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.

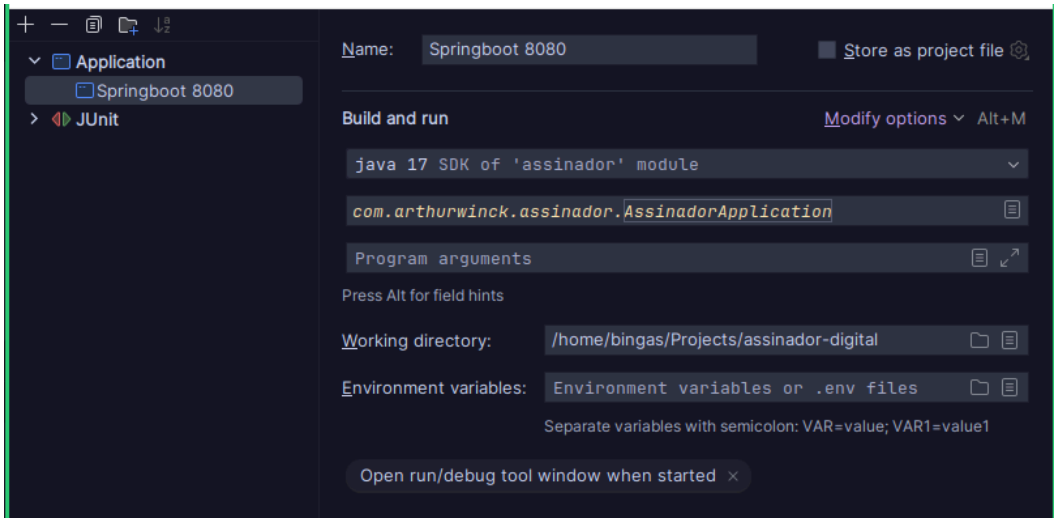
1. Clone o repositório do projeto

   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:



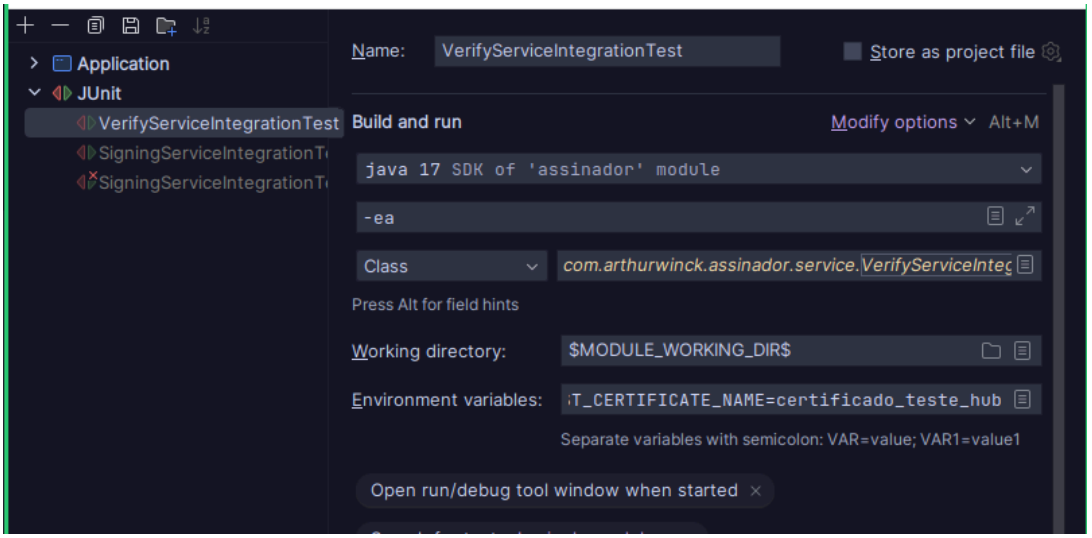
4. Inicialize o runner pelo Debug (ou Runner) com F5 para Debug e Ctrl + F5 pra Run

5. A API estará disponível no root path (caminho raíz). Requisições são feitas para a porta 8080 por padrão. 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 e também o certificado:

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

- Nome do certificado presente na variável de ambiente *TEST_CERTIFICATE_NAME*. Nome padrão do artefato é: “certificado_teste_hub”



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.

Por fim, outro ponto de melhoria do código é a implementação de mensagens de validação mais específicas. Erros de senha, ou de arquivos inutilizáveis retornam o mesmo erro de servidor e não indicam ao usuário exatamente qual foi o problema para que uma assinatura não tenha dado certo.

Validação

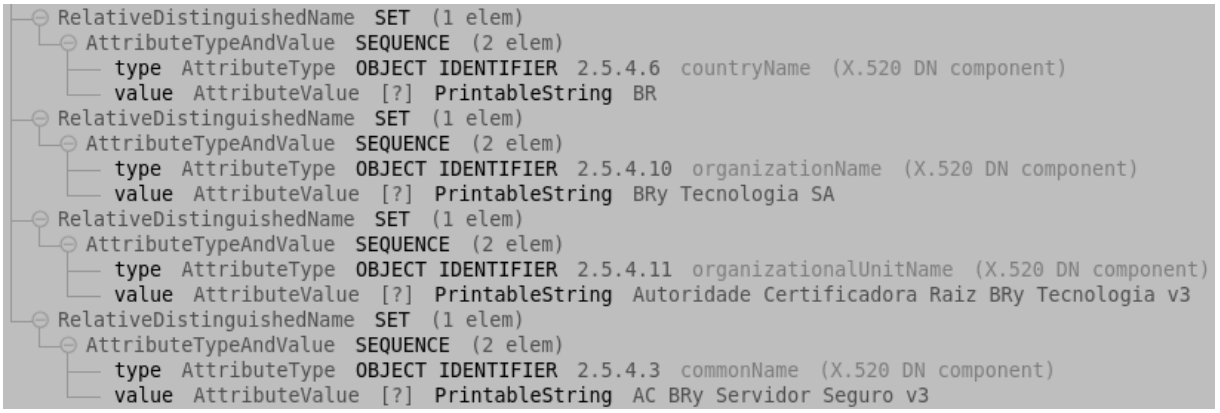
Assinatura

CURL realizado para assinatura do arquivo doc.txt:

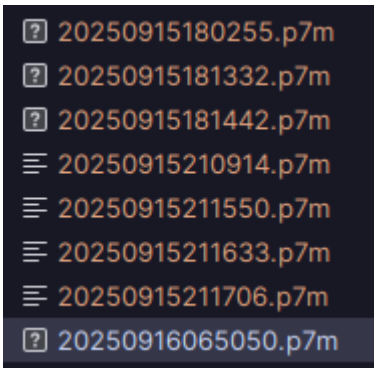
*curl -X POST http://localhost:8080/signature -F "file=@./src/main/resources/files/doc.txt" -F "pkcs12=@./src/main/resources/keys/certificado_teste_hub.pfx" -H "X-password: *********"*

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 “20250916131959.p7m” que acabou de ser gerado pelo passo anterior:

http://localhost:8080/verify -F "file=@./20250916131959.p7m"

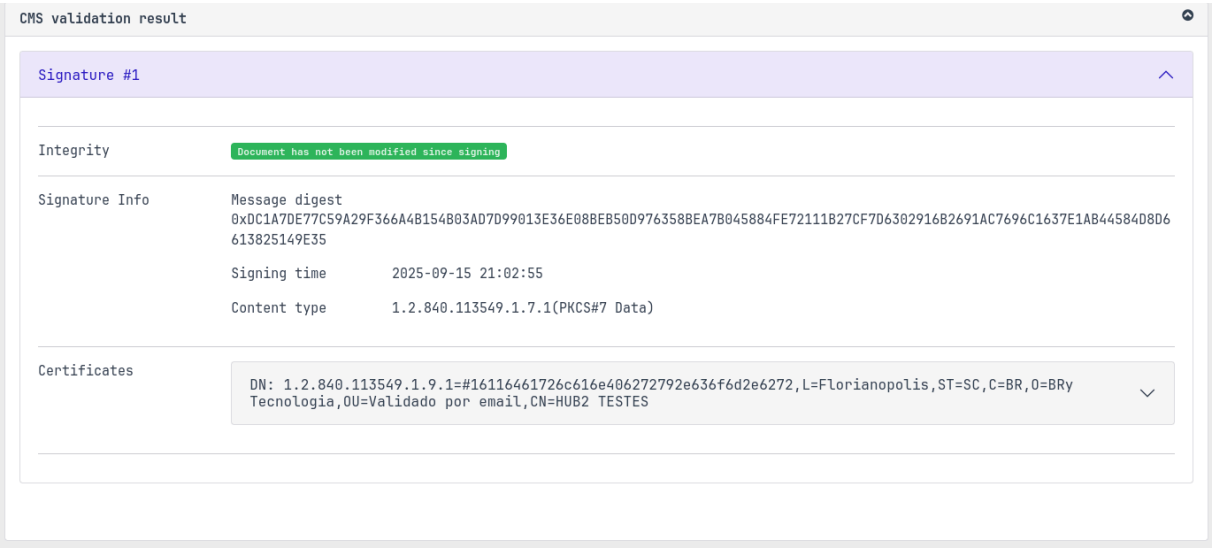
Resultado:


```
{
    "originalData": "546573746520766167612062616 36b2d656e64204a617661",
    "status": "VALIDO",
    "signinTimeDate": "Tue Sep 16 13:19:59 GMT-03:00 2025",
    "encapContentInfoHash": "dc1a7de77c59a29f366a4b154b03ad7d99013e36e08beb50d976358bea7b045884fe72111b27cf7d6302916b2691ac7696c1637e1ab44584d8d6613825149e35",
    "digestAlgorithm": "SHA512",
    "cnsignerName": "CN=HUB2 TESTES,OU=Validado por email,O=BRy Tecnologia,C=BR,ST=SC,L=Florianopolis,E=darlan@bry.com.br,"
}
```


Alternativamente, em texto:

{
"originalData": "546573746520766167612062616 36b2d656e64204a617661",
"status": "VALIDO",
"signinTimeDate": "Tue Sep 16 13:19:59 GMT-03:00 2025",
"encapContentInfoHash": "dc1a7de77c59a29f366a4b154b03ad7d99013e36e08beb50d976358bea7b045884fe72111b27cf7d6302916b2691ac7696c1637e1ab44584d8d6613825149e35",
"digestAlgorithm": "SHA512",
"cnsignerName": "CN=HUB2 TESTES,OU=Validado por email,O=BRy Tecnologia,C=BR,ST=SC,L=Florianopolis,E=darlan@bry.com.br,"
}

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

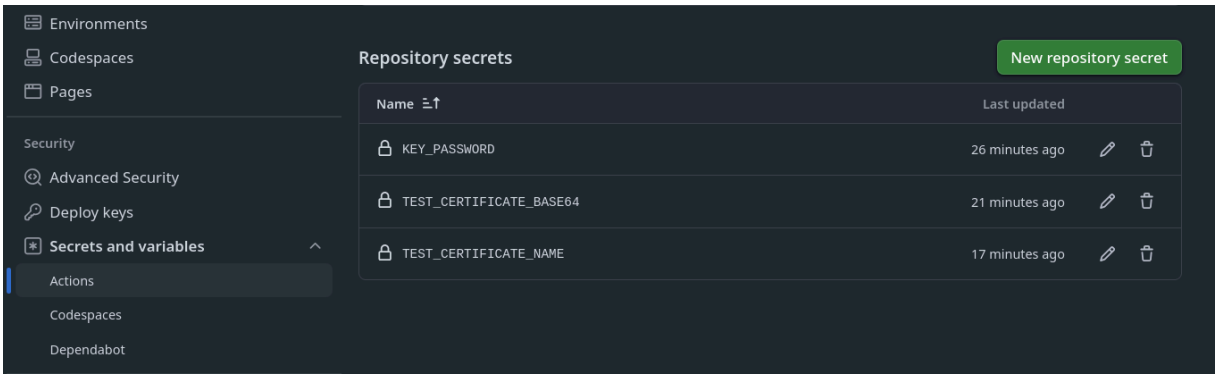


Distribuição de Código:

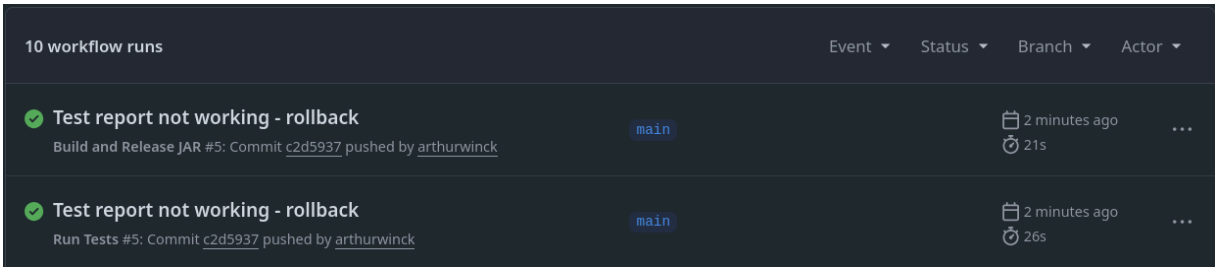
Para conseguirmos executar os testes de integração que foram implementados anteriormente, tivemos que fazer algumas alterações para que os testes busquem o certificado por meio de um *resource* no classpath (estando disponível na pasta resources). Porém, não podemos *commitar* tais arquivos, e para isso, criamos *secrets* (ou variáveis de ambiente “escondidas”) para a codificação Base64 do arquivo do certificado, para o nome do certificado codificado e também para a senha da chave privada que o acompanha.

Nessa situação, utilizamos os *secrets* de repositório, porém como melhoria, seria interessante criar as variáveis por *ambiente*, podendo ter variáveis específicas para ambientes de desenvolvimento e produção.

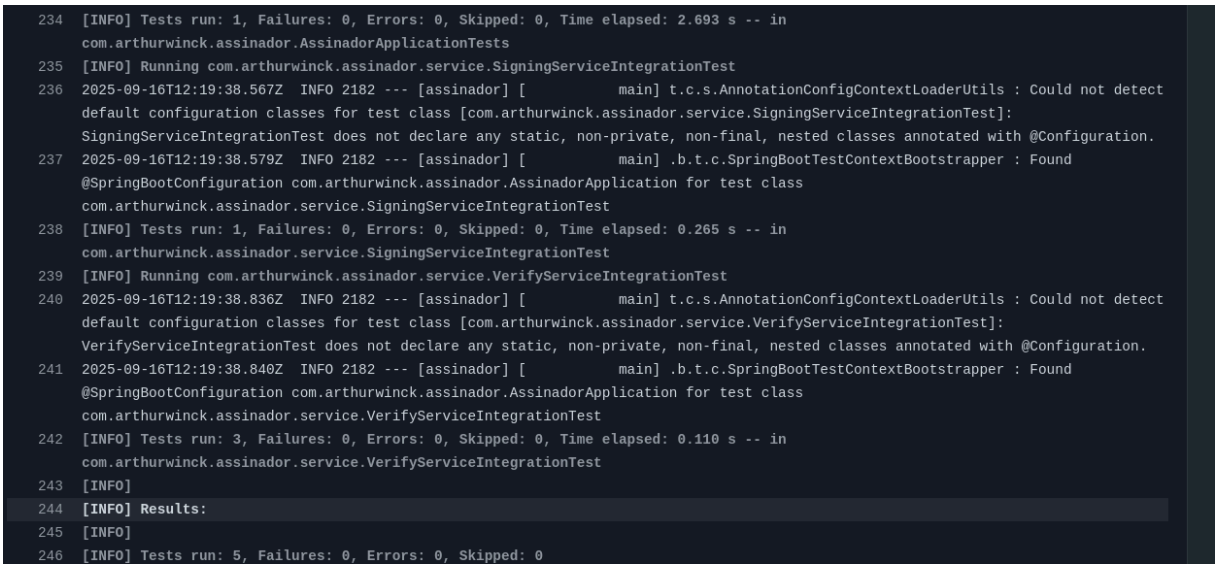
Outro ponto é o disparo desses workflows a partir da criação de uma tag, pois atualmente qualquer commit na main dispara os dois fluxos.



Com isso é possível ver o resultado dos testes na etapa “Run Tests” e “Build and Release JAR“ do workflow do repositório:

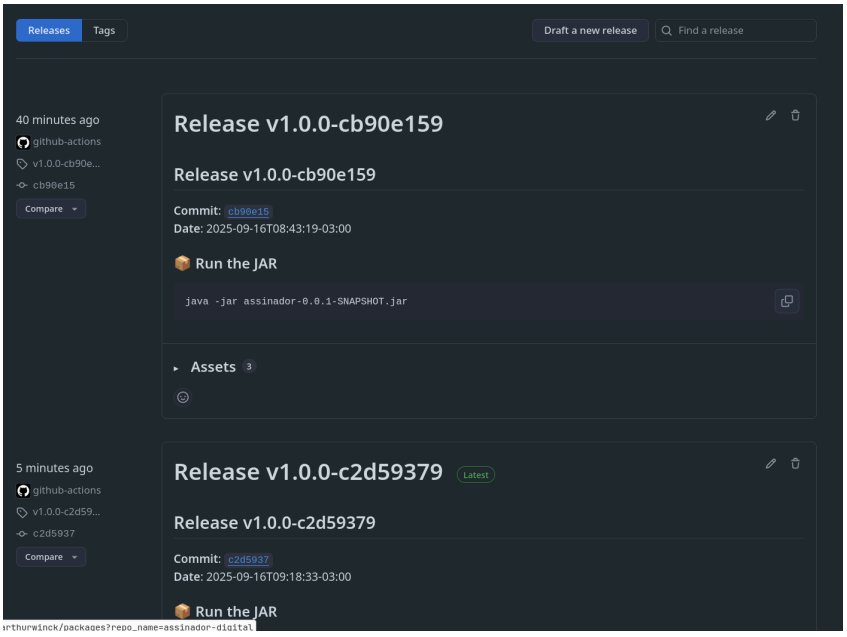


Exemplo de execução dos testes dentro do workflow:

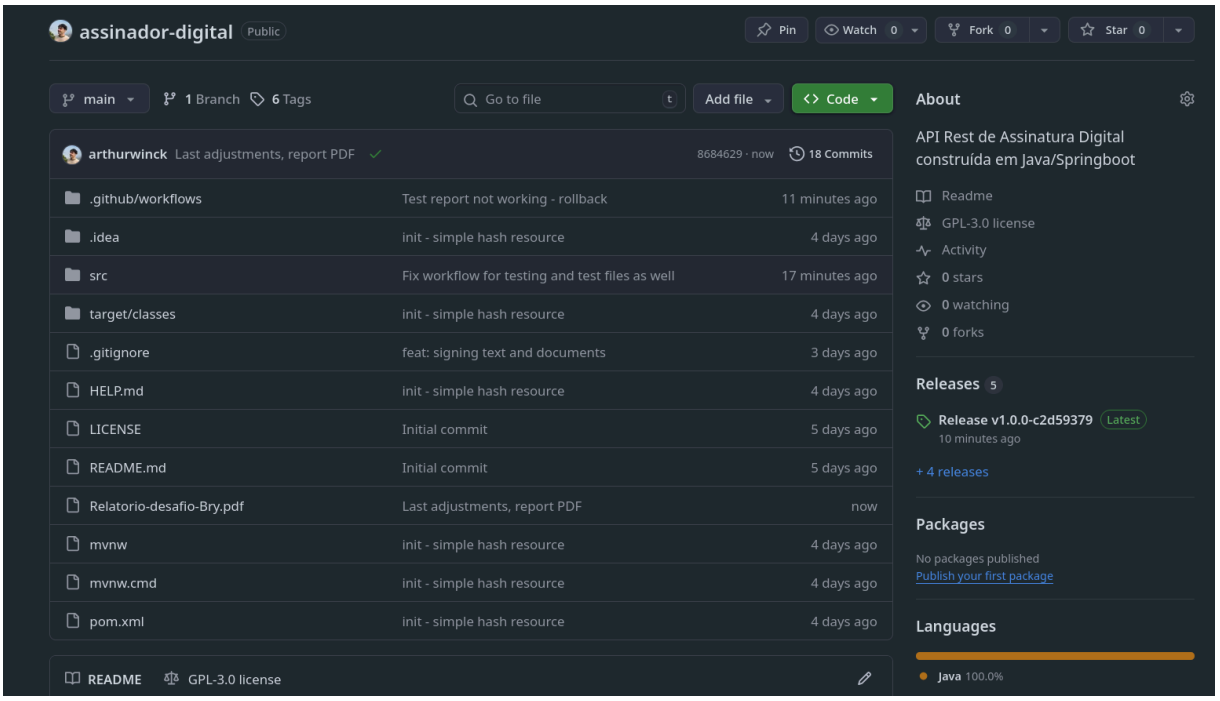


```
234 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.693 s -- in
    com.arthurwinck.assinador.AssinadorApplicationTests
235 [INFO] Running com.arthurwinck.assinador.service.SigningServiceIntegrationTest
236 2025-09-16T12:19:38.567Z  INFO 2182 --- [assinador] [           main] t.c.s.AnnotationConfigContextLoaderUtils : Could not detect
    default configuration classes for test class [com.arthurwinck.assinador.service.SigningServiceIntegrationTest]:
    SigningServiceIntegrationTest does not declare any static, non-private, non-final, nested classes annotated with @Configuration.
237 2025-09-16T12:19:38.579Z  INFO 2182 --- [assinador] [           main] .b.t.c.SpringBootTestContextBootstrapper : Found
    @SpringBootConfiguration com.arthurwinck.assinador.AssinadorApplication for test class
    com.arthurwinck.assinador.service.SigningServiceIntegrationTest
238 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.265 s -- in
    com.arthurwinck.assinador.service.SigningServiceIntegrationTest
239 [INFO] Running com.arthurwinck.assinador.service.VerifyServiceIntegrationTest
240 2025-09-16T12:19:38.836Z  INFO 2182 --- [assinador] [           main] t.c.s.AnnotationConfigContextLoaderUtils : Could not detect
    default configuration classes for test class [com.arthurwinck.assinador.service.VerifyServiceIntegrationTest]:
    VerifyServiceIntegrationTest does not declare any static, non-private, non-final, nested classes annotated with @Configuration.
241 2025-09-16T12:19:38.840Z  INFO 2182 --- [assinador] [           main] .b.t.c.SpringBootTestContextBootstrapper : Found
    @SpringBootConfiguration com.arthurwinck.assinador.AssinadorApplication for test class
    com.arthurwinck.assinador.service.VerifyServiceIntegrationTest
242 [INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.110 s -- in
    com.arthurwinck.assinador.service.VerifyServiceIntegrationTest
243 [INFO]
244 [INFO] Results:
245 [INFO]
246 [INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
```

Disponibilização das releases .JAR criadas pelo workflow de releases:



Todos os testes realizados estão disponíveis no [repositório Github](#). Muito obrigado!



Tratamento de erros e exceções

Exceções customizadas foram criadas para mapear situações específicas de erros. A seguir temos as definições de cada exceção. A exceção retornada ao Resource será sempre uma ***SigningValidationException*** ou uma ***VerifyValidationException*** para o serviço de assinatura e o serviço de verificação, respectivamente. Erros não mapeados serão transformados em ***GenericException***. Qualquer erro não mapeado retorna um status de 500, erro de servidor.s

- **SigningValidationException**
  - Exceção base disparada sempre que o serviço encontra um erro ao tentar realizar uma assinatura.

- **InvalidCertificateException (400 - Bad Request)**
  - Disparada sempre que não seja possível recuperar a chave privada a partir do arquivo enviado.

- **CMSException (Nativa BC) (500 - Server Error)**
  - Exceção nativa da biblioteca que ocorre quando não é possível adicionar certificados ao gerador de assinatura.

- **OperatorCreationException (Nativa BC) (500 - Server Error)**

    - Disparada quando não é possível instanciar estruturas necessárias para realizar assinatura
- **CertificateException (Nativa BC) (400 - Bad Request)**
    - Disparada em uma variedade de erros relacionados ao certificado.

- **KeyStoreException (Nativa Java Security) (500 - Server Error)**
    - Disparada quando keyStore não foi inicializado corretamente (erro ao tentar instanciar keystore a partir do arquivo enviado).

- **UnrecoverableKeyException (Nativa Java Security) (400 - Bad Request)**
    - Disparada quando a senha do keyStore está incorreta

- **VerifyValidationException**
    - Exceção base disparada quando o serviço de verificação de assinaturas encontra um erro.

- **InvalidSignedContentException (400 - Bad Request)**
    - Disparada quando conteúdo assinado não está presente no arquivo. Assinatura do tipo "*detached*" não funcionará pois o conteúdo original precisa estar anexado ao arquivo.

- **InvalidSignatureFileException (400 - Bad Request)**
    - Disparada quando não é possível instanciar o objeto de assinatura com conteúdo anexado a partir do arquivo de entrada ou quando algum dos certificados não se encontra na estrutura necessária.

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