Desafio prático BRy Tecnologia - Estágio desenvolvedor back-end Java

Classes do projeto

**main.control.CryptoController:** A classe que contém todas os métodos criados com o objetivo de manipular e criar assinaturas e também autenticar documentos com elas.

**main.control.FileRestController:** Classe que manipula os comandos e arquivos providos por meio das requests REST. Possui os dois endpoints /signature/ e /verify/ conforme solicitado no PDF.

**main.AssDigitalApplication:** A classe principal da aplicação, que apenas serve para inicializar o Spring Boot que por sua vez disponibiliza os endpoints para utilização.

Timeline

**Etapa 1 - Obtenção do resumo criptográfico** – Arquivo: hashed doc.txt

Essa etapa foi rapidamente concluída com a utilização de um web app que faz o hash em SHA256. No arquivo resultante consta o hash SHA256 em hexadecimal conforme solicitado.

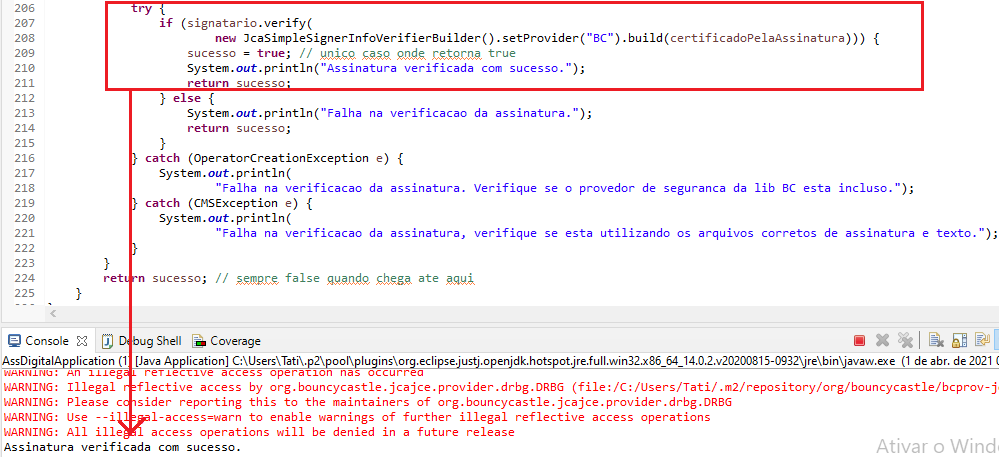
**Etapa 2 - Realizar uma assinatura digital** – Arquivos: certificado.txt, private key.txt, ArquivoAssinado.p7s

Essa etapa foi bem mais complicada (com vários erros de CertStore, até eu perceber que podia inicializar direto o JcaCertStore com X509Certificates de outro buffer). Primeiro, foi necessário entender o funcionamento das classes da lib BouncyCastle, bem como criar toda a estrutura de projeto e suas dependências – foi escolhido o Maven, e as dependências detalhadas no PDF foram adicionadas ao projeto. Após compreender mais ou menos o funcionamento das classes do BC, consegui extrair o certificado do alias provido e também a private key. Então, finalmente, após algumas tentativas, foi possível gerar o arquivo p7s com a assinatura digital.

**Etapa 3 - Verificar a assinatura gerada** – Arquivo: (Imagem abaixo)

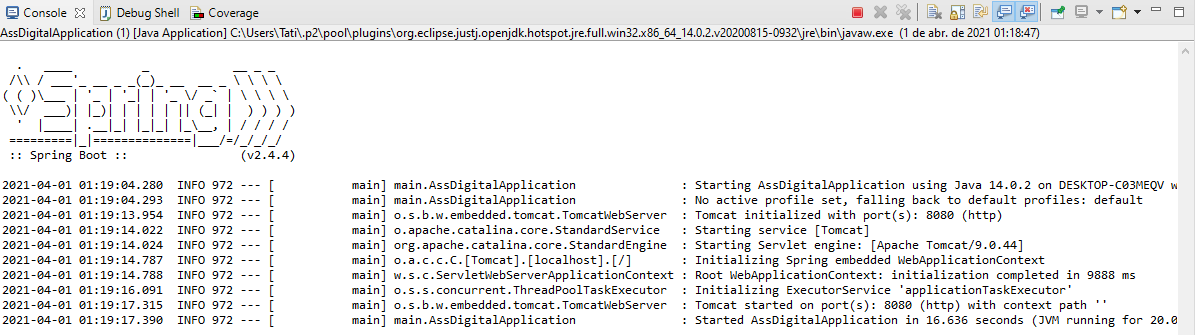
Fiquei um bom tempo tentando corrigir um “bug”, que era na verdade a ausência da declaração “Security.addProvider(new BouncyCastleProvider());” e seu respectivo import na hora de utilizar a BC.

Após corrigir este erro do provedor, foi possível pegar o certificado por meio da assinatura e em seguida, a verificação por meio do JcaSimpleSignerInfoVerifierBuilder. Na foto abaixo, confirmo a validação (retorno true) do próprio doc.txt provido utilizando a assinatura digital correspondente criada anteriormente (ArquivoAssinado.p7s).

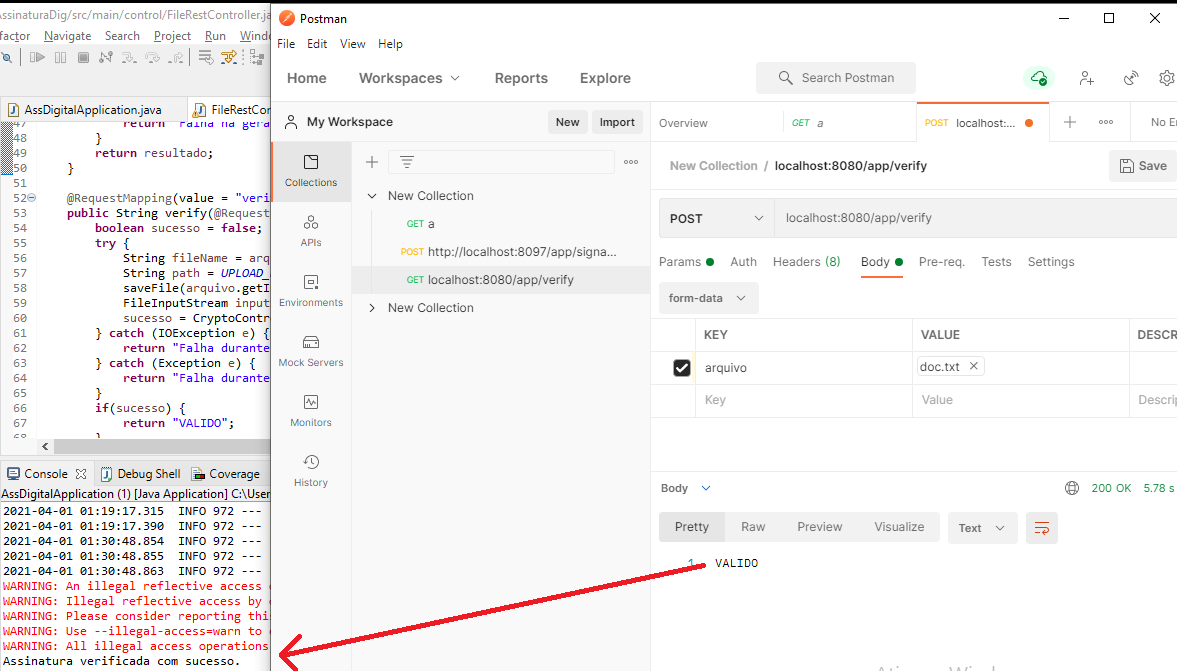
****

**Etapa 4 - API REST** – Arquivo: (Imagens abaixo)

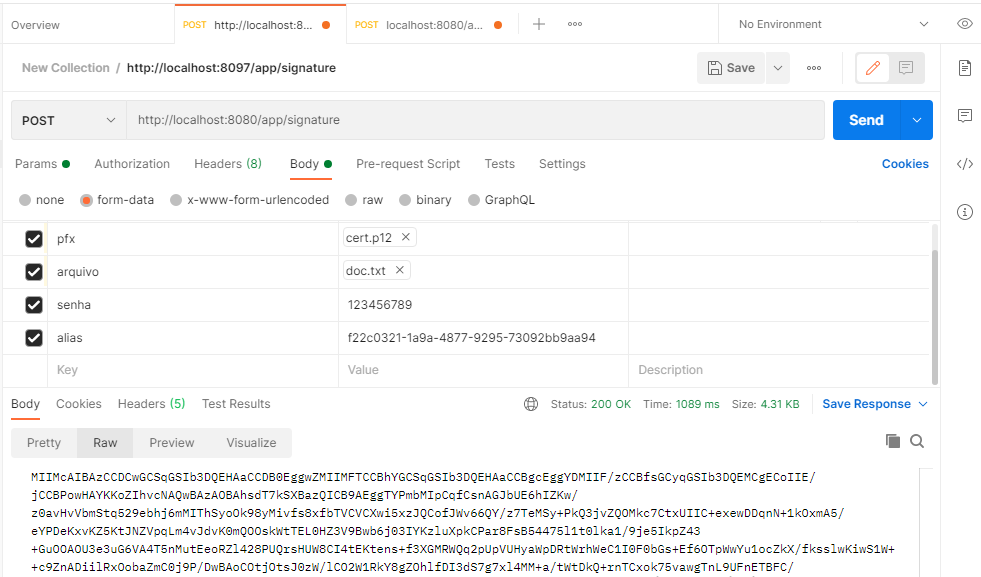
***First things first:*** Segue abaixo imagem comprovando o start do servidor sem erros.



Nesse momento, eu tive que fazer uma boa pesquisa e entender primeiro como funciona a utilização da API REST no Spring Boot (marinheiro de primeira viagem, apesar de já ser cliente antigo no assunto REST por já ter o utilizado em JS / Python). Confesso que fiquei um bom tempo imaginando como que faria aquilo sem interface gráfica, como que funcionaria tudo aquilo de receber o arquivo pela requisição – mas rapidamente percebi que nada disso era necessário e que era possível fazer tudo só utilizando a API REST do Spring Boot e um par de requests com as keys corretas no Postman. Consegui então, primeiro, fazer a verificação da assinatura por meio do arquivo original, pelo método “verify”. Fiz a validação do mesmo doc.txt utilizando a assinatura digital correspondente criada anteriormente (ArquivoAssinado.p7s). Abaixo, segue a imagem da resposta no Postman para a requisição abaixo:



Em seguida, depois de conseguir validar pela requisição, tinha chegado a hora de criar a assinatura também dessa forma. Segue abaixo a requisição e a resposta do procedimento de assinatura (cert.p12 é o **mesmo** arquivo de chaves provido pela BRy no teste, apenas encurtei o nome):



E com isso, encerro a participação no desafio, com todas as etapas concluídas com sucesso, agradecendo a empresa pela oportunidade de participar do mesmo. **Atenciosamente, Filipe Serena D’avila.**