Tampere University

COMP.SEC.300 – Secure Programming

Searchable Symmetric Encryption

Exercise work

Group members

Muhammad Aftab Hameed (150904111)

Saqib Hussain

Table of Contents

[General Description: 3](#_Toc133447750)

[Structure of program: 6](#_Toc133447751)

[Secure programming solutions / practices: 8](#_Toc133447752)

[Security Testing: 8](#_Toc133447753)

[Security issues or vulnerabilities: 9](#_Toc133447754)

[Suggestions for improvement: 9](#_Toc133447755)

[Some other usages of SSE scheme: 9](#_Toc133447756)

# General Description:

**Introduction:** Searchable Symmetric Encryption (SSE) is a technique of storing the encrypted data into the database. Then it allows user to search over the encrypted database without need of decrypting the data first. SSE allows user to perform search over the encrypted database securely and efficiently without decrypting the database.

We implemented the SSE to upload and encrypted and store encrypted file into the database. Then user can perform search queries over the encrypted database to find that file and if file exists user can download it. Once the file is found system decrypts it and allows user to download decrypted file.

**Tools and technologies:** The system can be divided into three parts; frontend, backend, and database. Frontend is developed using React while backend is developed using .Net Core (C#). For the database technologies we used MS SQL Server and SQL database.

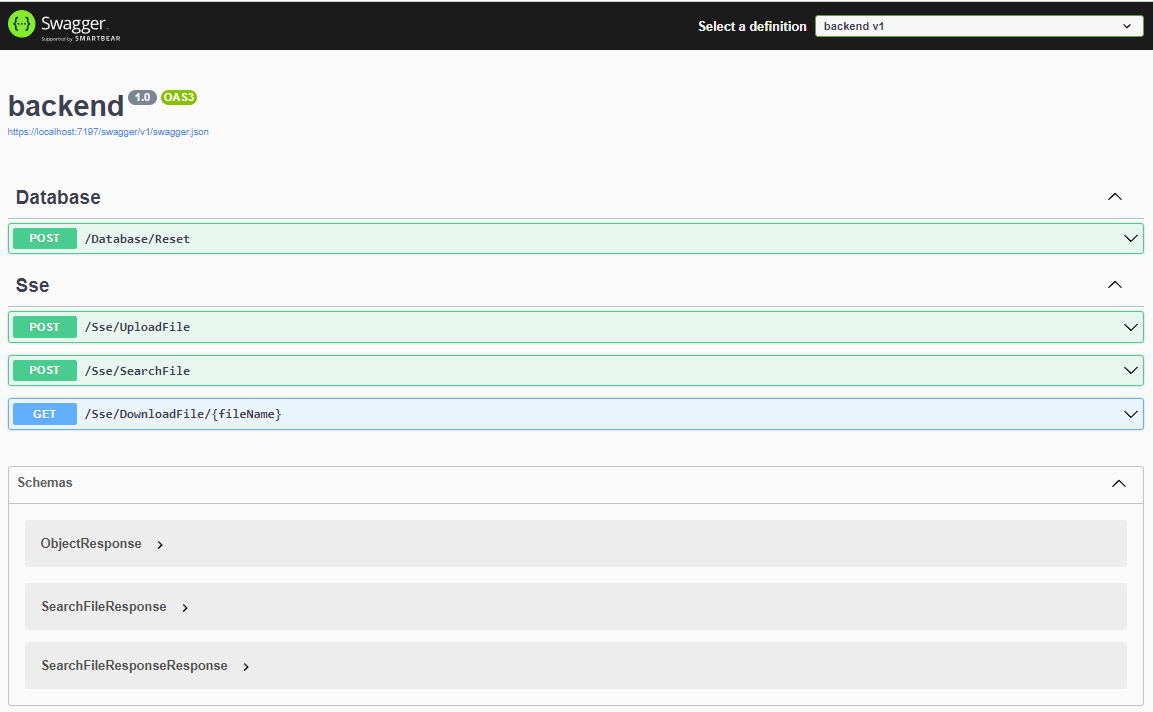
React is a JavaScript library used to develop frontend single page applications.

.Net Core is a framework that allows user to develop cross platform applications.

MS SQL Server is a database management system studio that allows user to maintain the SQL database.

**Backend:**

First, we have to open the backend project in visual studio and execute the project. On executing the project, it will open the backend website in the browser. Swagger API documentation will be displayed in the document as below which give provide all the information regarding APIs.

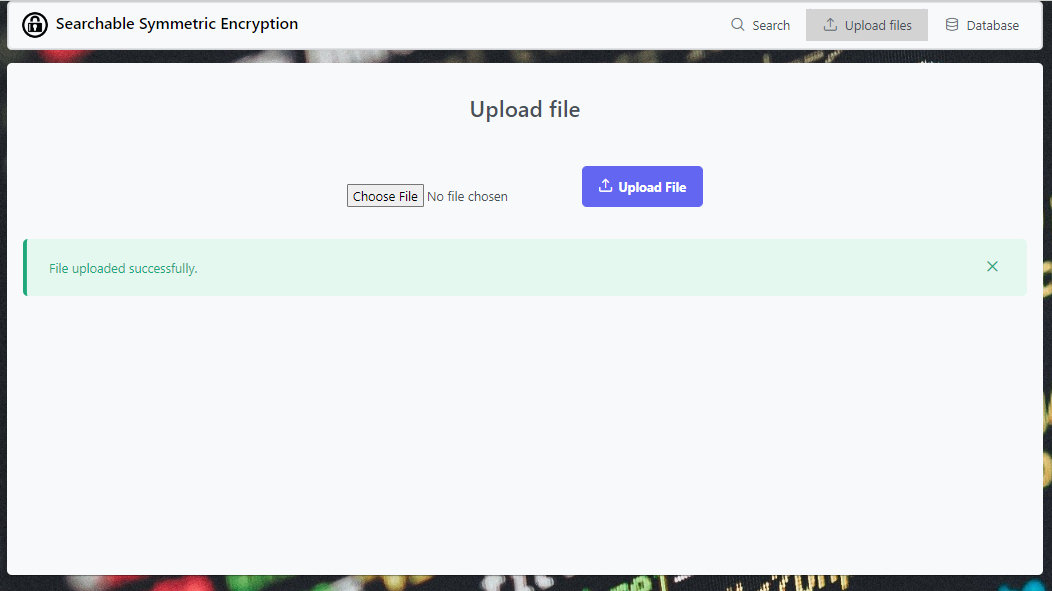


**Frontend:**

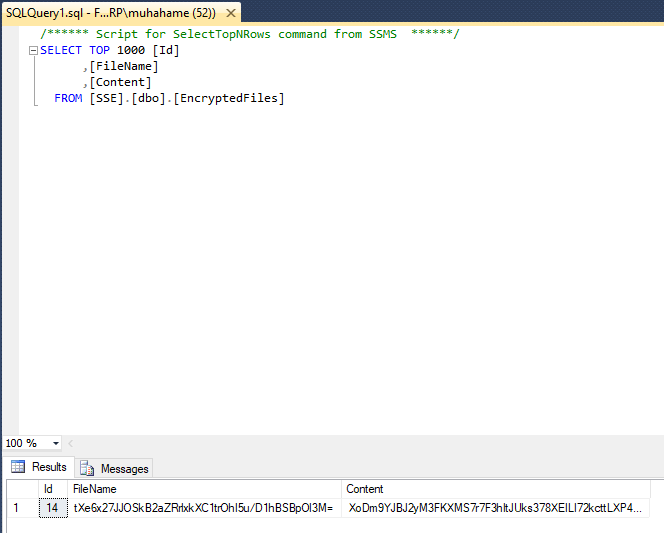
For running frontend website, we need to open frontend folder in Visual Studio Code and run **npm start** in the terminal. It will start the application and will open in the browser as shown below:



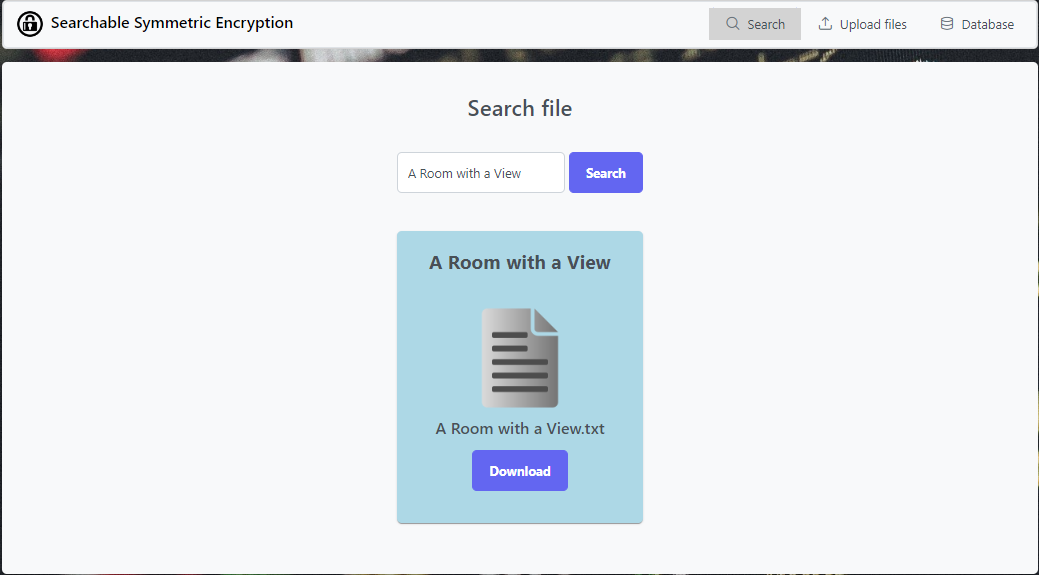
On upload files page, we can upload files to the database as below:



We can see that uploaded file has been encrypted and saved into the database.

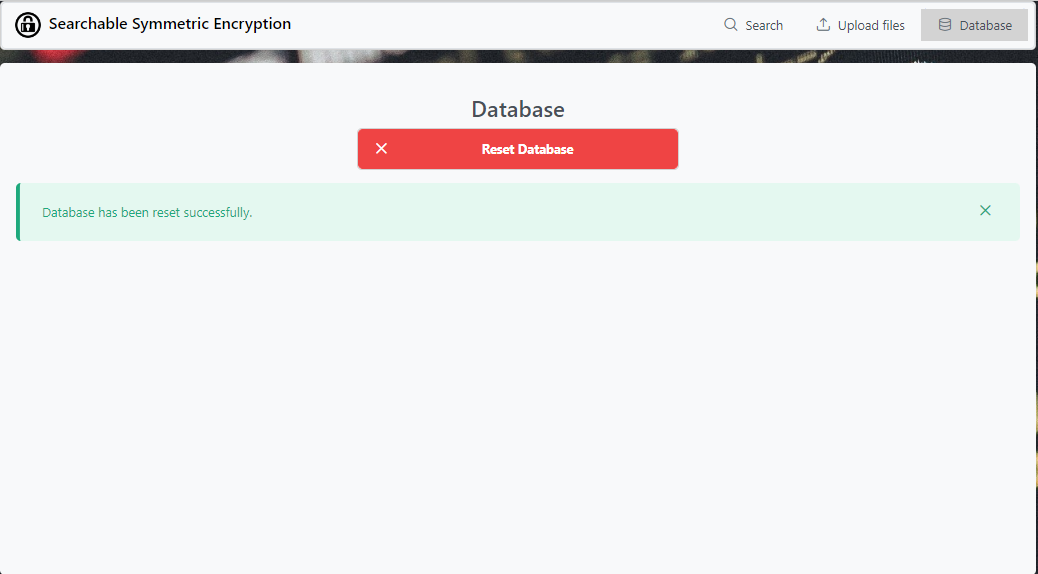


Now we search that file from the encrypted database using filename as below:



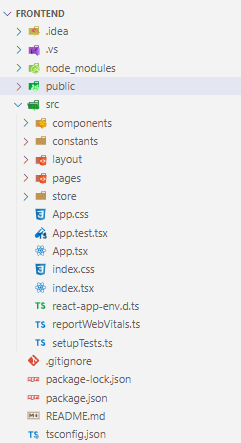
By clicking download, we can download this file as unencrypted file.

On database page, we can reset and clear our database by click reset database button as shown in below image:



# Structure of program:

**Frontend:**



**src:** src directory contains all the program logic and source code.

**components:** Our frontend application is divided into multiple components responsible for our frontend application as a Single Page Application.

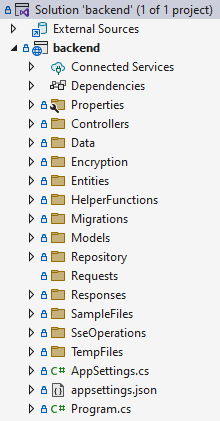
**constants:** This folder contains all the configuration constants which are being used by the application on various places.

**layout:** layout folder contains the theme or layout of the project.

**pages:** This folder contains multiple pages of our application i.e., Home, Search, Upload and Database.

**store:** This folder contains stores of the application which is responsible for consuming the backend APIs.

**Backend:**



**Controllers:** Since this is an API application that follows MVC structure, controllers are responsible for handling the API requests receiving from the frontend application.

**Data:** It contains the database context which contains all the DbSets of our database.

**Encryption:** This class contains encryption and hashing operations.

**Entities:** Entities folder contains respective classes to the database tables.

**HelperFunctions:** This folder contains helper functions used by the application.

**Migrations:** Migrations folder contains all the migrations of our database. Since we used Code First approach with Entity Framework Core, we have migrations for our all the database changes.

**Models:** This folder contains our model classes.

**Repository:** This class is responsible for interacting with the database since it contains all the database operations.

**Responses:** Our APIs return custom response objects, so this folder contains return response objects.

**SseOperations:** This class is responsible for performing SSE Operations for instance, search operation.

# Secure programming solutions / practices:

In the implementation of this system, we followed best practices of secure programming which are explained as below:

**Exception handling:** Every piece of code is written in the try catch block so we can catch exceptions instead of crashing down our system.

**Error handling:** We handled system defined (exceptions) and user defined errors very well which give proper message when an operation either fails or succeeds.

**File Validations:** We applied following file validations.

* ensure that user uploads actual text files.
* file extension must be .txt
* file size must not be greater than 5 MB.
* file must be safe (means file must not contains executable code or malicious script)
* file name must be unique.

**Encrypted database:** Our entire database is encrypted since it is an implementation of SSE technique.

# Security Testing:

During the testing phase, we tried to violate all the validations of this system. However, fortunately, every validation is working fine. Some of the Security Test Cases are given below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case description | Test steps | Expected results | Actual results | Pass/Fail |
| File validation | Upload any file other than text file | System should reject the input and return the error that file type is not allowed | System rejected the input and returned the error that file type is not allowed | Pass |
| Upload file larger than 5MB | System should return an error “File cannot be larger than 5mb.” | System returned an error “File cannot be larger than 5mb.” | Pass |
| Upload any file other than .txt extension | System should return an error “Allowed file extension is only .txt” | System returned an error “Allowed file extension is only .txt” | Pass |
| Upload exe file by changing the extension to .txt | System should not upload this file and should return an error. | System didn’t upload that file and returned an error “File is not safe.” | Pass |
| Upload a file twice with the same name | System should reject the file and return error that file already exists. | System rejected the file and returned the error that file already exists. | Pass |
| Error handling | Try to click upload button without selecting a file | System should return an error “File is required”. | System returned an error “File is required”. | Pass |

# Security issues or vulnerabilities:

We couldn’t find any security issues or vulnerabilities in our system.

# Suggestions for improvement:

Following are some suggestions for improvement of this system:

* Search by filename should be case insensitive.
* Remove limitation that user has to remember the exact filename.

# Some other usages of SSE scheme:

Some other usages of SSE scheme are as follows:

* **Cloud storage:** SSE is often used to encrypt data stored in cloud storage services like Dropbox, Google Drive, and OneDrive. This allows users to securely store and search their data without revealing its contents to the cloud provider.
* **Healthcare:** SSE is used to protect patient privacy in healthcare systems. Medical records are encrypted using SSE, which allows healthcare professionals to search for and retrieve records without exposing sensitive patient data.
* **Financial services:** SSE is used in financial services to protect customer data. Bank account information and transaction histories can be encrypted using SSE, which allows financial professionals to search for and retrieve data while maintaining the confidentiality of the customer's financial information.
* **Messaging applications:** SSE is used to protect the privacy of messaging applications. Chat histories can be encrypted using SSE, which allows users to search for and retrieve messages without exposing the contents of the messages to the messaging service.