

INSTITUTO SUPERIOR TÉCNICO

SISTEMAS DISTRIBUÍDOS

3º ANO, 2º SEMESTRE 2018/2019

Relatório - Segunda Parte

Autores

Filipe Marques
Jorge Martins
Paulo Dias

Docente

Tomás Grelha da Cunha

A41-ForkExec

May 1, 2019

Contents

1	Definição do Modelo de Faltas	1
2	Solução de Tolerância a Faltas	1
3	Descrição e breve explicação da solução	1
4	Descrição de otimizações/simplificações	2
5	Detalhe do protocolo (troca de mensagens)	2
5.1	Funções no Gestor de Réplica	2
5.1.1	<i>read(String userEmail)</i>	2
5.1.2	<i>write(String userEmail, int points, Tag t)</i>	2
5.2	Funções no <i>Front End</i>	2
5.2.1	<i>activateUser(String userEmail)</i>	2
5.2.2	<i>pointsBalance(String userEmail)</i>	2
5.2.3	<i>addPoints(String userEmail, int pointsToAdd)</i>	2
5.2.4	<i>spendPoints(String userEmail, int pointsToSpend)</i>	2

1 Definição do Modelo de Falhas

Assume-se que:

- 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
- Existem N gestores de réplicas e N é constante e igual a 3
- Os gestores de réplicas podem falhar silenciosamente mas não arbitrariamente
- No máximo, existe uma minoria de gestores de réplica em falha em simultâneo

2 Solução de Tolerância a Falhas

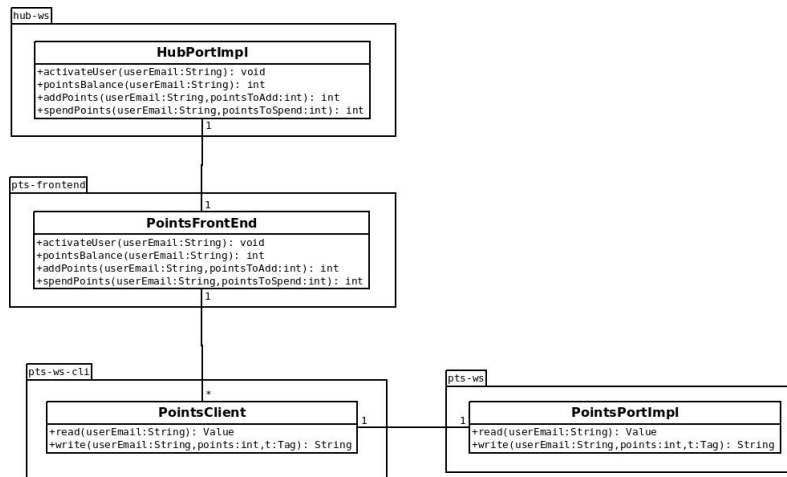


Figure 1: Simplificação do domínio da solução implementada

3 Descrição e breve explicação da solução

Para manter a interface para o *Hub* a *FrontEnd* do *points* continua a implementar as funções necessárias para retornar os valores para o *Hub*, no entanto esta classe implementa o protocolo de replicação ativa.

Para o *FrontEnd* o *PointsClient* funciona apenas como ponto de ligação com o gestor de réplica.

4 Descrição de otimizações/simplificações

5 Detalhe do protocolo (troca de mensagens)

5.1 Funções no Gestor de Réplica

5.1.1 *read(String userEmail)*

- Ao receber *read(userEmail)*:
 1. Vai buscar a *tag* e os *pontos* associados ao *userEmail*
 - 1.1. Se *userEmail* não existe no sistema, então adiciona.
 2. responde com *Value =< pontos, tag >* associado ao utilizador, em que: *tag =< seq, cid >*

5.1.2 *write(String userEmail, int points, Tag t)*

- Ao receber *write(userEmail, points, t)*:
 1. Vai buscar a *tag* associada ao *userEmail*
 - 1.1. Se *userEmail* não existe no sistema, então adiciona.
 2. Se *t.getSeq() > tag.getSeq()*:
 - 2.1. atualiza os *pontos* do utilizador com *points*
 - 2.2. atualiza a *tag* do utilizador com *t*
 - 2.3. responde *ack*
 3. Senão responde *nack*

5.2 Funções no *Front End*

5.2.1 *activateUser(String userEmail)*

5.2.2 *pointsBalance(String userEmail)*

5.2.3 *addPoints(String userEmail, int pointsToAdd)*

5.2.4 *spendPoints(String userEmail, int pointsToSpend)*