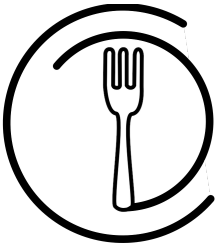
DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

LEIC-A, LEIC-T, LETI, MEIC-T, MEIC-A

Sistemas Distribuídos

2º Semestre – 2018/2019

Enunciado do Projeto: ForkExec



Introdução

O objetivo do projeto de Sistemas Distribuídos é desenvolver um sistema baseado em *Web Services SOAP*, implementados na plataforma Java, para a gestão de uma plataforma de encomenda de refeições para executivos chamada ForkExec.

À semelhança de sistemas como o UberEats, a plataforma ForkExec permite aos seus utilizadores encomendarem refeições em diferentes restaurantes espalhados pela cidade utilizando uma aplicação móvel.

1. Primeira parte: sistema ForkExec

O sistema ForkExec é composto por um agregador (*hub*) que recebe pesquisas e encomendas. Este *hub* é responsável por delegar as pesquisas nos restaurantes conhecidos.

Cada restaurante oferece menus diversos. Um menu é constituído por entrada, prato principal e sobremesa. Cada restaurante dá indicação do tempo de confeção do menu pretendido, o que é importante para quem está com pressa.

As refeições neste sistema não são pagas diretamente em dinheiro. Primeiro é necessário converter dinheiro em *pontos-comes*.

Cada utilizador tem uma conta no sistema, identificada por um endereço de *email* e protegida por uma palavra-chave [1]. A cada conta está associado um saldo de pontos. Inicialmente,

[1] Por simplificação, a autenticação dos utilizadores está fora do âmbito do projeto. Ou seja, considera-se que o utilizador apenas precisa de indicar o seu endereço de *email* quando invoca operações neste sistema.

cada utilizador começa com um saldo de 100 pontos de oferta. O utilizador pode carregar o seu saldo a qualquer momento. Os *pontos-comes* são consumidos quando são feitas encomendas de refeições.

A Figura 1 mostra uma visão global dos componentes da solução.

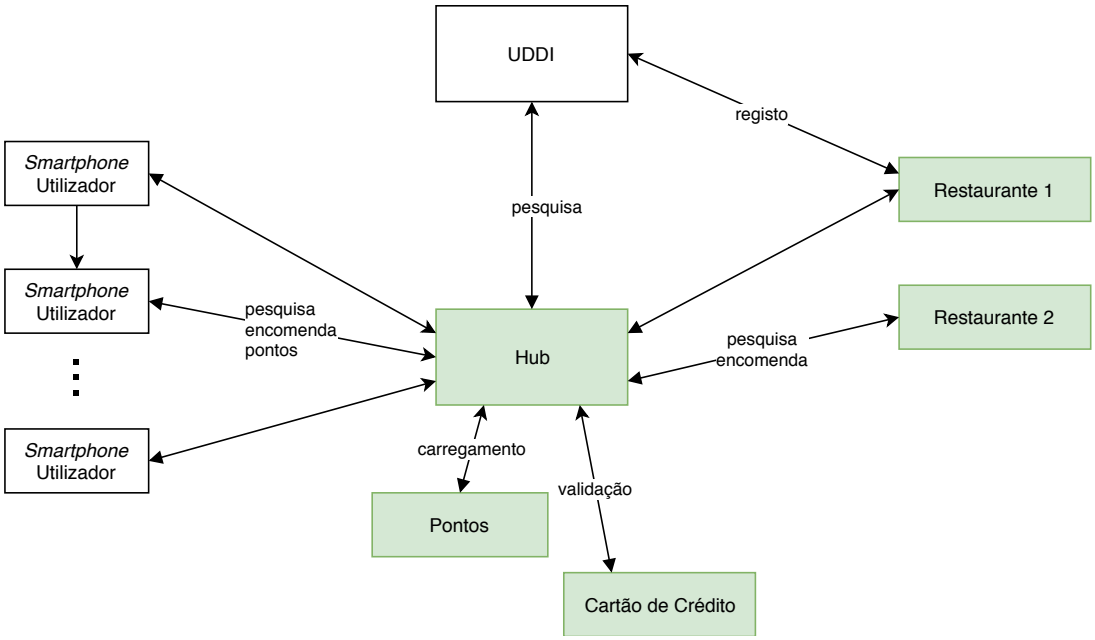


Figura 1: Componentes do projeto.

A componente central é o *hub*, que serve de serviço intermediário entre os utilizadores e os serviços autónomos que compõem o *back-end* do sistema ForkExec.

Os utilizadores utilizam uma aplicação cliente no seu *smartphone*. A aplicação invoca serviços do *hub*, que permitem ao utilizador consultar o saldo da sua conta, fazer pesquisas e encomendar refeições.

O servidor de pontos (*pts*) é responsável por manter as contas dos utilizadores e respetivos saldos.

Como o carregamento de novos *pontos-comes* é pago por cartão de crédito, o servidor de validação de cartões de crédito (*cc*) permite ao *hub* validar os cartões de crédito antes de submeter os carregamentos ao *pts*.

Muitos dos serviços oferecidos pelo ForkExec implicam invocar serviços oferecidos pelos restaurantes que estão espalhados pela cidade. Para tal, em cada restaurante (*rst*) existe um servidor local que gere a informação desse restaurante. Em particular, o servidor de cada restaurante é capaz de responder a pesquisas e de receber encomendas.

O objetivo do projeto é desenvolver os serviços *hub*, *pts* e *rst*. O serviço *cc* já está implementado pelo corpo docente e pronto a ser invocado. Fica **fora do âmbito do projeto** desenvolver a aplicação móvel utilizada pelos utilizadores.

1.1. Tecnologia

Todos os componentes do projeto serão implementados na linguagem de programação Java. A invocação remota de serviços deve ser suportada por *Web Services SOAP* utilizando JAX-WS

(Java API for XML Web Services).

Cada servidor (*hub*, *rst*, *pts*) é lançado de forma autónoma. Todos os servidores devem ser localizados dinamicamente pelos respetivos clientes, por intermédio de um servidor UDDI bem conhecido. Mais precisamente, cabe a cada servidor, quando lançado, registar-se a si próprio no servidor UDDI, indicando o seu URL. Os nomes a usar para os serviços são: `CXX_hub`, `CXX_pts`, `CXX_rst1`, `CXX_rst2`, etc[2].

Note-se que o conjunto de restaurantes não é estático nem pré-conhecido pelo *hub*, logo é através da consulta do UDDI que o *hub* descobre e localiza o conjunto atual de restaurantes.

Não se exige nem será valorizado o armazenamento persistente do estado dos servidores. Isto aplica-se a todas as partes do projeto.

Nas sub-secções seguintes fornecem-se detalhes sobre cada serviço.

1.2. Servidor Agregador (*hub*)

O *hub* permite fazer pesquisas em todos os restaurantes, encomendar refeições e consultar a conta do utilizador. De seguida descrevem-se as operações que este serviço oferece.

Existem duas operações de pesquisa: `searchHungry` e `searchDeal`. Ambas permitem fazer uma pesquisa por texto e devolvem os menus cuja descrição de entrada, prato principal ou sobremesa contenha o texto recebido. O texto de pesquisa não pode ser vazio nem ter espaços. A lista de resultados destas operações é ordenada por ordem crescente do tempo de preparação no caso da operação `searchHungry` e por ordem crescente do preço no caso da operação `searchDeal`. Caso não sejam encontrados produtos, é devolvida uma lista vazia (mas não nula).

Existe uma operação para adicionar um menu ao carrinho de compras (`addFoodToCart`). É possível depois encomendar os vários menus no carrinho através da operação `orderCart`. É também possível limpar o carrinho com a operação `clearCart`.

O *hub* fornece operações que vão ser depois encaminhadas para o servidor de pontos: `activateAccount` para ativar uma conta de utilizador, `loadAccount` para carregar a conta com pontos e `accountBalance` para consultar o saldo de pontos.

As operações estão descritas de forma detalhada no seguinte contrato WSDL:

`http://disciplinas.tecnico.ulisboa.pt/leic-sod/2018-2019/labs/proj/hub.1_0.wsdl`

1.3. Servidor de Pontos (*pts*)

A operação `activateUser` ativa um novo utilizador no sistema, levando o servidor de pontos a criar e inicializar o estado interno sobre este utilizador. Esta operação recebe o endereço de *email* do novo utilizador, que identifica a sua conta. Assume-se que o endereço de *email* segue o formato geral *utilizador@dominio*, em que utilizador e domínio são sequências de caracteres alfa-numéricos, com uma ou mais partes separadas por ponto "." e que não podem ser vazios.

Existem também operações para adicionar (`addPoints`) e remover pontos (`spendPoints`).

[2]O identificador do grupo tem o formato CXX, onde: C representa o campus (A para Alameda e T para Taguspark); XX representa o número do grupo de SD atribuído pelo Fénix. Por exemplo, o grupo A22 corresponde ao grupo 22 sediado no campus Alameda; já o grupo T31 corresponde ao grupo 31 sediado no Taguspark.

Os carregamentos em EUR são convertidos em *pontos-comes* de acordo com a Tabela 1.

Tabela 1: Valores para carregamento de *pontos-comes*.

EUR	***pontos-comes***
10	1000
20	2100
30	3300
50	5500

O saldo de pontos pode ser consultado com a operação `pointsBalance`.

As operações do servidor de pontos estão descritas de forma detalhada no seguinte contrato WSDL:

`http://disciplinas.tecnico.ulisboa.pt/leic-sod/2018-2019/labs/proj/pts.1_0.wsdl`

A informação dos pontos é apenas guardada neste servidor, sendo que o *hub* deverá sempre consultar este servidor para saber o saldo de uma conta.

1.4. Servidor de Validação de Cartão de Crédito (*cc*)

Para autorizar um carregamento de pontos será necessário fornecer um número de cartão de crédito válido. Podem usar, por exemplo, o número 4024007102923926. Existem outros números de teste disponíveis na Internet.

A validação de cartões de crédito será feita recorrendo a um serviço externo disponibilizado pelos docentes e de uso obrigatório.

Importante: nunca testar o serviço com dados de cartões de crédito reais, pois a comunicação HTTP entre o cliente e o servidor não é segura e pode ser facilmente intercetada.

1.5. Servidor de Restaurante

As operações de pesquisa e encomenda do *hub* necessitam de invocar operações dos servidores de restaurantes. As informações dos menus, bem como os preços das refeições, são mantidas apenas nos próprios restaurantes.

A operação `getMenu` permite aceder aos detalhes de um menu, dado o seu identificador.

A operação `searchMenus` permite pesquisar por menus cuja descrição de entrada, prato principal ou sobremesa contenha o texto recebido. O texto de pesquisa não pode ser vazio nem conter espaços.

A operação `orderMenu` permite encomendar um menu.

As operações estão descritas de forma detalhada no seguinte contrato WSDL:

`http://disciplinas.tecnico.ulisboa.pt/leic-sod/2018-2019/labs/proj/rst.1_0.wsdl`

1.6. Operações auxiliares

Cada serviço dispõe de um conjunto de operações de controlo que se destinam a facilitar a realização de testes. Estão funções não necessitam elas próprias de ser testadas exaustivamente. Por convenção, o seu nome começa por `ctrl_`.

A operação `ping` responde com uma mensagem de diagnóstico não vazia que deve ilustrar o melhor possível o estado do servidor invocado.

A operação `clear` limpa totalmente o estado do servidor invocado; ou seja, coloca-o no estado inicial com os valores por omissão.

As operações `init...` permitem definir parâmetros de configuração do servidor.

1.7. Testes

Espera-se que cada projeto inclua testes que permitam cobrir todos os requisitos funcionais do enunciado.

Os *testes de integração* (IT) verificam o cumprimento do contrato de um *Web Service* através de invocações remotas. Devem usar JUnit para invocar todos os servidores remotos necessários (que se assume terem sido previamente lançados de forma correta).

Para os testes de integração deverão ativar **uma** instância do servidor *hub*, **uma** instância do servidor *pts* e pelo menos **duas** instâncias de servidores *rst*.

2. Segunda Parte: Tolerância a Faltas

A tolerância a faltas do serviço ForkExec tem alguns aspetos críticos. Em especial, é importante que as alterações aos saldos de cada utilizador (pontos gastos, pontos ganhos) não se percam nem sejam corrompidas, mesmo sabendo que o servidor de pontos poderá falhar ocasionalmente. Por essa razão, pretende-se estender o serviço para que suporte a replicação dos saldos das contas de utilizador, o que dará maior disponibilidade às operações que manipulem esses mesmos saldos.

A solução planeada passa por criar réplicas dos servidores *pts* para neles manter uma réplica dos saldos de conta dos utilizadores do ForkExec.

Assim sendo, sempre que o servidor ForkExec precise de consultar ou alterar o saldo de um utilizador, este poderá recorrer ao conjunto de gestores de réplica para aceder ao saldo do utilizador de forma tolerante a faltas.

Pretende-se seguir a abordagem de replicação ativa, implementando o protocolo *quorum consensus* a estudar nas aulas teóricas para coordenar as leituras e escritas nas réplicas. O sistema replicado pode assumir que existe um conjunto estático de N gestores de réplica; o parâmetro N é carregado pelo servidor *hub* aquando do seu lançamento e não pode ser alterado enquanto o sistema estiver em execução.

O parâmetro N é também carregado por cada gestor de réplica (instância do servidor *pts*) aquando do lançamento do sistema e não pode ser alterado enquanto o sistema estiver em execução. Cada cliente é identificado univocamente por um inteiro, que se assume previamente atribuído e conhecido pelo próprio cliente.

É desejável que, tanto quanto possível, as opções tomadas privilegiem o desempenho da

resposta ao cliente. Recomenda-se que cada grupo considere variantes do protocolo *quorum consensus* original que, tendo em conta o cenário específico deste projeto, permitam um melhor desempenho do sistema.

Em particular, recomenda-se que os grupos desenhem e implementem variantes com desempenho e/ou disponibilidade superior explorando as seguintes oportunidades:

- Neste sistema replicado, existe apenas um cliente (a instância única do *hub*) que pode invocar operações sobre o conjunto de gestores de réplicas.
- Um sub-conjunto das operações do serviço *pts* pode ser assegurado com garantias mais fracas.

O uso de tecnologias como *invocação assíncrona* de *Web Services* pode ser especialmente útil para alcançar esses objetivos. No entanto, as otimizações/simplificações não devem pôr em causa a correção do sistema replicado – mais precisamente, o modelo de consistência deve ser rigorosamente definido e depois respeitado.

Como modelos de interação e faltas, deve assumir-se que:

- Os gestores de réplica podem falhar silenciosamente mas não arbitrariamente, i.e., não há falhas bizantinas;
- No máximo, existe uma minoria de gestores de réplica em falha em simultâneo;
- O sistema é assíncrono e a comunicação pode omitir mensagens (apesar do projeto usar HTTP como transporte, deve assumir-se que outros protocolos de menor fiabilidade podem ser usados).

3. Avaliação

3.1. Colaboração e Entregas

O Git é um sistema de controlo de versões do código fonte que é uma grande ajuda para o trabalho em equipa. Toda a partilha de código para trabalho deve ser feita através do GitHub.

A atualização do repositório deve ser feita com regularidade, correspondendo à distribuição de trabalho entre os estudantes e às várias etapas de desenvolvimento. Cada elemento do grupo deve atualizar o repositório do seu grupo à medida que vai concluindo as várias tarefas que lhe foram atribuídas.

As entregas do projeto serão feitas também através do repositório GitHub[3]. A cada parte do projeto a entregar estará associada uma *tag*. Cada grupo tem que marcar o código que representa cada entrega a realizar com uma *tag* específica – `SD_P1` e `SD_P2` – antes da hora limite de entrega.

3.2. Serviços externos

Alguns dos serviços necessários serão disponibilizados para utilização remota:

[3]Os grupos que não usam GitHub para trabalhar podem recorrer a uma entrega via Fénix. Embora não seja suposto um grupo entregar o projeto por ambas as vias (Fénix e GitHub), nessas situações o corpo docente considerará **apenas** o projeto submetido via Fénix.

- UDDI: `http://uddi.sd.rnl.tecnico.ulisboa.pt:9090`
- Validação de Cartões de Crédito: `http://ws.sd.rnl.tecnico.ulisboa.pt:8080/cc?WSDL`

3.3. Qualidade do código

A qualidade da estrutura base engloba os seguintes aspetos de avaliação (em todas as partes):

- Configuração (POMs e *Handlers*)
- Código legível (incluindo comentários relevantes)
- Tratamento de exceções adequado
- Sincronização correta

3.4. Primeira parte (P1)

A primeira parte vale 10 valores em 20, distribuídos da seguinte forma:

- *hub* (30%)
  - Qualidade da estrutura base (20%)
  - Implementação das operações (50%)
  - Testes de integração desenvolvidos pelo grupo (30%)
- *pts* (25%)
  - Qualidade da estrutura base (20%)
  - Implementação das operações (50%)
  - Testes de integração desenvolvidos pelo grupo (30%)
- *cc* (10%)
- *rst* (25%)
  - Qualidade da estrutura base (20%)
  - Implementação das operações (50%)
  - Testes de integração desenvolvidos pelo grupo (30%)
- UDDI e restante suporte para múltiplos restaurantes (10%)

A data limite de entrega é: *sexta-feira, 5 de abril de 2019, 17:00*.

3.5. Segunda parte (P2)

A segunda parte vale 10 valores em 20, distribuídos da seguinte forma:

- Qualidade da estrutura base (20%)
- Replicação ativa (30%)

- Otimizações (20%)
- Uso de chamadas assíncronas (10%)
- Relatório e demonstração (20%)

A data limite de entrega é: *quarta-feira, 3 de maio de 2019, 17:00*.

Na segunda parte, deve ser também entregue um relatório (juntamente com o código), colocado na pasta `doc/` na raiz do projeto. O documento deve conter:

- 1 folha de rosto:
  - Identificador do grupo em formato CXX;
  - URL do repositório no GitHub;
  - Fotos, números e nomes dos membros do grupo (ordenados por número de estudante crescente, da esquerda para a direita) – Sugere-se que o grupo tire uma foto em conjunto para ajudar a demonstrar um verdadeiro espírito de equipa!
- 3 páginas de conteúdo:
  - Definição do modelo de faltas;
  - Figura da solução de tolerância a faltas;
  - Descrição da figura e breve explicação da solução;
  - Descrição de otimizações/simplificações;
  - Detalhe do protocolo (troca de mensagens).

3.6. Demonstração

Cada grupo deve preparar um *guião de demonstração* com casos de utilização que demonstram as melhores funcionalidades do trabalho.

Tolerância a faltas – Duração inferior a *5* minutos

- Caso *F1*: passos para demonstrar o funcionamento normal da replicação
- Caso *F2*: passos para demonstrar tolerância a falta

O guião deve ser incluídos na pasta `doc/` na raiz do projeto em formato PDF.

O guião deve também incluir instruções de instalação e configuração, que devem começar com a obtenção do código no repositório Git.

A demonstração do trabalho será realizada, ao vivo, na discussão do trabalho.

3.7. Discussão

Todos os estudantes têm discussão final do projeto. As notas das fases (P1 e P2) são indicativas e sujeitas a confirmação na discussão final, na qual todo o trabalho desenvolvido durante o semestre será tido em conta. As notas atribuídas são individuais. É muito importante que a divisão de tarefas ao longo do semestre seja equilibrada pelos membros do grupo.

Todas as discussões e revisões de nota do trabalho devem contar com a presença obrigatória de todos os membros do grupo.

3.8. Atualizações

Para as últimas novidades sobre o projeto, consultar a página Web regularmente:

`http://disciplinas.tecnico.ulisboa.pt/leic-sod/2018-2019/labs/`

O esclarecimento de dúvidas será realizado através do Piazza:

`https://piazza.com/tecnico.ulisboa.pt/spring2019/sd19`

Bom trabalho!