SafeVault: Secure File Sharing and Access Management System

CS-699 Project Report

Submitted by

Darshit Bimal Gandhi

(22M0824)

Shamik Kumar De

(22M0822)

Abstract

Nowadays, with the increase in the amount of data and to share that data with multiple users, there arises a need for storing it securely in some shared storage.

But normally, the data is stored in Plaintext mode in the shared storage. This leads to the generation of serious concern on data confidentiality in a system. The files should be stored in an encrypted form in the shared storage and only the authorized users should be able to access/decrypt them.

Thus, a solution to this will be storing files in an encrypted form on a system and retrieving and decrypting it as and when required by an authenticated user.

To implement these functionalities, our model has distributed modules for different schemes leading to an establishment of an integrated and secure system.

Description

Our project comprises a basic software implementation of **passkey-based encryption of files** before uploading them on a secure storage server to prevent breach of data and further **decryption of the files** before downloading them on demand of a authenticated user. Also, there are some extra features like **token-based session management** and **access control** on the files.

Technology Stack:

1 Spring Boot

2 PostgreSQL Database

3 HTML

4 CSS

5 JavaScript

6 Latex

7 Eclipse and VS Code (IDE)

8 Git

Modules:

1. User Registration.

In this module, new user registration is done using various fields like name, username, email id, password and contact details. All the input validation will be done in this module and the user will not be able to register with invalid data.

2. User Login.

This module enables the pre-registered users to login into their accounts. A list of authenticated users is maintained and this is used when users log in and request services. When some user logs in, the session of that current user is been stored in the backend in a table (table contains all the user ids of the users that are currently logged in). Also, after storing the userid in the backend, a **token** is been passed to the frontend and is stored in the browser and whenever the user tries to perform some operation, then that token (which was stored in the frontend) will be sent along with the other data to the backend which will help in **Session Management**. If the token is not found in the backend database, then the user will be redirected to the login page.

3. File Upload to the shared storage.

When the user wants to upload a file, then he needs to select a file and along with that, he needs to select a **pass-phrase**. This pass-phrase will be responsible for the generation of a unique key at the server side, and from this key, the server will **encrypt the file** and will **store that file in the common storage**. Now the files are encrypted and so they can be securely stored in the common storage.

4. File Download and Access control.

Now, suppose, some user wants to access the file, then he needs to ask the access permission from the owner. When someone ask for the access of a file, then the owner will be **notified** for this, and now he has the option to **accept or decline the request**. If the owner accepts the request, then the server will send the requested user the pass-phrase to the requested user, and the requested user can use that passphrase to download and decrypt the file. If the owner declines the request, then the requested user will get access denied. This is how the **access control management** will work.

After decrypting, the **file will be downloaded** in the client machine. So, even if the file is stored in common storage, it will not be seen by any other person, if he is not having the access.

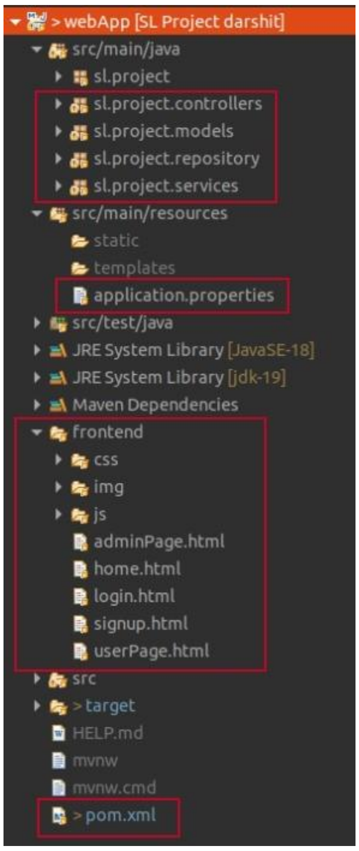
5. Deletion of Files.

There is an option for the user to delete the files which have been uploaded before.

Directories and File System

Four layered structure have been used in the Spring boot backend application.

- **Model:** It is a class-level annotation. It is used to denote a class as a model. We used @model across the application to mark the beans as Spring's managed components.
- **Services**: This encapsulates the main business logic. For instance, the LoginSignupService is responsible for creating an account, and performing the required logic to register the user or login.
- **Repository**: These were used between the service layer and the model layer. For instance, in the LoginSignupRepository methods were created that contained the code to read/write from the database.
- **Controllers**: This layer acts as a gateway between the input and the domain logic. It mainly decides what to do with the input data and how to output the response data.



```
> webApp [SL Project darshit]
  src/main/java
    sl.project
    sl.project.controllers
    sl.project.models
    sl.project.repository
    sl.project.services
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-18]
  JRE System Library [jdk-19]
  Maven Dependencies
  frontend
    css
    img
    js
    adminPage.html
    home.html
    login.html
    signup.html
    userPage.html
  src
  > target
  HELP.md
  mvnw
  mvnw.cmd
  > pom.xml
```

Our Directory Structure.

Compilation/ Running Instructions

- The frontend code does not need any kind of compilation, one can run the "home.html" file directly on the local browser.
- When using Eclipse IDE for the backend code, one can build the maven project by right clicking on the project base directory and then "Run As">"Maven Build" and as soon as the window pops up type "clean install" in "Goals" text box and then run.
- Once the Maven Build is successful, one can run the application by right clicking on base directory then "Run As">"Java Application".

Possible future work:

Possible works that can be implemented are as follows:

- We can extend our application so that it can store files with various other extensions like jpg, png, etc.
- Users may be provided with a reset password link.