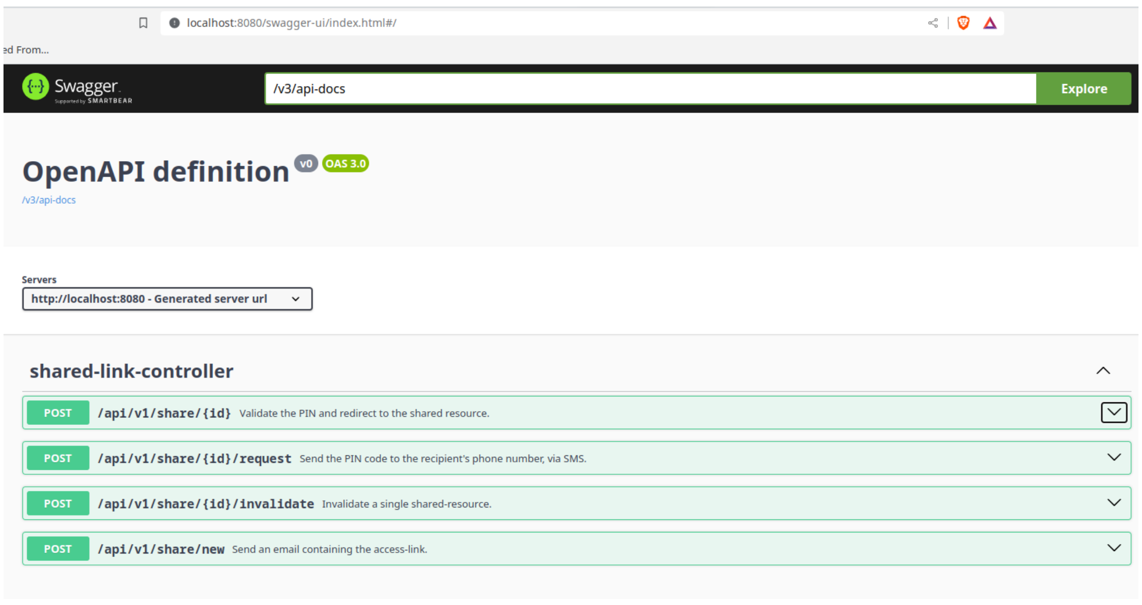
The `docker-compose.yaml` for the project specifies a multi-container setup that includes the application and a MariaDB database, simplifying the deployment process. This configuration ensures that both the application and the database are deployed together, with their dependencies correctly resolved, facilitating a seamless local setup.

- **app**: Configured to use the `openjdk:17-jdk-slim` image, ensuring the application runs in an optimized Java environment. The service maps port `8080` of the container to port `8080` of the host, making the application accessible through `http://localhost:8080`. The application's executable, a JAR file, is placed in a volume mounted from the host, allowing for easy updates without rebuilding the container.

- **mariadb**: Utilizes the mariadb:10.11 image to set up a relational database, with environment variables specifying the time-zone, database name, user, and password. Initialization scripts placed in `./config/mariadb-init` are executed to configure the database schema, ensuring the application's data storage requirements are met. Persistent data is stored in a named volume, mariadb-data, safeguarding against data loss between container restarts.

Deploying the secure file sharing application is accomplished by navigating to the directory containing the `docker-compose.yaml` file and executing `docker-compose up`. This command starts up both the application and the database services, with Docker Compose managing their lifecycle and ensuring that the database is ready before the application starts.

Comprehensive API documentation has been implemented using Swagger, offering a detailed overview of all available endpoints, request parameters, and responses. This documentation can be accessed at http://localhost:8080/swagger-ui.html.



OpenAPI definition v0 OAS 3.0

/v3/api-docs

Servers
http://localhost:8080 - Generated server url

shared-link-controller

POST /api/v1/share/{id} Validate the PIN and redirect to the shared resource.

POST /api/v1/share/{id}/request Send the PIN code to the recipient's phone number, via SMS.

POST /api/v1/share/{id}/invalidate Invalidate a single shared-resource.

POST /api/v1/share/new Send an email containing the access-link.

Figure 15: Swagger documentation.

The source code for the PoC, implemented as part of this thesis can be accessed at GitHub (10).

5 Analysis

This chapter embarks on a critical examination of the outcomes derived from the design and implementation of the PoC for secure file sharing. It aims to bridge theoretical frameworks with practical application, scrutinizing the system’s effectiveness, efficiency, and security enhancements. A particular focus is placed on the implementation of two-factor authentication (2FA) as a cornerstone for data integrity and confidentiality.

The analysis begins with an exploration of the application’s core functionality, highlighting the secure sharing mechanism enabled by the integration of two-factor authentication (2FA). The focus then shifts to the conditions governing access to shared links, including an assessment of the security protocols that determine link availability. This encompasses time-based access restrictions, user authentication status, and the impact of these protocols on the user experience.

5.1 Secure Link Sharing Overview

In the current system setup, file sharing is achieved through the generation of temporary links, like the expirable links provided by services such as Google Cloud Storage. This design decision was made to enhance security by ensuring that access to shared content does not necessitate database interactions, thereby minimizing potential vulnerabilities. The shared links are encrypted and stored in the **shared_link** table, ensuring that access is both secure and temporary. This approach highlights the project’s commitment to maintaining high security standards while offering a user-friendly file sharing solution.

- **Creating and Sharing New Links**: This section focuses on the process of creating and sharing new links, a fundamental aspect of the system’s functionality. When a user decides to share a file, the system generates a unique, temporary link. This process involves encrypting the link information to ensure its security before storing the encrypted link within the shared_link table. The temporary nature of these links is crucial; they expire after a set period, rendering the shared file inaccessible through that specific URL. This feature is designed to prevent unauthorized access and ensure that sensitive information remains protected over time.

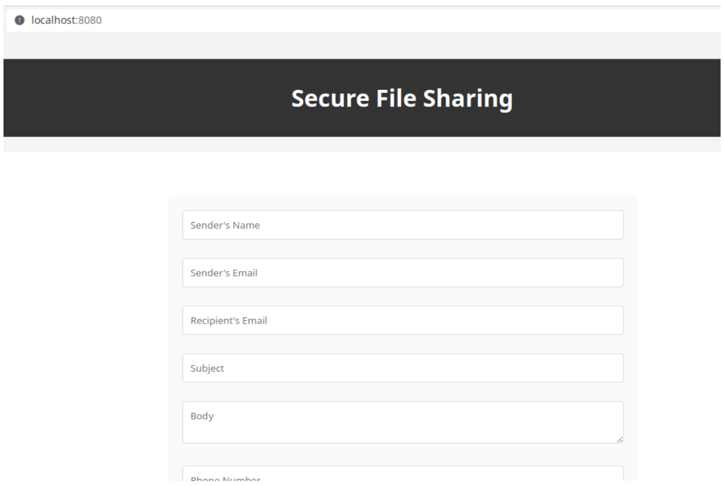
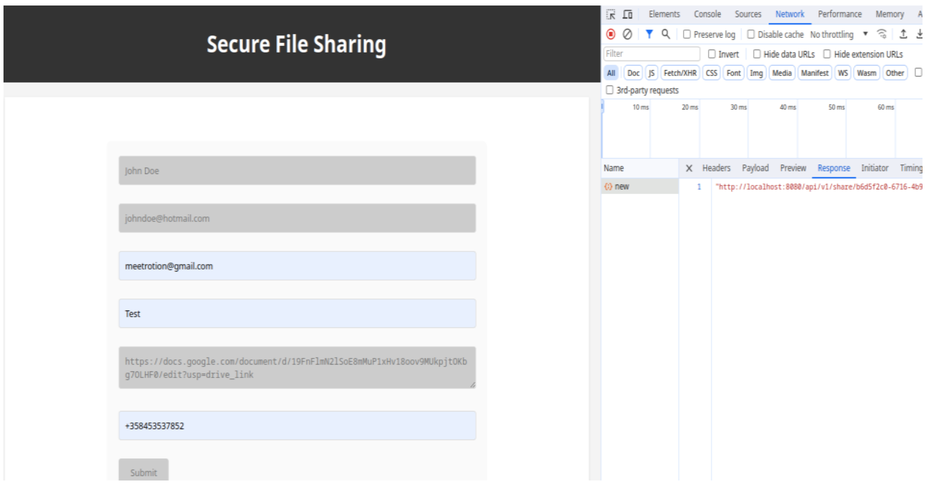


Figure 16: Landing page.



John Doe

johndoe@hotmail.com

meetrotion@gmail.com

Test

https://docs.google.com/document/d/19FnFlmN2lSoE8mMuP1xHv18oov9MUkpjtOKbg7OLHF0/edit?usp=drive_link

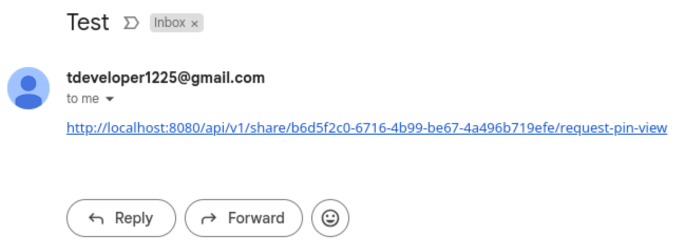
+358453537852

```
1 "http://localhost:8080/api/v1/share/b6d5f2c0-6716-4b9
```

Figure 17: Creation of new shared link.

Figure 17 shows a successful creation of shared link, with response, as seen in the developer tool view. This response is what's sent to the recipient.

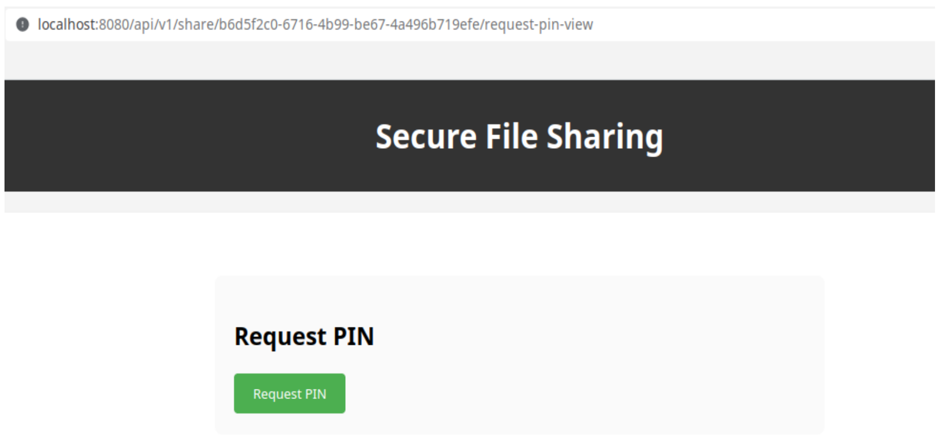
- **PIN Request**: When a recipient receives a shared link, accessing the file requires an additional layer of security: a PIN, which is sent via Short Message Service (SMS). This step ensures that even if the link falls into the wrong hands, unauthorized users cannot access the shared file without also having access to the recipient's mobile device. This process underscores the system's commitment to dual-layer security by leveraging something the user has (the link) and something the user knows (the PIN). This mechanism significantly enhances the security posture of the file-sharing process, aligning with best practices in data protection and access control.


Test Inbox ×

tdeveloper1225@gmail.com
to me

http://localhost:8080/api/v1/share/b6d5f2c0-6716-4b99-be67-4a496b719efe/request-pin-view

Figure 18: Shared link.


localhost:8080/api/v1/share/b6d5f2c0-6716-4b99-be67-4a496b719efe/request-pin-view

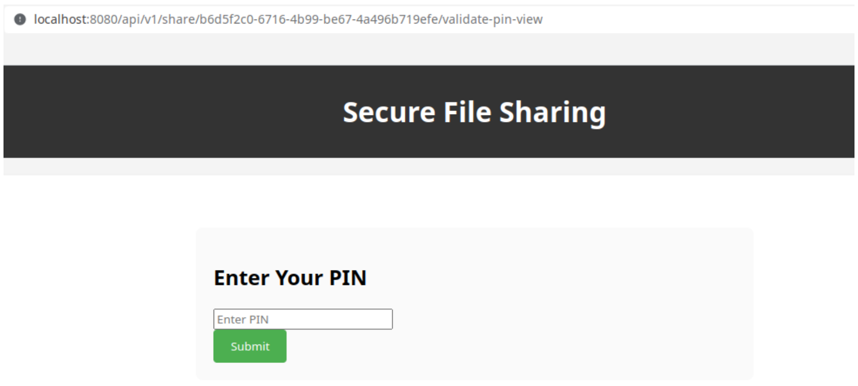
Secure File Sharing

Request PIN

Figure 19: PIN request view.

Figure 18 depicts an email message containing the shared link sent to the recipient, and Figure 19 illustrates the view to request the pin, followed by clicking the email link.

- **PIN Validation and Access**: Upon entering the PIN received via SMS, the system initiates a validation process to verify the correctness of the PIN against the one stored for the shared link. This step is critical for ensuring that access granted is legitimate. Successful validation grants the recipient access to the file, showcasing the system's efficient use of two-factor authentication to safeguard against unauthorized access.


localhost:8080/api/v1/share/b6d5f2c0-6716-4b99-be67-4a496b719efe/validate-pin-view

Secure File Sharing

Enter Your PIN

Figure 20: The view to enter the PIN.

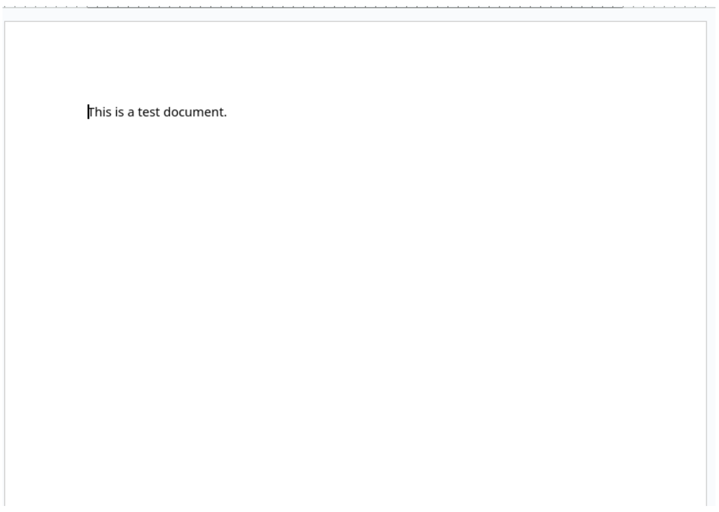

This is a test document.

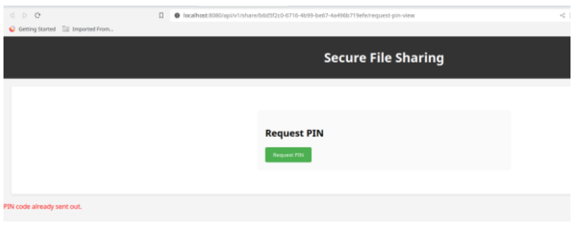
Figure 21: Successful redirection, followed by a correct PIN.

Figure 21 shows a successful redirection, rendering the shared resource, followed by entering a correct PIN in the view, as depicted in Figure 20.

5.2 Validity Checks and Response Mechanism

This section examines the system's approach to verifying shared link accessibility, focusing on scenarios that render the shared link invalid, for example, expired links and incorrect PIN inputs. Through detailed exploration of the system's validation processes and its immediate responses to these conditions, we illuminate the mechanisms in place to ensure secure and intended access, demonstrating the system's commitment to maintaining robust security protocols and user trust.

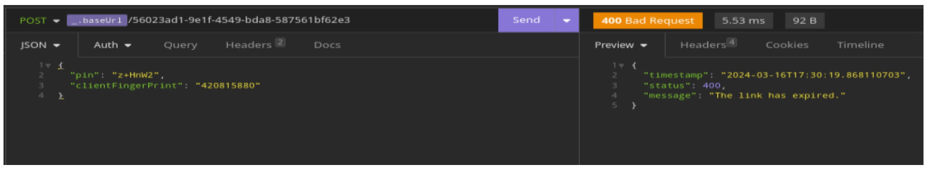
- **One-time pin distribution**: For all shared links, the PIN code for their accessibility is only sent out once. The server will respond with an error message for subsequent attempts made to request the PIN, as depicted in Figure 22.


localhost:8080/api/v1/share/b6d5f2c0-6716-4b99-be67-4a496b719efe/request-pin-view
Getting Started
Imported From...
Secure File Sharing
Request PIN
Request PIN

PIN code already sent out.

Figure 22: PIN for a resource is only sent out once.

- **Expirable link**: Once a link is shared, it maintains validity for a period of one hour. If an attempt is made to access the link after the expiry period, the system will respond accordingly, as evident in Figure 23.

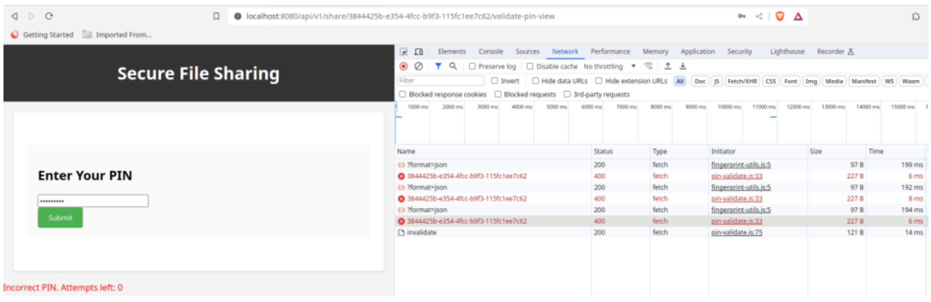


```
POST _.baseUrl/56023ad1-9e1f-4549-bda8-587561bf62e3
{
  "pin": "z+HnW2",
  "clientFingerPrint": "420815880"
}
```

```
400 Bad Request   5.53 ms   92 B
{
  "timestamp": "2024-03-16T17:30:19.868110703",
  "status": 400,
  "message": "The link has expired."
}
```

Figure 23: A link, once expired, cannot be accessed.

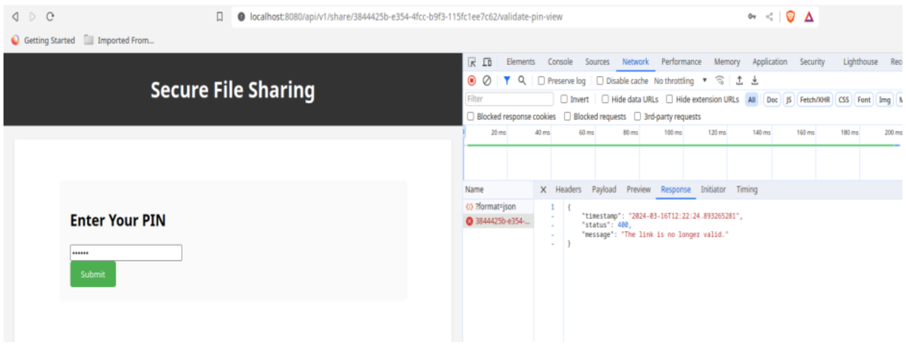
- **Invalid pin**: A correct PIN is a mandatory requirement for accessing the shared resource. Failure to provide the correct PIN three consecutive times results in the automatic invalidation of the resource.



Incorrect PIN. Attempts left: 0

Name	Status	Type	Initiator	Size	Time
?format=json	200	fetch	fingerprint-utils.js:5	97 B	199 ms
3844425b-e354-4fcc-b9f3-115fc1ee7c62	400	fetch	pin-validate.js:33	227 B	6 ms
?format=json	200	fetch	fingerprint-utils.js:5	97 B	192 ms
3844425b-e354-4fcc-b9f3-115fc1ee7c62	400	fetch	pin-validate.js:33	227 B	8 ms
?format=json	200	fetch	fingerprint-utils.js:5	97 B	194 ms
3844425b-e354-4fcc-b9f3-115fc1ee7c62	400	fetch	pin-validate.js:33	227 B	6 ms
invalidate	200	fetch	pin-validate.js:75	121 B	14 ms

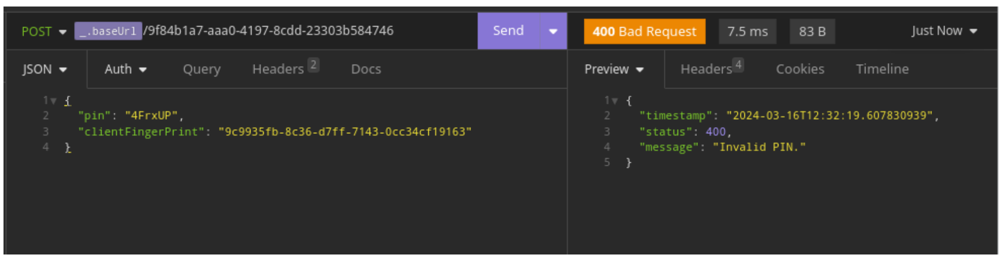
Figure 24: Link invalidated due to too many incorrect PIN.



```
{
    "timestamp": "2024-03-16T12:22:24.893265281",
    "status": 400,
    "message": "The link is no longer valid."
}
```

Figure 25: A link, once invalidated, cannot be accessed despite correct PIN.

- **Incorrect Device Fingerprint**: The preparation and sending of the PIN via SMS are subject to a unique device fingerprint, representing the device ID from which the recipient follows the email link. When validating the entered PIN, the validation must occur on the same device used to request the PIN.



```
{
  "pin": "4FrxUP",
  "clientFingerPrint": "9c9935fb-8c36-d7ff-7143-0cc34cf19163"
}
```

```
{
  "timestamp": "2024-03-16T12:32:19.607830939",
  "status": 400,
  "message": "Invalid PIN."
}
```

Figure 26: Correct device fingerprint is imperative for accessing the link.

The analysis conducted in this chapter is instrumental in evaluating the effectiveness of the designed secure file sharing system. By focusing solely on the solution’s operational aspects, it lays the groundwork for a comprehensive understanding of its strengths and areas for improvement. This analytical approach ensures a focused discussion, paving the way for a concluding chapter that revisits the research objectives and encapsulates the overall findings.

6 Conclusion

The thesis embarked on a journey that went beyond meeting a practical business requirement, delving into contributions to the academic and professional realms of information technology and networking services through the comprehensive study and development of a secure file-sharing solution.

The objective of this thesis was to engineer a system facilitating secure file sharing, adhering to the stringent restrictions and regulatory standards faced by modern businesses. Motivated by the pressing need for organizations to share confidential information with external parties while maintaining the highest levels of confidentiality and integrity, this endeavour aimed to ensure compliance with strict data protection laws such as the GDPR. The research led to the design and implementation of a system employing two-factor authentication (2FA), aimed at significantly reducing the likelihood of unauthorized access. The development leveraged contemporary technologies, including Spring Boot, Hibernate, jOOQ, and Docker, to produce a system that is both reliable and secure.

The evaluation of the developed system validated its effectiveness in enhancing file-sharing security within an ERP system. Utilizing temporary encrypted links for file sharing and PIN verification via SMS ensures that files are accessible only to their intended recipients within a designated timeframe. This methodology embodies secure file-sharing principles from both practical and regulatory perspectives, thus meeting legal requirements. Furthermore, this thesis lays the groundwork for future research and development in the domain of secure file sharing. The proposed system’s architecture and technology stack are well-positioned for scalability and adaptability to meet the evolving needs and dynamic challenges of cybersecurity threats.

The design and successful implementation of the secure file-sharing system mark a significant advancement towards secure data exchange in ERP systems. This thesis contributes insightful and practical solutions to the persistent challenge of protecting sensitive information in an increasingly digital business environment.

As businesses navigate the complexities of data protection and cybersecurity, the insights and developments presented in this thesis highlight a promising path toward more secure and efficient file-sharing practices. This work addresses a real-world challenge through detailed technical research and practical application, paving the way for a transformative approach to how businesses manage and share sensitive data in this ever-changing realm of technology, where industries must continue to evolve and adapt for future advancements.

References

1. Rissanen T. Electronic identity in Finland: ID cards vs. bank IDs. Identity in the Information Society. 2010 July 1; 3(1): 175-194. Available from: https://doi.org/10.1007/s12394-010-0049-8.

2. Traficom. Traficom. [Online]. [cited 2023 November. Available from: https://www.kyberturvallisuuskeskus.fi/en/our-activities/regulation-and-supervision/electronic-identification.

3. Eminagaoglu M, Cini E, Sert G, Zor D. A Two-Factor Authentication System with QR Codes for Web and Mobile Applications. In 2014 Fifth International Conference on Emerging Security Technologies; 2014; Alcala de Henares: IEEE. p. 105-112. Available from: https://ieeexplore-ieee-org.ezproxy.metropolia.fi/document/6982784.

4. Iyanda AR, Fasasi ME. Development of Two-factor Authentication Login System Using Dynamic Password with SMS Verification. International Journal of Education and Management Engineering (IJEME). 2022 June; 12(3): 13-21. Available from: https://www.mecs-press.org/ijeme/ijeme-v12-n3/v12n3-2.html.

5. Hölzl M, Roland M, Mayrhofer R. Real-World Identification: Towards a Privacy-Aware Mobile eID for Physical and Offline Verification. In Proceedings of the 14th International Conference on Advances in Mobile Computing and Multimedia (MoMM 2016); 2016; Singapore. p. 280-283. Available from: https://www.researchgate.net/publication/313108198_Real-World_Identification_Towards_a_Privacy-Aware_Mobile_eID_for_Physical_and_Offline_Verification.

6. Krupa W, Parkkonen M, Stempel K, Baglan N, Kontopoulou V. A Critical Assessment of the Strong Authentication System Using Bank Credentials: The Case Study of Finland.; 2022 [cited 2023 November 17. Available from: https://www.helsinki.fi/assets/drupal/2022-05/A%20critical%20assessment%20of%20the%20strong%20authentication%20system%20using%20bank%20credentials.pdf.

7. Suomi.fi. https://www.suomi.fi/instructions-and-support. [Online]. [cited 2023. Available from: https://www.suomi.fi/instructions-and-support.

8. Bruzgiene R, Jurgilas K. Securing Remote Access to Information Systems of Critical Infrastructure Using Two-Factor Authentication. Electronics. 2021; 10(15). Available from: https://doi.org/10.3390/electronics10151819.

9. Rao R, Swamy SR. Review on Spring Boot and Spring Webflux for Reactive Web Development. International Research Journal of Engineering and Technology (IRJET). 2020 April; 7(4): 3834-3837. Available from: https://www.researchgate.net/publication/341151097_Review_on_Spring_Boot_and_Spring_Webflux_for_Reactive_Web_Development.

10. Upreti R. GitHub. [Online].; 2023 [cited 2024. Available from: https://github.com/roshanupreti/secure-file-sharing.