Biblioteca para o uso de *Server-sent events* e *WebSockets* em sistemas *backend* multi-nó

Tiago Frazão, n.º 48666, e-mail: a48666@alunos.isel.pt, tel.: 919908403
João Bonacho, n.º 49437, e-mail: a49437@alunos.isel.pt, tel.: 916585748
André Gonçalves, n.º 49464, e-mail: a49464@alunos.isel.pt, tel.: 912259197

Orientador: Professor Pedro Félix, e-mail: pedro.felix@isel.pt

Março de 2024

Contexto

No contexto de sistemas baseados na arquitetura Web, existem aplicações *front-end*, em execução em *browsers* e dispositivos móveis que disponibilizam uma interface aos utilizadores, e sistemas *backend* que expõem APIs HTTP, em execução fora dos dispositivos dos utilizadores. A interação entre aplicações *front-end* e sistemas *backend* é feita, tipicamente, através do protocolo HTTP [1], em que as aplicações desempenham o papel de cliente e os sistemas o papel de servidor. De acordo com o protocolo HTTP, a iniciativa de comunicação parte dos clientes. No entanto, existem situações em que a iniciativa de comunicação por parte dos servidores revela-se importante, especialmente para prevenir o uso excessivo de técnicas *client-polling*. As técnicas *client-polling* consistem no envio periódico de pedidos aos servidores para obter novos dados, causando congestionamento e consumo da largura de banda de rede com pedidos que não resultam na aquisição de novos dados. Além disso, estas técnicas provocam sobrecargas nos servidores e inevitável desfasamento entre a disponibilidade e entrega dos dados [2].

Na tentativa de resolução dos problemas relacionados com as técnicas *client-polling* existem vários mecanismos *server-push* [2]. Estes mecanismos permitem aos servidores enviarem dados aos clientes de forma assíncrona, i.e., sem ser como resposta a pedidos de clientes. As

aplicações de *chat* e os jogos *turn-based* são alguns exemplos de aplicações que beneficiam da utilização destes mecanismos. No caso das aplicações de *chat*, os utilizadores recebem as novas mensagens, e nos jogos *turn-based*, os jogadores recebem o novo estado do jogo aquando da sua vez de jogar. Desta forma, os servidores entregam os novos dados em tempo útil, reduzindo os atrasos significativos nas atualizações de estado das aplicações.

Os *Server-sent events* [3] (SSE) e o protocolo *WebSockets* [4] destacam-se entre os mecanismos *server-push* atualmente existentes. Os *Server-sent events* são mais simples, possibilitando apenas a comunicação unidirecional no sentido servidor-cliente. Por outro lado, o protocolo *WebSockets* é mais complexo, permitindo a comunicação bidirecional, i.e., servidor-cliente e cliente-servidor. Em ambos os mecanismos, o princípio subjacente centra-se no estabelecimento de ligações TCP/IP de longa duração entre clientes e servidores, sempre iniciadas pelos clientes. Quando existe um evento do lado dos servidores que tenha como consequência informar os clientes são usadas essas ligações.

Os servidores, enquanto sistemas *backend* multi-nó, são compostos por um conjunto de nós homogéneos, i.e., instâncias da mesma aplicação *backend*, tipicamente geridas por balanceadores de carga. Desta forma, aumenta-se a capacidade computacional (*Horizontal Scaling* [5]) e redundância dos sistemas, potencialmente com a distribuição de nós por máquinas distintas. Para além disso, promove-se o aumento da disponibilidade, permitindo flexibilidade e mitigação dos efeitos de falhas e manutenção dos nós.

Contudo, o desenvolvimento destes sistemas *backend* multi-nó utilizando mecanismos *server-push* tende a aumentar a complexidade, uma vez que os clientes estabelecem ligações persistentes com um dos nós disponíveis do sistema *backend*. Para o efeito, em cada nó existem representações em memória das ligações, através das quais os nós enviam dados a cada cliente ligado.

Problemas

O desenvolvimento de sistemas *backend* multi-nó que utilizem *Server-sent events* ou *WebSockets* para comunicar com as aplicações *front-end* acarretam um conjunto de problemas inerentes à multiplicidade de nós. Com a existência de vários nós nos sistemas *backend*, um nó que pretende comunicar com um cliente específico, pode não ser o nó em que esse cliente tem a ligação estabelecida. Posto isto, a entrega dos novos dados a esse cliente tem que ser realizada através da representação da ligação presente em memória noutro nó.

Por outro lado, as ligações estabelecidas entre clientes e nós estão sujeitas a quebras, o que pode resultar na mudança do nó que mantinha a ligação com um cliente, após o restabelecimento da ligação. Como consequência, alguma entrega de dados em curso a esse cliente pode ficar em risco.

Solução

O projeto a desenvolver consiste numa biblioteca que visa auxiliar no desenvolvimento de sistemas *backend* multi-nó que utilizem *Server-sent events* ou *WebSockets* para comunicar com as aplicações *front-end*. Esta peça de *software* tem como objetivos reduzir a complexidade de desenvolvimento destes sistemas e resolver os problemas intrínsecos à multiplicidade de nós.

Para cumprir os objetivos, a biblioteca suportará três opções configuráveis para comunicação transparente entre os nós. Nas duas primeiras opções, a biblioteca usará sistemas existentes que forneçam funcionalidades de comunicação, nomeadamente a funcionalidade de *publish-subscribe* [6], que consiste num padrão de publicação de mensagens e recolha por

parte dos subscritores. Na primeira opção, como geralmente os sistemas *backend* recorrem a Sistemas de Gestão de Base de Dados Relacional para armazenamento de informação, aproveitar-se-á as funcionalidades oferecidas pelo *PostgreSQL*. Na segunda opção, será utilizado um outro sistema externo mais adequado para *publish-subscribe*, mediante análise, tais como *RabbitMQ*, *Apache Kafka* ou *Redis*. Na terceira opção, a biblioteca não dependerá de um sistema externo existente que forneça essas funcionalidades, em alternativa, serão os próprios nós responsáveis pela comunicação e coordenação entre si. Desta forma, a realização do projeto resulta numa divisão em três fases, em que cada fase corresponde à implementação da respetiva opção.

A peça de *software* proposta será desenvolvida na linguagem de programação *Kotlin* e terá o *Gradle* como ferramenta de *build* e gestor de dependências. O código fonte e respetiva documentação técnica serão disponibilizados publicamente num repositório *GitHub*.

A testabilidade e validação da solução proposta basear-se-á em testes automáticos para garantir robustez e flexibilidade. Paralelamente, serão produzidas aplicações demonstrativas que utilizem a biblioteca.

Riscos e Desafios

Os riscos do projeto centram-se na integração de novas tecnologias na biblioteca a desenvolver, e análise de mecanismos e protocolos que poderão comprometer o planeamento previamente estipulado e consequentes objetivos. Conjeturam-se como principais desafios, a arquitetura e respetiva implementação da terceira opção, i.e., desenvolvimento da biblioteca sem dependência a sistemas externos existentes que facilitem a comunicação entre nós. Outro desafio frequente no contexto de sistemas de comunicação entre múltiplos nós foca-se na necessidade de enviar dados exclusivamente a um subconjunto específico de clientes, ao invés de distribuí-los por todos os clientes ligados. A título de exemplo, numa aplicação de *chat*, as mensagens são destinadas apenas a um utilizador específico ou a um grupo de utilizadores.

Planeamento

A Figura 1 apresenta o planeamento semanal estabelecido para a realização do projeto, incluindo as entregas obrigatórias.

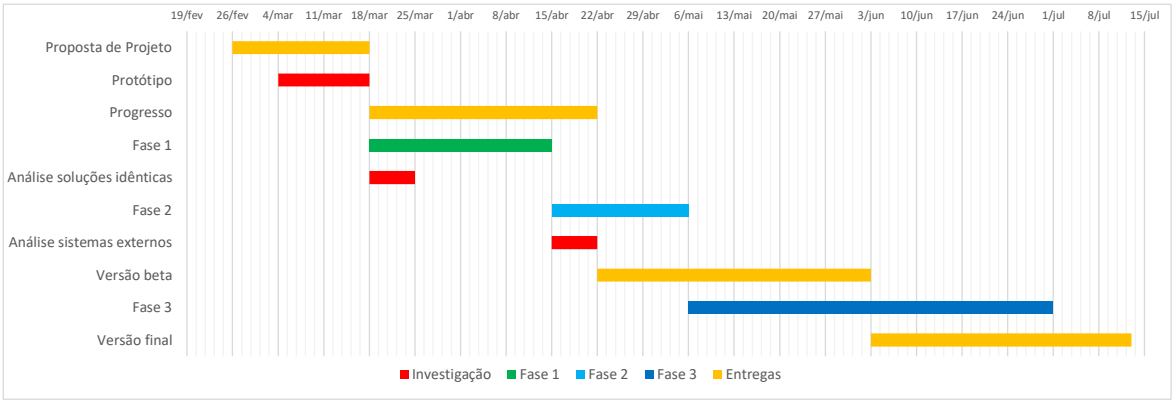

Figura 1: Planeamento Semanal

Referências

[1] Internet Engineering Task Force (IETF). Rfc 2616: Hypertext transfer protocol – http/1.1. https://datatracker.ietf.org/doc/html/rfc2616, 1999. [Online; acedido em 15/03/2024].

[2] Internet Engineering Task Force (IETF). Rfc 6202: Known issues and best practices for the use of long polling and streaming in bidirectional http. https://datatracker.ietf.org/doc/html/rfc6202, 2011. [Online; acedido em 15/03/2024].

[3] Web Hypertext Application Technology Working Group (WHATWG). Server-sent events. https://html.spec.whatwg.org/multipage/server-sent-events.html#server-sent-events, 2024. [Online; acedido em 16/03/2024].

[4] Internet Engineering Task Force (IETF). Rfc 6455: The websocket protocol. https://datatracker.ietf.org/doc/html/rfc6455, 2011. [Online; acedido em 15/03/2024].

[5] Amazon Web Services (AWS). Horizontal scaling. https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.concept.horizontal-scaling.en.html, 2020. [Online; acedido em 17/03/2024].

[6] Redis. Pub/sub (publish/subscribe). https://redis.com/glossary/pub-sub/, 2011. [Online; acedido em 17/03/2024].