Projeto 2

Sistemas Distribuídos, 2024-25

(Versão 1, 6 de Maio 2025)

Prazos

2º Projeto – 4 de junho, 23h59 (online - código + formulário)

A submissão do trabalho vai seguir os mesmos moldes do primeiro trabalho, vão ter de submeter no formulário online de submissão, o repositório *git*, e um link para o *commit* da vossa entrega. Podem usar o mesmo repositório que usaram para a primeira entrega ou usar um novo repositório. No caso de usarem um novo repositório vão ter de dar autorizações aos docentes tal como no primeiro projeto. Tenham atenção para evitar que o vosso repositório seja público.

Objetivo

O objetivo deste segundo projeto é estender o primeiro projeto com as seguintes funcionalidades:

- Mecanismos de segurança fundamentais;
- Mecanismos para eliminar imagens do serviço de imagens que já não se encontrem em uso em nenhum Post, e simetricamente, remover links para imagens dos Posts que apontem para imagens que já não se encontrem disponíveis no serviço *Image*;
- Interação com serviços externos;
- Tolerância a falhas (crash) – através da replicação do serviço de *Content*.

A arquitetura da solução (a menos das alterações referidas acima e ao longo deste enunciado) será a mesma, assim como as APIs e os Modelos de dados trocados com o cliente. A vossa solução deve obedecer estritamente às APIs e modelos de dados definidos no primeiro projeto.

Funcionalidades

De seguida discutimos as funcionalidades que se pretendem alcançar, incluindo algumas indicações de diferentes possibilidades de implementação para essas funcionalidades, indicando-se a valoração das mesmas (incluindo de soluções que sigam estratégias distintas).

Segurança (3,5 valores):

O objetivo desta funcionalidade é tornar o sistema seguro, impedindo que elementos não autorizados executem operações em qualquer um dos serviços que compõe o sistema (*Users*, *Content*, *Image*).

Para alcançar este objetivo, a solução deve incluir os seguintes mecanismos:

1. *utilizar canais seguros, usando TLS, com autenticação do servidor.*

Garantir que a comunicação entre clientes e servidores é efetuada sobre canais cifrados através de TLS (garantindo assim integridade e confidencialidade) e que quando uma entidade (cliente final

ou servidor a agir como cliente) o mesmo tem garantias de que a entidade com quem está a comunicar é o servidor correto. Note que as operações dos clientes que podem manipular o estado do sistema incluem uma password para verificar que o cliente está autorizado a efetuar a operação indicada (esta última parte já se verificava nas interfaces definidas no sistema);

2. *caso existam operações executadas apenas entre os servidores, garantir que estas não podem ser invocadas por um cliente*

Existem situações em que um servidor deve efetuar pedidos a outro servidor do sistema (e.g., quando um utilizador é removido é necessário efetuar pedidos do servidor de *Users* para o servidor de *Content* de forma a manipular a informação existente neste servidor para refletir a remoção do utilizador. Esta API por ser utilizada por um utilizador não autorizado para manipular de forma inválida o estado dos serviços.

Para evitar isto, quando um servidor executa uma destas operações especiais, o mesmo deve autenticar-se perante o servidor que está a contactar. Para conseguir fazer esta autenticação sugere-se a utilização dum segredo partilhado entre os servidores, o qual pode ser passado como um parâmetro da linha de comandos ao arrancar cada um dos servidores.

Eliminação automática de imagens não utilizadas (2 valores):

Tal como identificado por vários alunos durante a realização do primeiro projeto, existem casos (e.g., remoção de um Post) em que todos os *Posts* que apontavam para uma imagem no serviço *Image* deixam de existir, o que faz com que essa imagem passe a ser virtualmente inacessível (exceto pelo seu dono, que, entretanto, pode ele próprio já não existir no sistema). De forma a evitar o desperdício de espaço de armazenamento no serviço de *Image*, estas imagens devem eventualmente ser removidas.

Uma sugestão para a implementação desta funcionalidade passa por no servidor de *Image* ter um *thread* que periodicamente para imagens que tenham sido criadas à mais de um certo tempo (e.g., 30 segundos) interroga através de um novo método no serviço de *Content* se existem ainda posts com referências para essa imagem, e em caso negativo as remove (esta operação entre servidores deve autenticar o servidor de origem).

Outra alternativa de implementação é o uso de um serviço de publish-subscribe (e.g., Kafka) para permitir de forma reativa e indireta a interação entre o(s) servidores de *Content* e o servidor de *Image* de forma a que o última consiga manter uma contagem com o número de referências ativas para cada imagem, de forma a poder eliminar imagens sem referências (mais uma vez não devem ser eliminadas imagens acabadas de criar um por período (e.g., 30 segundos) de forma a dar tempo ao utilizador que criou a imagem de a utilizar num dos seus Posts.

Remoção automática de links para imagens não disponíveis em Posts (2 valores):

Também identificado por alunos durante a realização do primeiro projeto, existe a hipótese de uma imagem ser removida enquanto existem referências para essa imagem em Posts no serviço de

Content. Esta situação pode ser resolvida de forma automática pelo servidor de *Content*, substituindo essas referências pelo valor ***null***.

As alternativas discutidas acima para endereçar o problema simétrico podem ser exploradas (com as devidas alterações) de forma a resolver esse problema. Uma outra alternativa mais simples, mas que afeta o tempo de resposta para a operação de apagar imagem emitida por um utilizador final para o serviço de *Image*, é a criação de um novo endpoint/função no serviço de *Content* para remover todas as referências a essa imagem em todos os Posts atualmente gravados (claro que esta solução peca por não ser efetiva para um Post que é criado concorrentemente com a remoção da imagem e que refere essa imagem).

Interação com um serviço externo (4,5 valores):

O objetivo desta funcionalidade é criar um servidor com o serviço *Image* (e que disponibilize uma interface REST ou gRPC – os alunos podem escolher) que interaja com o serviço externo Imgur com autenticação OAuth para armazenar o conteúdo das imagens.

Para implementação desta funcionalidade sugere-se a utilização da biblioteca ScribeJava (https://github.com/scribejava/scribejava), como apresentado nas aulas práticas.

Apesar deste serviço não ser replicado, deve ser possível lançar mais do que um destes servidores simultaneamente, sem que estes partilhem o estado através do serviço Imgur.

Para que seja possível testar o serviço de forma automática, usando o Tester, é necessário poder indicar ao servidor, quando este inicia, que deve iniciar com o estado do serviço externo *limpo*. Para este fim, o Tester passará como primeiro parâmetro do servidor que interage com o serviço externo o valor **true** para indicar que o estado anterior deve ser ignorado.

Se o Tester passar o valor **false**, o estado existente no serviço Imgur deve ser usado pelo servidor.

Tolerância a falhas – Servidor de Content (até 9 valores):

De forma a garantir a tolerância a falhas do serviço de Contents o mesmo será replica num conjunto de máquinas (containers Docker no contexto do Tester) diferentes. Os alunos devem garantir a correta operação deste serviço em todas as suas operações. Existem várias estratégias de implementação para alcançar este fim que são descritas de seguida. Deve-se notar que cada uma destas estratégias têm uma valoração distinta máxima de acordo com a semântica providenciada e a dificuldade associada a essa implementação:

Alternativa 1 (até 7 valores):

Implementar uma solução que permita tolerar falhas numa máquina em que esteja a correr um servidor de *Content*, replicando o servidor de *Content* com uma solução de replicação de máquina de estados, recorrendo a um sistema de comunicação indireta - e.g. Kafka.

A solução deve tolerar a falha de qualquer um dos servidores de *Content*. A solução deve garantir que um cliente lê sempre o estado dum servidor que tem uma versão tão atual quanto a versão do servidor que foi acedido anteriormente.

Para tal, o servidor pode adicionar **headers** a todas as respostas a enviar aos clientes, os quais serão enviados nas próximas operações executadas pelo mesmo cliente (o Tester reenviará todos os headers começados em **X-FCTREDDIT**).

Esta solução deve apenas considerar a replicação de servidores de *Content* que expõem uma interface REST.

Alternativa 2 (até 6 valores):

Implementar uma solução que permita tolerar falhas numa máquina em que esteja a executar um servidor que disponibilize o serviço de *Content*, replicando o servidor de *Content* com uma solução de replicação de máquina de estados, implementando o protocolo primário/secundário.

A solução deve tolerar a falha de qualquer servidor; no caso da falha do servidor primário, o sistema deve garantir que continua a ser possível fazer leituras, mas não necessita permitir a execução de escritas.

A solução deve garantir que um cliente lê sempre o estado dum servidor que tem uma versão tão atual quanto a versão do servidor que foi acedido anteriormente.

Para tal, o servidor pode adicionar **headers** a todas as respostas a enviar aos clientes, os quais serão enviados nas próximas operações executadas pelo mesmo cliente (o Tester reenviará todos os headers começados em **X-FCTREDDIT**).

O programa deve suportar a falha de 1 servidor, estando os protocolos implementados configurados para tal. Esta solução deve apenas considerar a replicação de servidores de *Content* que expõem uma interface REST.

Alternativa 3 (até 9 valores):

Implementar uma solução que permita tolerar falhas numa máquina em que esteja a corer um servidor que disponibiliza o serviço de Content, replicando o servidor de *Content* com uma solução de replicação de máquina de estados, implementando o protocolo primário/secundário.

A solução deve tolerar a falha de qualquer servidor, incluindo o servidor primário -- para tal, deve eleger um novo primário, eventualmente recorrendo a um serviço externo como por exemplo o Zookeeper.

A solução deve garantir que um cliente lê sempre o estado dum servidor que tem uma versão tão atual quanto a versão do servidor que foi acedido anteriormente.

Para tal, o servidor pode adicionar **headers** a todas as respostas a enviar aos clientes, os quais serão enviados nas próximas operações executadas pelo mesmo cliente (o Tester reenviará todos os headers começados em **X-FCTREDDIT**).

O servidor deve suportar a falha de 1 servidor, estando os protocolos implementados configurados para tal. Esta solução deve apenas considerar a replicação de servidores de *Content* que expõem uma interface REST.

Fatores depreciativos

O código entregue deverá seguir boas práticas de programação. A repetição desnecessária de código, inclusão de constantes mágicas, o uso de estruturas de dados inadequadas, soluções ineficientes, etc., poderá incorrer numa penalização (penalização max: 2 valores).
Falta de robustez e comportamentos erráticos da solução são motivo para penalização (penalização máxima: variável de acordo com a instabilidade observada).

A solução desenvolvida deve continuar a suportar falhas temporárias de comunicação tal como no primeiro projeto (para além das garantias descritas acima).

Execução

O trabalho deve ser executado nos mesmo grupos utilizados para o primeiro projeto. Alunos que não queiram manter o grupo utilizado no primeiro projeto terão de fazer o projeto sozinhos.

Este projeto pode ser desenvolvido sobre o código desenvolvido pelo grupo/aluno no primeiro projeto ou alternativamente utilizando como base uma das implementações do primeiro projeto desenvolvido pelos docentes da disciplina, as quais se encontram aqui:

Solução 1 (feita por Pedro Camponês):
Solução 2 (feita por João Leitão):

Avaliação

A avaliação do trabalho terá em conta os seguintes critérios:

- Funcionalidades desenvolvidas e a sua conformidade com a especificação, tendo como base os resultados da bateria de testes automáticos;
- Qualidade das soluções utilizadas para desenvolver cada funcionalidade;
- Qualidade do código desenvolvido.

A classificação final do aluno na componente prática é individual e será menor ou igual à classificação do trabalho, sendo estas variações em função dos resultados obtidos numa discussão individual, no caso de o aluno ser convocado para esta (falha de comparência na discussão levará a uma nota final na avaliação de zero valores). No caso do aluno não ser convocado para discussão, a nota do trabalho passará a ser a nota final do aluno na componente prática.

Bateria de testes

A bateria de testes destinada a verificar a conformidade da solução com a especificação será disponibilizada brevemente (de forma incremental).

Ambiente de desenvolvimento

Todo o material de apoio fornecido pressupõe que o desenvolvimento será em ambiente Linux e Java 17. A validação do trabalho por via da bateria de testes automática fará uso de tecnologia Docker.

Histórico de alterações

6 maio 2025

- Disponibilização da primeira versão do enunciado.