FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Informática

Mestrado Integrado em Engenharia Informática
Sistemas Distribuídos – 1º teste, 13 de Abril de 2016 - versão A
1º Semestre, 2015/2016

NOTAS: Leia as questões atentamente antes de responder. **O teste é sem consulta. A duração do teste é 1h30min**. O teste contém **7** páginas.

Nome: _____ **Número**: _____

1) Considere o código de um servidor de ficheiros REST muito simples no Anexo A, o qual foi escrito recorrendo à biblioteca Jersey como foi realizado no primeiro projeto de Sistemas Distribuídos. O servidor suporta duas funcionalidades: i) criar um novo ficheiro recebendo o nome do mesmo no URL; e ii) listar todos os ficheiros que estão guardados no servidor.

   a) Considerando a listagem de anotações que se encontram no final do Anexo A, complete o código fornecido do servidor, como achar necessário, sabendo que são usados os seguintes URLs para criar o ficheiro a.txt e listar os ficheiros do servidor, para um servidor a executar na máquina local no porto 9090.

      i) http://localhost:9090/simpleFileServer/create/a.txt

      ii) http://localhost:9090/simpleFileServer/list

   b) No Anexo B apresenta-se o código do cliente que lista todos os ficheiros presentes no servidor. Ao executar o cliente foi gerada a exceção apresentada no mesmo anexo. Faça, no Anexo B, as correções necessárias para que o cliente execute com sucesso.

   c) Suponha que se pretende fazer uma versão deste servidor que funcione em SOAP. No Anexo C.1 apresente o código do servidor – só a assinatura da classe e dos métodos. No Anexo C.2 apresente uma versão do cliente, apresentado no Anexo B, que funcione em SOAP, sabendo que o servidor está no endereço "s.unl.pt:9000".

2) Indique se cada afirmação é **[V]erdadeira** ou **[F]alsa** (**nota: respostas incorretas descontam**):

   ___ Diz-se que um sistema garante a integridade dos dados quando um atacante não pode aceder aos dados.

   ___ Num sistema distribuído aberto devem ser conhecidas as interfaces e os protocolos de comunicação entre os clientes e os servidores.

   ___ Um exemplo de falha de omissão é a receção de uma mensagem cujo conteúdo é diferente do enviado pelo emissor.

   ___ Um sistema cliente/servidor particionado cada servidor recebe em média um maior número de pedidos que num sistema cliente/servidor replicado.

   ___ O sistema BitTorrent original combina um modelo cliente/servidor com um modelo peer-to-peer.

   ___ Num sistema peer-to-peer estruturado, o número de nós pelos quais passa um pedido de lookup depende do número de ligações que cada nós tiver para outros nós.

   ___ Um sistema peer-to-peer não estruturado é particularmente indicado para ambientes com filiação (*membership*) com elevado dinamismo e pesquisas pouco frequentes?

___ Num sistema de comunicação síncrono, quando no emissor termina o envio de uma mensagem, sabe-se que o recetor já recebeu a mensagem.

___ A disponibilidade mede a fração do tempo que um sistema está a funcionar corretamente.

No contexto da invocação remota de métodos/procedimentos:

___ Os *web services* (SOAP), por omissão, implementam uma semântica de invocação "*at most once*".

___ No .NET remoting, além de ser possível invocar métodos é possível aceder a atributos (variáveis) do servidor.

___ No Java RMI todos os métodos do servidor devem lançar RemoteException como forma de assinalar erros de comunicação.

___ O mecanismo de codificação dos dados ProtoBuf é tipicamente mais eficiente do que a serialização do Java em termos de espaço e tempo de processamento.

___ Nos web services SOAP, o URL usado para enviar a mensagem para invocar um método permite identificar claramente qual o método a invocar.

___ No suporte Java para SOAP, é necessário recriar o stub do cliente usando o wsimport sempre que se altera o código do servidor, mesmo que a assinatura dos métodos se mantenha inalterada.

___ No REST, quando se usa o método POST para criar um novo recurso, o conteúdo do recurso é enviado codificado no URL do pedido HTTP.

3) Considere um sítio de notícias com escala global – e.g. CNN – e várias edições: Europa, US, Ásia. Neste tipo de sítios existem diferentes tipos de páginas: página de entrada, com apontadores para várias notícias que são modificadas frequentemente ao longo do dia, páginas das notícias que podem sofrer algumas alterações numa fase inicial, e imagens e vídeos usados nas notícias. Para implementar este sistema, seria interessante usar edge servers (servidores na periferia da rede, tipicamente colocados nos ISPs)? Se sim, explique como os poderia usar.

Sim/ Não , porque…

4) Considere que pretende implementar um sistema de partilha e divulgação de fotografias, à escala mundial, com as seguintes características:
    - Os utilizadores podem guardar as suas fotografias, as quais estarão organizadas em álbuns.
    - Os álbuns de um utilizador podem ser estritamente privados, ser partilhados com uma rede de amigos (também utilizadores) ou, ainda, ser completamente públicos.
    - Existe um álbum *Favoritos,* para cada utilizador, que guarda as últimas 500 fotografias públicas que este assinalou com um "like". Estes álbuns são públicos e habitualmente são muito consultados pela sua rede de amigos.
    - Existe um número reduzido de utilizadores que são celebridades, cujos álbuns não privados são consultados avidamente pela sua vasta rede de amigos e público em geral.
    - Existem álbuns "BestOf", públicos, que reúnem as fotografias mais populares do sistema. Estes álbuns revelaram-se muito populares pelo mundo fora.

a) No contexto do sistema pretendido, indique para que tipo de álbuns / fotografias usaria cada uma das seguintes técnicas: *particionamento*, *replicação* e *geo-replicação* nos servidores, explicando porque considera apropriado usar essa técnica e o benefício da mesma.

Particionamento

Replicação

Geo-replicação

b) Das características apresentadas, qual (ou quais) é que poderiam justificar a utilização duma arquitetura P2P quando consideradas isoladamente? Quais as propriedades dessa aproximação?

5) Considere que se pretende criar um protocolo de comunicação para implementar uma semântica exatamente uma vez. De seguida apresenta-se o pseudo-código do servidor single-threaded:

```
forever
    msg = s.receive()
    if( ! log.contains( msg.id)) {
        atomic {
            res = exec(msg)
            log.put( msg.id, res)
        }
    }
    res = cache.get( msg.id)
    s.send( [msg.id,res])
```

a) Explique porque é que, no servidor, é necessário executar duas operações de forma atómica (indicadas com a keyword atomic).

b) Considere que todas as operações a executar são idempotentes. Apresente o pseudo-código mais simples possível do cliente e do servidor para executar uma semântica equivalente a exactly-once.

CLT	**SRV**

Anexo A

```
10 @Path("/simpleFileServer")
11 public class SimpleFileServer {
…
70
71
72
73
74 public Response createNewFile(                    String filename,          byte[] data) {
75    File f = new File(".", filename);
76    if(!f.exists()) {
77        try {
78            Files.write(f.toPath(), data, StandardOpenOption.CREATE_NEW);
79            return Response.ok().build();
80        } catch (IOException e) {
81            e.printStackTrace();
82            return Response.status(Status.INTERNAL_SERVER_ERROR).build();
83        }
84    } else {
85        return Response.status(Status.CONFLICT).build();
86    }
87  }
88
89
90
91
92  public Response listFiles() {
93        String[] files = new File(".").list();
94        return Response.ok(files).build();
95  }
96 }
```

Anotações JAVA REST Jersey:

@Path( )
@GET
@POST
@PUT
@DELETE
@Consumes( )
@Produces( )
@PathParam( )

Listagem de Media Types a considerarem:

MediaType.APPLICATION_OCTET_STREAM
MediaType.APPLICATION_JSON

Anexo B

Código do cliente: (linhas de código da classe indicadas à esquerda)

```
10  public class ListFiles {
11
12      public static void main(String[] args) {
13          ClientConfig config = new ClientConfig();
14          Client client = ClientBuilder.newClient(config);
15
16          String url = "http://localhost:9090/simpleFileServer/listFiles";
17          WebTarget target = client.target(UriBuilder.fromUri(url).build());
18
19          try {
20              String[] fileList = target.request().get(String[].class);
21              for(String s: fileList)
22                  System.out.println(s);
23          } catch (RuntimeException e) {
24              System.err.println("There was an error. Exception Stack trace is the following");
25              e.printStackTrace(System.out);
26          }
27      }
28  }
```

Output produzido pela execução do cliente:

```
There was an error. Exception Stack trace is the following
javax.ws.rs.NotFoundException: HTTP 404 Not Found
    at org.glassfish.jersey.client.JerseyInvocation.convertToException(JerseyInvocation.java:1008)
    at org.glassfish.jersey.client.JerseyInvocation.translate(JerseyInvocation.java:816)
    at org.glassfish.jersey.client.JerseyInvocation.access$700(JerseyInvocation.java:92)
    at org.glassfish.jersey.client.JerseyInvocation$2.call(JerseyInvocation.java:700)
    at org.glassfish.jersey.internal.Errors.process(Errors.java:315)
    at org.glassfish.jersey.internal.Errors.process(Errors.java:297)
    at org.glassfish.jersey.internal.Errors.process(Errors.java:228)
    at org.glassfish.jersey.process.internal.RequestScope.runInScope(RequestScope.java:444)
    at org.glassfish.jersey.client.JerseyInvocation.invoke(JerseyInvocation.java:696)
    at org.glassfish.jersey.client.JerseyInvocation$Builder.method(JerseyInvocation.java:420)
    at org.glassfish.jersey.client.JerseyInvocation$Builder.get(JerseyInvocation.java:316)
    at clt.ListFiles.main(ListFiles.java:20)
```

Anexo C.1

Anexo C.2