

FCT-NOVA

Segurança de Redes e Sistemas de Computadores

Relatório

A Secure REST-Based Messaging Repository System with Mutual TLS Client/Server Authentication and Access Control

Autores

Jorge Pereira nº 49771

Rodrigo Lopes nº 50435

Professor:

Henrique João Lopes Domingos

Turno: P1

11 de Dezembro de 2019



Resumo

Este projecto tem como objectivo implementar um repositório de mensagens seguro, não só do ponto de vista de transporte de mensagens, onde estas, iram protegidas com cipher-suites descritas ao longo deste relatório, mas também contra ataques internos de possíveis funcionários honestos mas curiosos (Honest-but-curious), que com as nossas medidas e implementações de segurança, fará com que este tipo de ataques não sejam possíveis. Decidimos usar a framework Spring tanto para a implementação do servidor, como para a do cliente.

Conteúdo

1	Introdução	2
2	Modelo do Sistema e Arquitectura	3
2.1	Modelo do Sistema	3
2.2	Arquiterura	5
2.3	Modelo de Adversário	5
3	Detalhes da implementação	7
3.1	Especificação da API	9
3.1.1	API das mensagens	9
3.1.2	API dos utilizadores	9
3.2	Configurações de TLS	10
4	Validação e Avaliação da solução	11
4.1	Auditoria de Segurança ao servidor	11
5	Conclusão	15
A	Tabela de Sumário da implementação do trabalho TP2 submetido para avaliação	16

Introdução

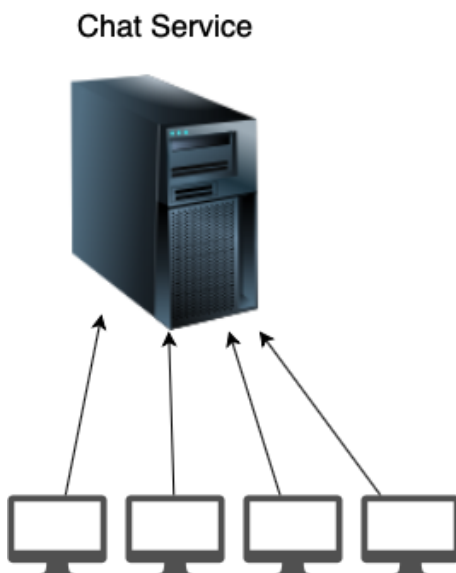
Este projecto terá como principais, um servidor, que será um repositório de mensagens cifradas pelos clientes de forma a ser impossível, a quem tenha acesso à base de dados, ver o conteúdo das mensagens. O servidor foi desenvolvido em Spring e com algumas modificações daquilo que seria a proposta inicial da API. Toda a parte criptográfica ficou no lado do cliente, sendo este o principal que cifra a mensagem e está encarregado de a decifrar e assinar de acordo com as suas configurações de segurança. Para o handshake inicial entre o servidor e o cliente, usamos TLS que utiliza as mesmas configurações (certificado e chaves) que são usadas para a cifragem das mensagens a enviar ao servidor, de modo a ser mais fácil o desenvolvimento do projecto. Por fim iremos apresentar a nossa pequena auditoria de segurança ao nosso servidor de modo a perceber e ter uma ideia prática das ferramentas e metodologias que são usadas nos dias de hoje para esse fim.

Modelo do Sistema e Arquitectura

2.1 Modelo do Sistema

Numa análise muito superficial do sistema, podemos visualizar o sistema como um repositório que contém uma dada interface para executar invocações remotas, como podemos visualizar na figura 2.1. Estas invocações permitem aos utilizadores enviarem mensagens para outros utilizadores, receberem mensagens, enviarem recibos de leitura, verificar o estado de uma mensagem, entre outras. O servidor serve apenas de repositório onde os utilizadores executam acções, semelhante a um sistema de publish/subscriber mas, que cada utilizador tem o seu tópico.

Figura 2.1: Vista superficial do sistema

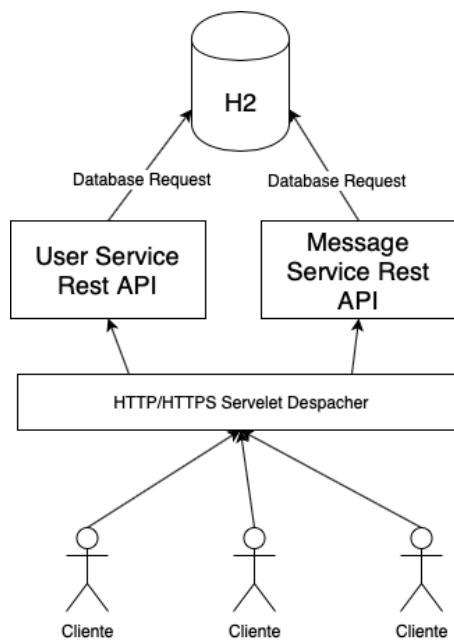


Esta interface para as operações remotas está implementada sobre *REST*. Contendo dois serviços (um para operações relacionadas com os utilizadores e outro para operações relacionadas com operações de mensagens). Como estamos a usar a *Spring framework* existe um *Http Servlet Dispatcher* que

seleccionará, dependendo do tipo do caminho do recurso, a qual serviço entregar para este posteriormente, responder. Para guardar de forma persistente as mensagens dos utilizadores e até mesmo os utilizadores, é usada uma base de dados SQL (H2).

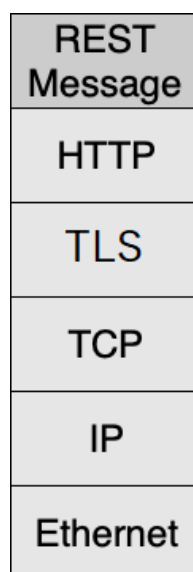
Vejamos a figura 2.2:

Figura 2.2: Vista interna do sistema



As comunicações entre o cliente e o servidor de forma a garantir confidencialidade, integridade e autenticidade dos dados passados são executadas sobre TLS. Vejamos o stack de protocolos usados presente na figura 2.3.

Figura 2.3: Stack de Protocolos usados



2.2 Arquiterura

Na nossa arquitectura foi utilizada a Spring Framework, onde implementamos a nossa API para o nosso Server Repository. Na API dividimos os nossos serviços em dois, o das mensagens e o dos utilizadores que serão especificados mais detalhe no ponto 3.1. Para a base de dados, decidimos utilizar H2 de forma a facilitar a manipulação e visualização dos dados numa maneira rápida e intuitiva.

Também para o cliente, utilizámos a Spring Shell, que nos forneceu facilidade no desenvolvimento do nosso cliente, tendo sido uma excelente aprendizagem neste tipo de tecnologias.

2.3 Modelo de Adversário

O servidor e o cliente em cooperação deverão respeitar o seguinte modelo de adversário:

- Confidencialidade de Mensagens; Integridade e autenticação das mensagens, Escuda das mensagens passadas na rede; Modificação e introdução de mensagens;

Também deverá ser garantida a não repudição de um recibo de leitura tal como a autenticidade da mesma e a confidencialidade das mensagens durante todo o seu tempo de vida.

Para conseguir suportar os requisitos desde modelo foram usadas diversas estratégias para as diferentes fases do ciclo de vida de uma mensagem.

Durante a transmissão:

Durante a transmissão será necessário garantir a confidencialidade, integridade, a escuda e a modificação ou introdução de mensagens. Para isso foi usado o protocolo de TLS que irá criar um tunel seguro entre o cliente e o servidor. Um atacante acima do protocolo *TLS* não irá conseguir visualizar que tipo de tráfego está a passar na rede. Qualquer tentativa de introdução ou alteração de mensagens será detectado pelo *TLS* e alertado, caso seja grave, para o termino da comunicação *TLS* corrente.

Guardada no repositório e posterior envio para cliente:

De forma a garantir a confidencialidade das mensagens quando estão guardadas no repositório do servidor, é necessário que estas estejam a usar um envelope para esconder o seu conteúdo e garantir a sua integridade contra administradores de sistema que possam ter acesso a estas mensagens (agentes curiosos). Este envelope também ajudará a defender contra ataques que um cliente tente passar por outro e obter as mensagens deste.

A estrutura do envelope é a seguinte para o receptor:

$$\{ks\}_{k_{pubReceptor}} || \{M\}_{ks} || DigitalSignature_{K_{pubSender}}$$

Quem envia a mensagem também precisará de ter uma estrutura semelhante de modo a que possa mais tarde saber o conteúdo enviado. Vejamos o envelope:

$$\{ks_2\}_{k_{pubSender}} || \{M\}_{ks_2} || DigitalSignature_{K_{pubSender}}$$

Envio prova de leitura da mensagem:

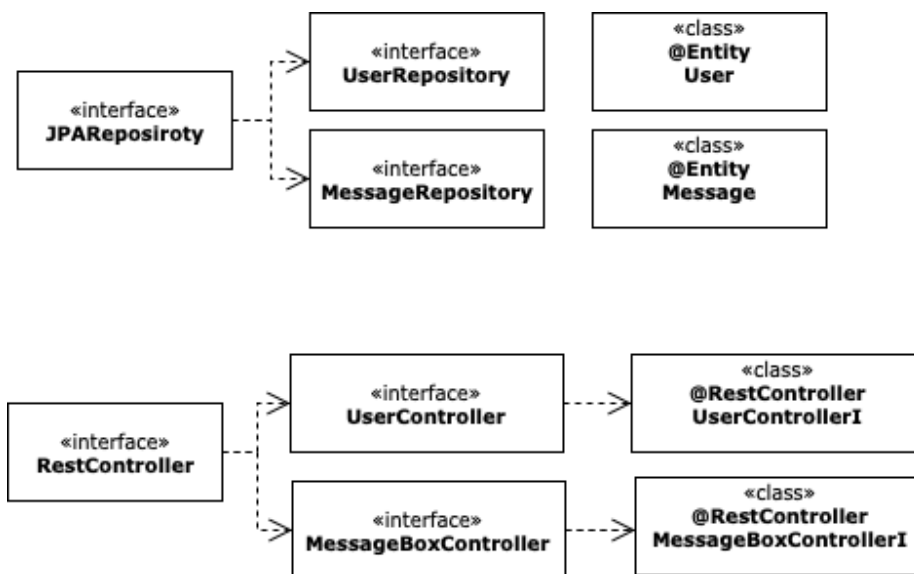
$$DigitalSignature_{e_{k_{pubReceptor}}}(ReceptorEnvelope)$$

Detalhes da implementação

Como foi dito anteriormente, foi usado *Spring* com o modulo *Spring Boot* para a implementação do servidor e do cliente, assim para construir software que respeite as construções usadas nessa *framework*, foi necessário usar o padrão de desenho MVC (Model View Controller). Assim foi necessário dividir as componentes do trabalho em Model's (User,Message) , Controller's (UserController , MessageBoxController) o View não está presente mas se fizermos analogia seria a linha de comandos do cliente. Também foi necessário oferecer componentes de persistência de dados para comunicarem com a base de dados escolhida. Estes componentes são os Repositórios (UserRepository, MessageRepository).

Utilizamos uma *in-memory database* para ser só necessário instanciar um serviço no sistema de Cloud da Microsoft (o Azure).

Figura 3.1: Componentes



No modo geral todos os modelos são convertidos JSON e colocados no payload de uma mensagens HTTP. Vejamos os campos de cada modelo de dados.

Uma mensagem:

- **id** - O ID representa um número único na base de dados de cada mensagem
- **from** - ID do utilizador que envia a mensagem
- **to** - ID do destino da mensagem, ou seja do receptor
- **messageFrom** - Conteúdo da mensagem para o remetente (Envelope cifrado)
- **messageTo** - Conteúdo da mensagem para o receptor (Envelope cifrado)
- **sendDate** - Data de envio da mensagem (data de recepção por parte do servidor)
- **receivedDate** - Data de recepção da mensagem (quando o receptor envia recibo de leitura)
- **signature** - Recibo de leitura por parte do receptor. (Assinatura digital do messageTo com a sua chave privada)
- **parameters** - Parâmetros de segurança usados (SAP - Security Association Parameters)

Exemplo de uma representação em JSON da mensagem:

```
1 {
2   "id": "123",
3   "from": "10",
4   "to": "15",
5   "messageFrom": "0cQ3CrNyuuKBdLL1mV+gAAABNgOMNKEYmnKEyyYnySF0K
6     00IXr|RkehQZTbay0JW...",
7   "messageTo": "o9IQKnLdwyE0yzs0WQ4xgAAABMcgC6xKivMm/bkiXM+/4
8     NTXmXK|AHJhGMHHBV+GZUF...",
9   "receivedDate": null,
10  "signature": null,
11  "parameters": ["ASSYM": "AES", "ASSYMKEY": "128", ...]
12 }
```

Um Utilizador:

- **id** - Id único gerado pelo servidor
- **uuid** - Identificador único de utilizador (Hash da chave publica)
- **secdata** - Chave publica do utilizador

```

1 {
2   "id": "1",
3   "uuid": "udack9TDHr5jVFne6cfjVGlBx0WqrF5FFSKAxVWj0Xw\u003d",
4   "secdata": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAn+DzG5
5     ATsJRHwiiFFdMzngtzns..."

```

3.1 Especificação da API

Os endpoints do nosso servidor estão divididos em dois tipos de serviços:

- **API para as mensagens**
- **API para os utilizadores**

3.1.1 API das mensagens

Aqui fica toda a logística e tratamento das mensagens guardadas no nosso servidor. O tipo de funções que este serviço tem, são os listados no enunciado, como:

- Listar as mensagens ainda não lidas pelo utilizador (NEWS).
- Listar todas as mensagens de um utilizador (ALL).
- Enviar uma nova mensagem (SEND).
- Receber uma mensagem (RECV).
- Assinar uma mensagem de forma a registar a sua leitura (RECEIPT).
- Ver o estado de uma mensagem (STATUS).

3.1.2 API dos utilizadores

Neste serviço estão implementadas as funcionalidades dedicadas aos utilizadores, tais como:

- Criar um novo utilizador no sistema (CREATE).
- Listar todos os utilizadores inseridos no sistema (LIST).

3.2 Configurações de TLS

Para configuração do TLS poderá alterar-se as propriedades do ficheiro *application.properties*:

server.port= 8443 - indica a porta a que o servidor está ligado.

server.ssl.enabled= true - activar ssl/tls

server.ssl.key-store= test.jks - keychain com chave privada e chain de certificados do cliente

server.ssl.key-password= password - password da chave privada

server.ssl.key-store-password= password - password da keystore

server.ssl.client-auth= need - caso seja necessário autentificação do cliente.

server.ssl.enabled-Protocols: [TLSv1.2] - protocolos aceites de tls pelo servidor

server.ssl.ciphers:"TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256"

Validação e Avaliação da solução

4.1 Auditoria de Segurança ao servidor

Para tornar o nosso relatório um pouco mais completo, decidimos adicionar este capítulo, que basicamente serve para testar-mos algumas ferramentas que testam e encontram possíveis vulnerabilidades ao nosso servidor, e assim conseguirmos, mais rapidamente, encontrar alguns erros para mais tarde podermos corrigi-los. Por falta de tempo não nos foi possível melhorar alguns erros dados como críticos, mas sem dúvida que entendê-los e perceber-los foi uma importante aprendizagem.

Iremos começar por mostrar o início da nossa auditoria, onde corremos uma ferramenta que verifica os serviços de um dado servidor e os seus suportes de cifras TLS/SSL, protocolos e possíveis falhas e vulnerabilidades criptográficas.

Figura 4.1: testssl.sh localhost:8443

```
root@bufferoot:~/Desktop/testssl.sh-3.0rc5# ./testssl.sh localhost:8443
#####
testssl.sh      3.0rc5 from https://testssl.sh/dev/

This program is free software. Distribution and
modification under GPLv2 permitted.
USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

Please file bugs @ https://testssl.sh/bugs/
#####

Using "OpenSSL 1.0.2-chacha (1.0.2k-dev)" [~179 ciphers]
on bufferoot:./bin/openssl.Linux.x86_64
(built: "Jan 18 17:12:17 2019", platform: "linux-x86_64")

Start 2019-12-01 14:32:25 -->> 127.0.0.1:8443 (localhost) <<--
A record via:      /etc/hosts
rDNS (127.0.0.1):  localhost.
Service detected:  HTTP
```

Como podemos ver na figura seguinte, o nosso servidor oferece TLS 1 a TLS 1.2, não oferecendo SSLv2 ou SSLv3 do que no ponto de vista deste teste é bom.

Testing Robust (perfect) forward secrecy /TODO/

Figura 4.2: Teste de protocolos e cifras.

```
Testing protocols via sockets except NPN+ALPN

SSLv2      not offered (OK)
SSLv3      not offered (OK)
TLS 1      offered
TLS 1.1    offered
TLS 1.2    offered (OK)
TLS 1.3    not offered -- connection failed rather than downgrading to TLSv1.2
NPN/SPDY   not offered
ALPN/HTTP2 not offered

Testing cipher categories

NULL ciphers (no encryption)      not offered (OK)
Anonymous NULL Ciphers (no authentication) not offered (OK)
Export ciphers (w/o ADH+NULL)     not offered (OK)
LOW: 64 Bit + DES, RC[2,4] (w/o export) not offered (OK)
Triple DES ciphers / IDEA        not offered (OK)
Average: SEED + 128+256 Bit CBC ciphers offered
Strong encryption (AEAD ciphers)  offered (OK)

Testing robust (perfect) forward secrecy, (P)FS -- omitting Null Authentication/Encryption, 3DES, RC4

No ciphers supporting Forward Secrecy offered
```

Aqui podemos ver que para todos os protocolos aceites pelo servidor, têm como cipher suits "preferidas" Diffie Hellman, do que do ponto de vista de segurança está correto e seguro para este teste.

Figura 4.3: preferencias do servidor (cipher suits).

```
Testing server preferences

Has server cipher order?      yes (OK)
Negotiated protocol           TLSv1.2
Negotiated cipher             DHE-DSS-AES256-GCM-SHA384, 1024 bit DH
Cipher order
  TLSv1:      DHE-DSS-AES256-SHA DHE-DSS-AES128-SHA
  TLSv1.1:    DHE-DSS-AES256-SHA DHE-DSS-AES128-SHA
  TLSv1.2:    DHE-DSS-AES256-GCM-SHA384 DHE-DSS-AES256-SHA256
              DHE-DSS-AES256-SHA DHE-DSS-AES128-GCM-SHA256
              DHE-DSS-AES128-SHA256 DHE-DSS-AES128-SHA
```

Com este teste reparamos que alguns parâmetros nos certificados que geramos, estavam em falta e outros "inseguros", o que rapidamente entendemos o porquê, pois dada a maneira como geramos os certificados era perfeitamente normal este tipos de "vermelhos". Alguns destes erros podem ser resolvidos utilizando a ferramenta openssl com o seguinte comando fornecido pelo professor:

```
$ openssl req
  -new
  -sha256
  -key domain.key
  -subj "/C=US/ST=CA/O=Acme, Inc./CN=example.com"
  -reqexts SAN -config <(cat/etc/ssl/openssl.cnf
```

```
<(printf"\n[SAN]\nsubjectAltName=DNS:example.com,DNS:www.example.com" ) )
-out domain.csr
```

Figura 4.4: Teste Server Hello.

```
Testing server defaults (Server Hello)

TLS extensions (standard)      "renegotiation info/#65281"
                                "extended master secret/#23"
Session Ticket RFC 5077 hint  no -- no lifetime advertised
SSL Session ID support         yes
Session Resumption             Tickets no, ID: no
TLS clock skew                 -1 sec from localtime
Signature Algorithm            DSA with SHA256
Server key size                DSA 2048 bits
Server key usage               --
Server extended key usage      --
Serial / Fingerprints          3D3069FB / SHA1 886A2F3F87D45B410B774F8DB14ED11FFE30C2E6
                                SHA256 BB0F450846EF0C53313C557867CB8877A5BA0093895588FFA7510770EA788B01
Common Name (CN)              Root CA
subjectAltName (SAN)          missing (NOT ok) -- Browsers are complaining
Issuer                         self-signed (NOT ok)
Trust (hostname)              certificate does not match supplied URI (same w/o SNI)
Chain of trust                 NOT ok (self signed)
EV cert (experimental)        no
"eTLS" (visibility info)      not present
Certificate Validity (UTC)     expires < 60 days (55) (2019-10-28 10:34 --> 2020-01-26 10:34)
# of certificates provided     1
Certificate Revocation List    --
OCSP URI                      --
                                NOT ok -- neither CRL nor OCSP URI provided
OCSP stapling                 not offered
OCSP must staple extension     --
DNS CAA RR (experimental)     not offered
Certificate Transparency       --
```

No teste de vulnerabilidades, podemos notar que praticamente todos os testes deram "luz verde", excepto um, LOGJAM, que é uma vulnerabilidade contra a troca de chaves usando Diffie Hellman. Para resolver este problema, teríamos que perceber melhor como este tipo de ataque funciona e tentar encontrar uma solução.

Figura 4.5: Testes de vulnerabilidades.

```
Testing vulnerabilities

Heartbleed (CVE-2014-0160)      not vulnerable (OK), no heartbeat extension
CCS (CVE-2014-0224)            not vulnerable (OK)
Ticketbleed (CVE-2016-9244), experiment. not vulnerable (OK), no session ticket extension
ROBOT                          Server does not support any cipher suites that use RSA key transport
Secure Renegotiation (CVE-2009-3555) not vulnerable (OK)
Secure Client-Initiated Renegotiation likely not vulnerable (OK), timed out
CRIME, TLS (CVE-2012-4929)     not vulnerable (OK)
BREACH (CVE-2013-3587)         no HTTP compression (OK) - only supplied "/" tested
POODLE, SSL (CVE-2014-3566)    not vulnerable (OK)
TLS_FALLBACK_SCSV (RFC 7507)   Downgrade attack prevention NOT supported
SWEET32 (CVE-2016-2183, CVE-2016-6329) not vulnerable (OK)
FREAK (CVE-2015-0204)          not vulnerable (OK)
DROWN (CVE-2016-0800, CVE-2016-0703) not vulnerable on this host and port (OK)
                                no RSA certificate, thus certificate can't be used with SSLv2 elsewhere
LOGJAM (CVE-2015-4000), experimental VULNERABLE (NOT ok): common prime: RFC2409/Oakley Group 2 (1024 bits),
                                but no DH EXPORT ciphers
BEAST (CVE-2011-3389)          TLS1: DHE-DSS-AES256-SHA
                                DHE-DSS-AES128-SHA
                                VULNERABLE -- but also supports higher protocols TLSv1.1 TLSv1.2 (likely mitigated)
LUCKY13 (CVE-2013-0169), experimental potentially VULNERABLE, uses cipher block chaining (CBC) ciphers with TLS. Check patches
RC4 (CVE-2013-2566, CVE-2015-2808) no RC4 ciphers detected (OK)
```

Na imagem seguinte estão apenas alguns testes fornecidos por este script, para ver-mos quais as cipher suits acordadas por cada tipo de plataforma. Sendo possível ver que todas as plataformas que conseguiram estabelecer conexão com o nosso servidor, acordaram em usar Diffie Hellman.

Figura 4.6: Simulação de vários tipos de clientes.

```
Running client simulations (HTTP) via sockets

Android 4.2.2          TLSv1.0 DHE-DSS-AES256-SHA, 1024 bit DH
Android 4.4.2          TLSv1.2 DHE-DSS-AES256-GCM-SHA384, 1024 bit DH
Android 5.0.0          TLSv1.2 DHE-DSS-AES256-SHA, 1024 bit DH
Android 6.0            No connection
Android 7.0            No connection
Android 8.1 (native)   No connection
Android 9.0 (native)   No connection
Chrome 65 Win 7        No connection
Chrome 74 (Win 10)     No connection
Firefox 62 Win 7       No connection
Firefox 66 (Win 8.1/10) No connection
IE 6 XP                No connection
IE 7 Vista             TLSv1.0 DHE-DSS-AES128-SHA, 1024 bit DH
IE 8 Win 7             TLSv1.0 DHE-DSS-AES128-SHA, 1024 bit DH
IE 8 XP                No connection
IE 11 Win 7            TLSv1.2 DHE-DSS-AES256-SHA256, 1024 bit DH
IE 11 Win 8.1          TLSv1.2 DHE-DSS-AES256-SHA256, 1024 bit DH
IE 11 Win Phone 8.1    TLSv1.2 DHE-DSS-AES128-SHA256, 1024 bit DH
IE 11 Win 10           TLSv1.2 DHE-DSS-AES256-SHA256, 1024 bit DH
Edge 15 Win 10         No connection
Edge 17 (Win 10)       No connection
Opera 60 (Win 10)      No connection
Safari 9 iOS 9         No connection
Safari 9 OS X 10.11    No connection
Safari 10 OS X 10.12   No connection
Apple ATS 9 iOS 9      No connection
Tor 17.0.9 Win 7       TLSv1.0 DHE-DSS-AES256-SHA, 1024 bit DH
Java 6u45              TLSv1.0 DHE-DSS-AES128-SHA, 1024 bit DH
Java 7u25              TLSv1.0 DHE-DSS-AES128-SHA, 1024 bit DH
Java 8u161             TLSv1.2 DHE-DSS-AES256-SHA256, 1024 bit DH
Java 9.0.4             TLSv1.2 DHE-DSS-AES256-GCM-SHA384, 1024 bit DH
OpenSSL 1.0.1l          TLSv1.2 DHE-DSS-AES256-GCM-SHA384, 1024 bit DH
OpenSSL 1.0.2e          TLSv1.2 DHE-DSS-AES256-GCM-SHA384, 1024 bit DH
OpenSSL 1.1.0j (Debian) No connection
OpenSSL 1.1.1b (Debian) No connection
Thunderbird (60.6)     No connection

Done 2019-12-01 14:33:17 [ 54s] --> 127.0.0.1:8443 (localhost) <<--
```

Use this section to discuss the validation and correctness of your designed system and related implementation (prototype). Explain the experimental evaluation done, the considered, focused and observed evaluation criteria. Discuss how you evaluated or measured your system for those criteria (referring experiments, practical observations/deployment and possible qualitative, as well as, quantitative metrics you're your performed observations). If you want to structure more clearly the section, can use an initial paragraph describing the evaluation and validation objectives as addressed, dedicating a sub-section to specific observations done and argumentation about the system validity from your observations.

Conclusão

Em suma, este projecto revelou ser quase um full stack project, onde desenvolvemos um serviço seguro que nos obrigou a aprender um pouco mais não só sobre o tema da cadeira (segurança) mas também sobre desenvolvimento de uma API, usando a Spring Framework, manipulação de base de dados SQL, usando Spring com H2, e por fim, ainda descobrimos dentro da Spring Framework, a existência da Spring Shell que nos facilitou imenso na construção do nosso cliente.

Em termos dos objectivos da cadeira, foram todos os requisitos propostos, concluídos, dentro do tempo estimado. Por motivos de falta de tempo, não nos foi possível desenvolver um front-end mais apelativo para o cliente, mas a nosso ver o nosso shell client vai de acordo com o pretendido.



**Tabela de Sumário da implementação do
trabalho TP2 submetido para avaliação**