Relatório Desafio Bry
Arthur Pellenz Winck
16/09/25

Como Buildar o Projeto:

Pré-requisitos:

- Java 17 (17.0.16)
- Maven 3.9.10
- IntelliJ (Ou outra IDE, para esse passo a passo estaremos utilizando os *runners* pela IDE)
- Recomendo para a utilização de mais de uma versão java o SDKMan que permite que existam vários *candidates* para a versão atual do Java e do Maven, entre outras linguagens/gerenciadores semelhantes.

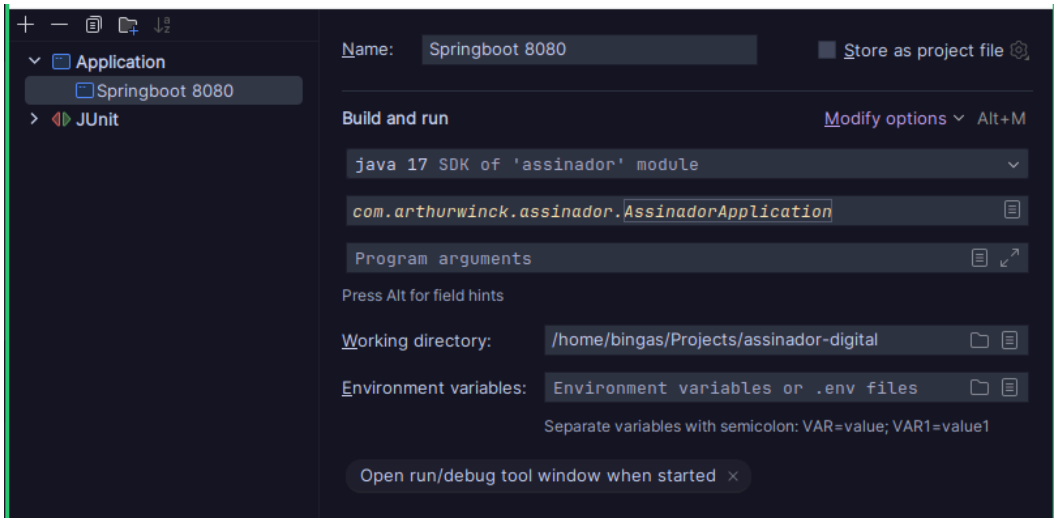
1. Clone o repositório do projeto

   git clone https://github.com/arthurwinck/assinador-digital

2. Builde e instale o projeto com:

   mvn clean install -U

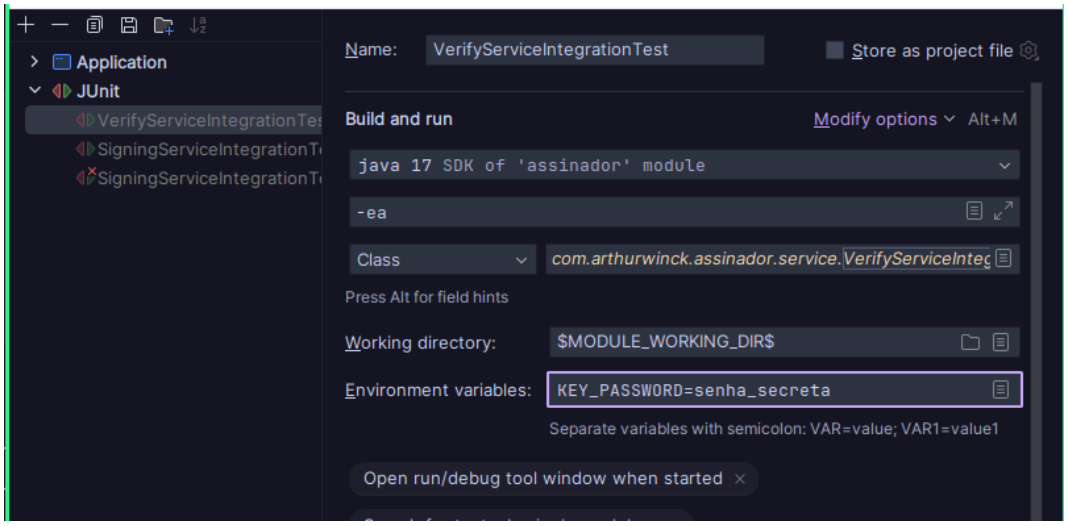
3. Crie um *runner* dentro de Run/Debug Configurations com as seguintes propriedades:



4. Inicialize o runner pelo Debug (ou Runner) com F5 para Debug e Ctrl + F5 pra Run

5. A API estará disponível no root path. requisições para localhost:8080. Ex: localhost:8080/verify

6. Para a execução dos testes de integração, é necessário alterar o *runner* padrão do teste do JUnit para que seja possível utilizar a senha da chave privada:



7. Além disso, é necessário disponibilizar o arquivo .pfx no path src/test/resources/keys. Dessa forma é possível executar os dois testes de integração que utilizam o certificado e a chave privada.

Desenvolvimento:

A estrutura desenvolvida é a seguinte: Resources são os arquivos responsáveis por gerenciar as requisições e os Services os arquivos responsáveis pela lógica de negócio: realizar a assinatura/verificação de assinatura e devolvê-la para o usuário.

De início para facilitar o teste das funcionalidades, foram criados 3 resources, o *HashResource*, *SigningResource* e *VerifyResource.* O desenvolvimento do HashService demonstrou-se bem simples, sendo sua funcionalidade somente retornar o hash SHA512 de uma string ou de um arquivo (em /hash/upload). Os dois outros services foram mais desafiadores, sendo necessário ler a documentação do BouncyCastle, e outros sites de ajuda por conta de nunca ter utilizado a biblioteca. Porém, ela é robusta e possui muita documentação sobre seus métodos e como utilizá-los.

Outra dificuldade foi na declaração do tipo do arquivo da assinatura salva, algumas horas foram gastas debuggando o motivo pelo qual o arquivo .p7 não podia ser lido corretamente nas validações. Somente após esse tempo ajustei para que os arquivos sempre fossem salvos no formato correto, .p7m, indicando a presença do conteúdo assinado dentro da assinatura.

Os testes unitários se mostraram um desafio, pois seria necessário realizar o “mock” de todos os retornos das classes e métodos da biblioteca BouncyCastle. Para conseguir realizar um teste mais completo sem que seja necessário “mockar” muitos retornos de métodos de bibliotecas, foram criados dois arquivos de teste, o VerifyServiceIntegrationTest e o SigningServiceIntegrationTest.

Testes unitários relacionados aos métodos auxiliares, e autenticação para uso são algumas das melhorias que gostaria de implementar caso possuísse mais tempo. Outro fator seria mudar a maneira que realizamos a assinatura pois estamos enviando a senha de uma privada para o servidor, de forma que qualquer atacante que pudesse interceptar a mensagem poderia roubar a chave privada daquele usuário.

Validação:

Pré-requisitos:

- Senha presente na variável de ambiente *KEY_PASSWORD* para testes de integração SigningServiceIntegrationTest e VerifyServiceIntegrationTest.

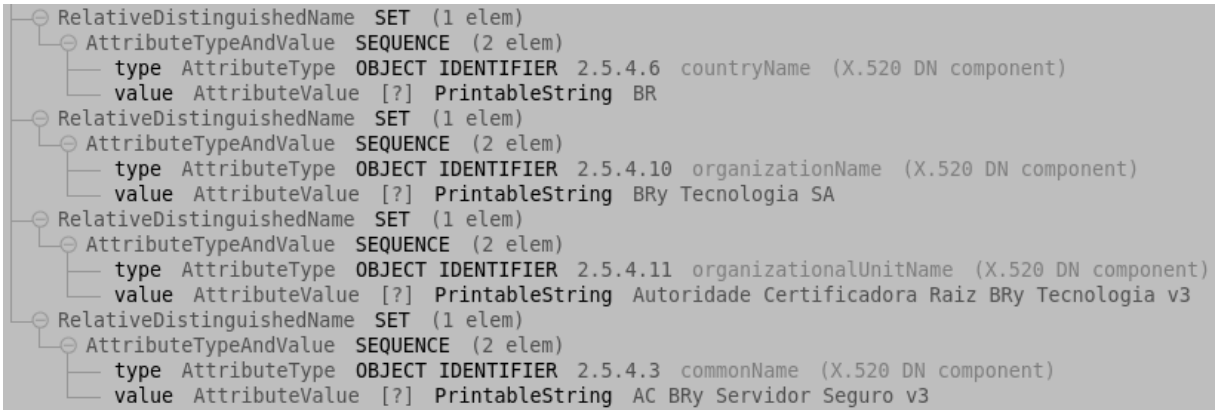
Assinatura

CURL realizado para assinatura do arquivo doc.txt:

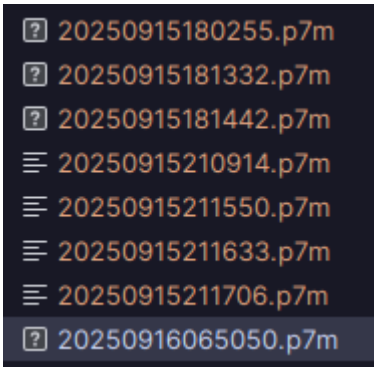
*curl -X POST http://localhost:8080/signature -F "file=@./src/main/resources/files/doc.txt" -F "pkcs12=@./src/main/resources/keys/certificado_teste_hub.pfx" -H "X-password: *********"*

O arquivo gerado é salvo com o timestamp do momento em que foi gerado.

Para validar o resultado, também foi utilizado o [ASN.1 JavaScript decoder]. Nele podemos ver as informações sobre o certificado digital que foi utilizado para a assinatura do documento



Exemplo de assinaturas geradas:



Testes de integração foram realizados para que fosse possível verificar que o fluxo de assinatura funciona corretamente com strings.

Verificação

CURL realizado utilizando o arquivo “20250915180255.p7m” que acabou de ser gerado pelo passo anterior:

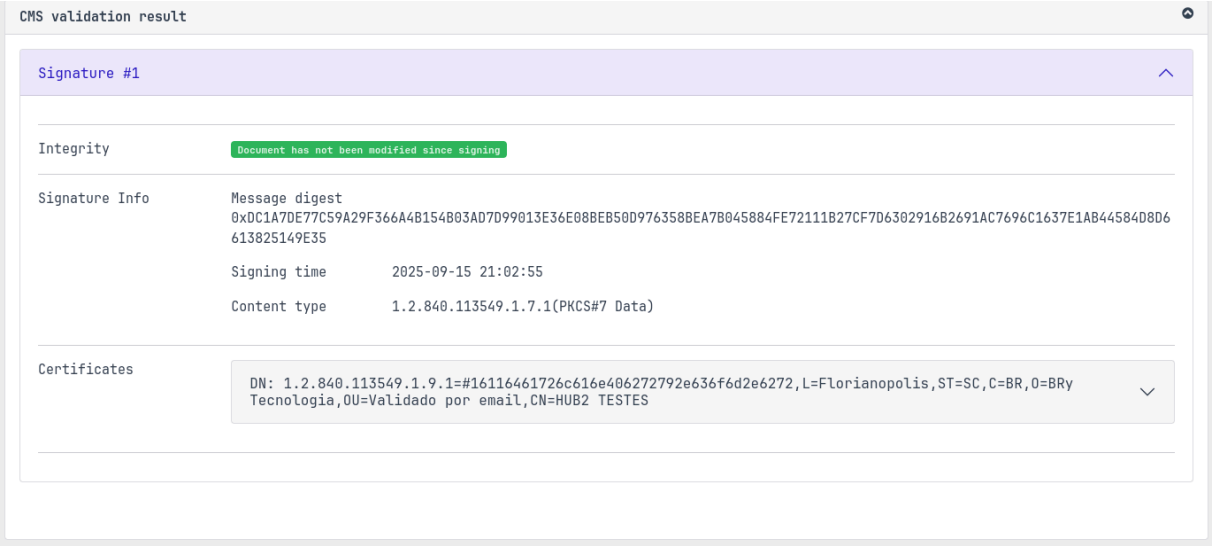
http://localhost:8080/verify -F "file=@./20250915180255.p7m"

Resultado:

Alternativamente, em texto:

{"originalData":"546573746520766167612062616 36b2d656e64204a617661","status":"VALIDO","signinTimeDate":null,"encapContentInfoHash":null,"digestAlgorithm":"SHA512","cnsignerName":null}

Como teste de controle, foi feita também a validação por meio do site CMS Validator, tendo como resultado:



Referências:

https://stackoverflow.com/questions/27917846/explore-a-bouncy-castle-store-object
https://javadoc.io/doc/org.bouncycastle/bcpkix-jdk15to18/latest/index.html
https://stackoverflow.com/questions/35099408/generate-a-cms-pkcs7-file-with-bouncycastle-in-c-sharp
https://downloads.bouncycastle.org/java/docs/bcpkix-jdk13-javadoc/org/bouncycastle/cms/SignerInformationVerifier.html