**1. Overview**

The File Signer Application is a web-based solution that enables users to securely upload files, digitally sign them using RSA keys, store and manage these signed files, and verify signatures for data integrity. The system offers an intuitive interface for users to manage files, keys, and signatures with full CRUD operations.

The application is built using Spring Boot (Java), with Thymeleaf and Bootstrap for the front end.

**2. Features**

* **File Upload and Signing**  
  Users can upload files and their cryptographic keys, then generate a digital signature. The signed data is stored securely in the database.
* **Signature Verification**  
  Users can verify the integrity of files using the uploaded public key and signature.
* **Key Pair Generation**  
  The application can generate new RSA key pairs (2048 bits) on demand and provide them for download in a ZIP file.
* **File Management**  
  Users can download original files, download signatures, and delete signed entries.
* **Notification System**  
  Uses Bootstrap Toast messages to inform the user of success or failure.

**3. Architecture**

**Packages:**

* com.tarek.filesignerapp.controller  
  Contains Spring MVC controllers handling user requests and routing.
* com.tarek.filesignerapp.service  
  Contains service classes for core business logic, including signing, verification, and key management.
* com.tarek.filesignerapp.model  
  Defines the JPA entity SignProcess used to store signed file data.
* com.tarek.filesignerapp.repository  
  Contains the Spring Data JPA repository interface for CRUD operations.
* templates  
  Contains the Thymeleaf HTML template(s).

**4. Entity Design**

**SignProcess**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | Long | Auto-generated primary key |
| fileName | String | Uploaded file name |
| privateKeyName | String | Uploaded private key file name |
| publicKeyName | String | Uploaded public key file name |
| signName | String | Chosen signature name |
| fileData | byte[] | File contents |
| privateKeyData | byte[] | Private key contents |
| publicKeyData | byte[] | Public key contents |
| signature | byte[] | Generated digital signature |
|  |  |  |

**5. Controller Endpoints**

| **URL** | **Method** | **Description** |
| --- | --- | --- |
| /file-signer | GET | Loads the home page |
| /sign | POST | Upload and sign a file |
| /verify | POST | Verify a file’s signature |
| /generateKeys | GET | Generate and download a key pair (ZIP) |
| /downloadFile/{id} | GET | Download a signed file |
| /downloadSignature/{id} | GET | Download the signature of a signed file |
| /delete/{id} | GET | Delete a signed file record |

**6. Service Methods**

**SignProcessService**

* uploadAndSign()  
  Handles the upload of a file, private key, public key, and signature name, then uses the private key to generate a digital signature on the uploaded file contents. The resulting data is stored in the database.
* verifySignature()  
  Verifies a file’s signature using a provided public key. Uses RSA SHA256 verification.
* allSigns()  
  Returns a list of all stored sign processes.
* findById()  
  Looks up a sign process by its ID.
* delete()  
  Removes a sign process by its ID from the database.

**KeyPairService**

* init()  
  Initializes an RSA key pair when the application starts.
* getPrivateKey()  
  Returns the generated private key.
* getPublicKey()  
  Returns the generated public key.
* getPrivateKeyEncoded()  
  Returns the encoded byte array of the private key.
* getPublicKeyEncoded()  
  Returns the encoded byte array of the public key.

**7. User Interface**

* Built using **Thymeleaf**
* Responsive layout with **Bootstrap 5**
* Supports uploading, verifying, and listing files
* Shows notifications via Toast messages
* Allows generating new key pairs for signing

**8. How It Works (Functional Flow)**

1. **Home Page Load**
   * The home page displays all existing signed files in a table.
2. **Upload and Sign**
   * User uploads a file, provides private/public key files, and gives a signature name.
   * The server generates the digital signature using the private key and stores the result.
3. **Verify**
   * User uploads the file to verify, its signature, and the public key.
   * The system checks if the signature matches the file contents.
4. **Key Pair Generation**
   * Generates a fresh RSA key pair, which the user can download as a ZIP file.
5. **Download / Delete**
   * User can download the original file or the signature, or delete the record.

**9. Technologies Used**

* Java 17+
* Spring Boot
* Spring MVC
* Spring Data JPA
* Thymeleaf
* Bootstrap 5
* MySQL

**10. Running the Application**

1. Build with Maven:

mvn clean install

1. Run:

java -jar target/file-signer-app.jar or just run using IntelliJ IDEA

1. Then access:

http://localhost:8585/file-signer

(after setting your server port to 8585 and context path to /file-signer)

**11. Deployment Notes**

* Application properties :

1. spring.application.name=fileSignerApp
2. server.servlet.context-path=/file-signer
3. server.port=8585
4. spring.datasource.url=jdbc:mysql://localhost:3306/filesignature
5. spring.datasource.username=springstudent
6. spring.datasource.password=springstudent