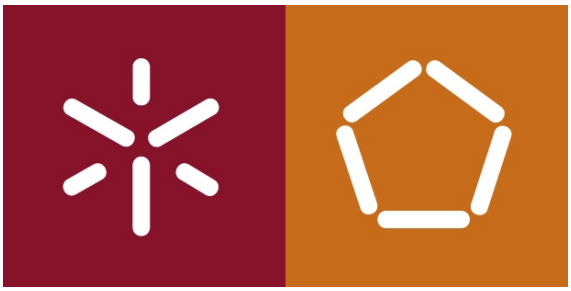

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Interoperabilidade Semântica

Orcid Resume

Trabalho de Grupo

Bárbara Cardoso (A80453)
João Vilaça (A82339)

5 de Março de 2021

Resumo

Depois do aparecimento da Internet e com a democratização do seu uso, temos assistido a uma constante digitalização da sociedade. A maioria dos serviços que usamos estão hoje já disponíveis nos websites das respectivas organizações, onde muitas das quais têm também disponíveis para uso webservices que permitem que a outros programadores desenvolver aplicações que façam uso desses mesmo serviços. Isto permite uma maior facilidade na interoperabilidade de sistemas, sendo muito fáceis e baratos de implementar e também bastante seguros.

Neste contexto, e a título de exemplo de aplicação dos conceitos e conhecimentos adquiridos durante as aulas de Interoperabilidade Semântica, vamos construir um website onde sejam apresentados vários dados dos investigadores pesquisados, entre os quais a sua biografia, as suas publicações e quantas citações cada uma tem.

Conteúdo

1 Introdução

Para garantir o reconhecimento do trabalho realizado por investigadores e demais membros associados a trabalhos académicos, principalmente universitários, existem inúmeros serviços online onde estes podem se registar e usar como "montra"para o seu trabalho. Um exemplo de sucesso neste contexto, é o ORCID, um serviço que atribui aos investigadores que se registam um código pessoal que este podem usar para se identificarem unicamente em demais serviços relacionados. O ORCID é utilizado pelas melhores e mais utilizadas plataformas e repositórios científicos como o SCOPUS ou o RESERARCHID, onde o seu trabalho pode ser apresentado e a influência do mesmo no meio avaliada.

Uma das grandes vantagens deste serviços agora mencionados, é possuírem APIs públicas que permitem o desenvolvimento de novos serviços complementares baseados nas funcionalidades suportadas por estes, sendo desse modo possível complementar possíveis falhas ou informação incompleta em qualquer um destes serviços recorrendo a outros.

Assim, neste projeto, pretende-se desenvolver um serviço, que fazendo uso destas mesmas API, e tirando proveito das facilidades de interoperabilidade que as mesmas proporcionam, seja capaz de recolher, processar e mostrar dados e análise das publicações de um investigador, da sua informação pessoal e do seu percurso académico e profissional tudo a partir de apenas o seu ORCID.

2 Base de Dados

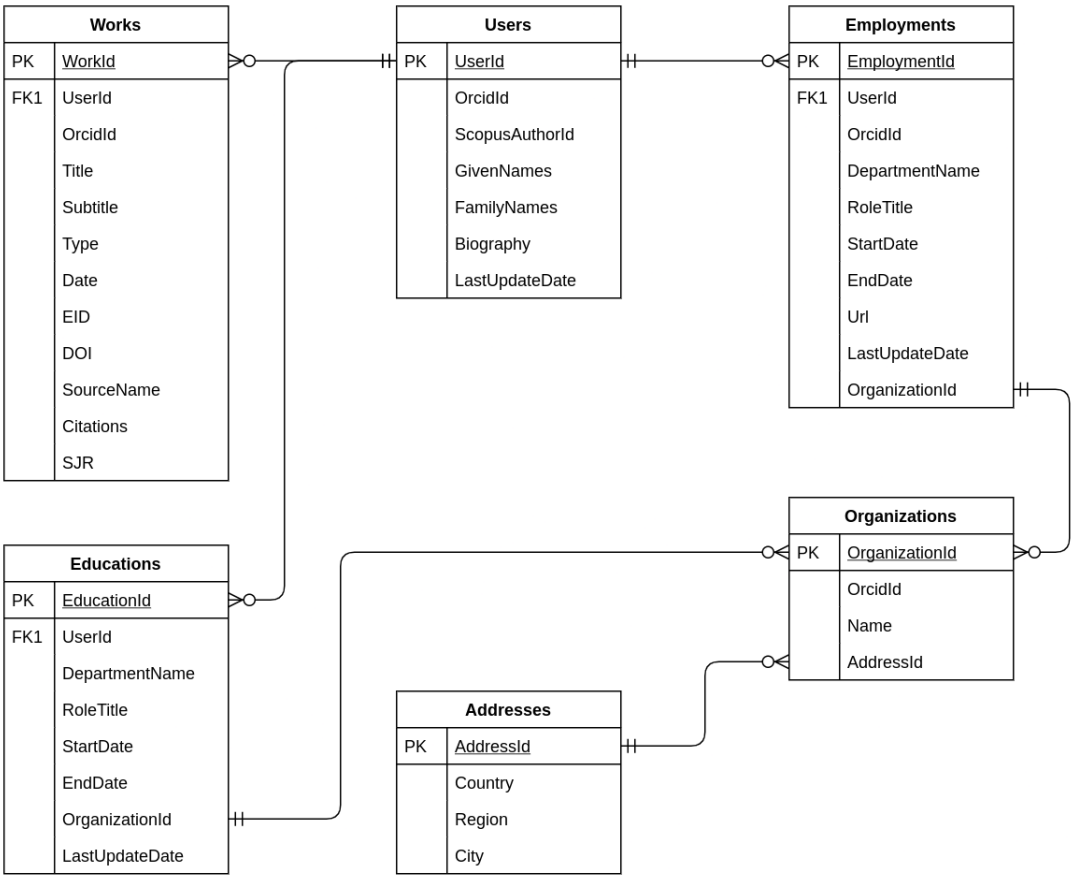

Figura 1: Modelo lógico

A base de dados permite guardar os dados dos utilizadores, dos seus trabalhos, percurso em termos de educação e organizações que serão efectivamente usados pela nossa aplicação. Esta estrutura de base de dados permite ser bastante rápido em termos de consulta de dados que os nossos utilizadores querem, uma vez que consultar a API da Orcid é relativamente lento e permite reduzir imenso a quantidade de dados que percorrem a rede.

3 Consulta de WebServices

A consulta de dados de Web Services será feita exclusivamente pelo Backend, que caso tenha informações novas úteis as armazena na sua base de dados antes de as enviar para o utilizador.

Os módulo de acesso a estes Web Service estão isolados e independentes do resto da aplicação, permitindo uma fácil evolução e acompanhamento de possíveis alterações deste Web Services no futuro, sendo rápido fazer adaptações e manter a disponibilidade do nosso serviço.

3.0.1 Orcid

O Web Service da Orcid está a ser usado para consultar todas as informações pessoais do investigador procurado, das informações gerais dos seus trabalhos e de educação.

3.0.2 Elsevier

O Web Service da Elsevier é usado para consultar o número de citações de determinada publicação e o seu respectivo Impact Factor.

4 Backend API

Para a API, optamos por usar TypeScript pela grande facilidade e rapidez de desenvolvimento que proporciona e pela imensa quantidade de bibliotecas já feitas para JavaScript e para NodeJS existentes que podem ser usadas nesta linguagem.

Para além disto, o TypeScript assegura e força a que os tipo das variáveis e das funções estejam correctos, garantindo uma maior correcção do código do que veríamos em JavaScript, aumentando a segurança e robustez do código, sem perder facilidade em termos de programação.

4.1 Migrações

A construção da base de dados fiel ao modelo previamente apresentado é feita pela própria aplicação, com recurso a migrações que a biblioteca Knex.js oferece, onde podemos incrementalmente fazer a gestão da base de dados, criando ou apagando tabelas, adicionando ou removendo colunas mantendo a coerência em termos de desenvolvimento dos vários programadores do projecto.

4.2 Modelos

O mapeamento entre as entradas da base de dados e os objectos em TypeScript é feita através da biblioteca Objection.js.

Esta biblioteca permitiu-nos facilmente declarar modelos e definir relacionamentos entre os mesmos, fazer uso de transacções para garantir certas propriedades na criação e armazenamento de novas entidades, e ainda simplificar o processo de consulta de informação na base de dados através de operações de Query Building.

4.3 Routes

As rotas são responsáveis por redireccionar os pedidos para os respectivos controladores que serão responsáveis por executar o pedido. Estas seguem todas o mesmo estilo de código, implementando o protocolo REST e os métodos respectivos onde eles fazem sentido.

4.4 Controllers

Os controladores são os responsáveis por executar as operações pedidas, sejam elas a consulta de dados, fazendo o pedido através do ORM e devolvendo as informações filtradas do(s) objecto(s) pedidos, seja actualizar dados, chamando os middlewares que fazem os pedidos às APIs, recebendo os dados, dar-lhes parse, armazenar na base de dados através do ORM e dar return para o utilizador.

5 Página WEB

A página web foi construída para ser dinâmica e simples para o utilizador. Construída principalmente com recurso a JavaScript que captura eventos do utilizador e actualiza a página em conformidade. Mas também a pensar numa boa performance, reduzindo ao máximo o número de chamadas necessárias à API.

Como framework para as partes mais gráficas escolhemos usar Semantic UI, que permite facilmente ter separadores para as várias parte do nosso UI, bem como ter botões e locais para input de texto bonitos e fáceis de usar.

5.1 Investigador

Para escolhermos qual o investigador sobre o qual queremos consultador os dados, basta apenas inserirmos o Orcid ID no campo de texto e pesquisar. Logo de seguida, é feito um pedido à API, que devolve várias informações sobre o mesmo, nomeadamente o seu nome, o seu ID do Scopus, caso exista, e a sua biografia.

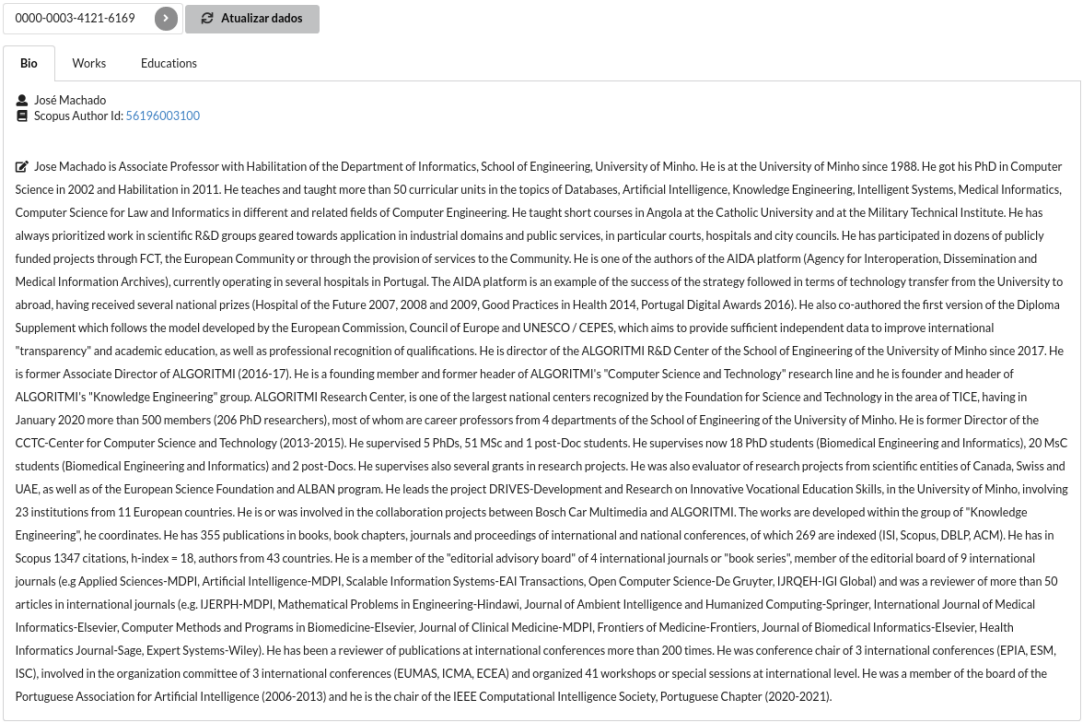

Figura 2: Investigador

5.2 Trabalhos

Mal é feita a pesquisa do utilizador, ficam disponíveis para consulta duas TABs, uma das quais com os seus trabalhos realizados. No fundo desta TAB existe um lista de páginas possíveis de serem consultadas. Para não sobrecarregar a API, a página web faz apenas pedidos de trabalhos 10 a 10 à medida que o utilizador vai pedido movendo-se entre páginas. Assim, como existem investigadores com centenas de trabalhos, esses não são todos carregados de imediato, aumentando a performance da página web, diminuindo a carga na rede e na API.

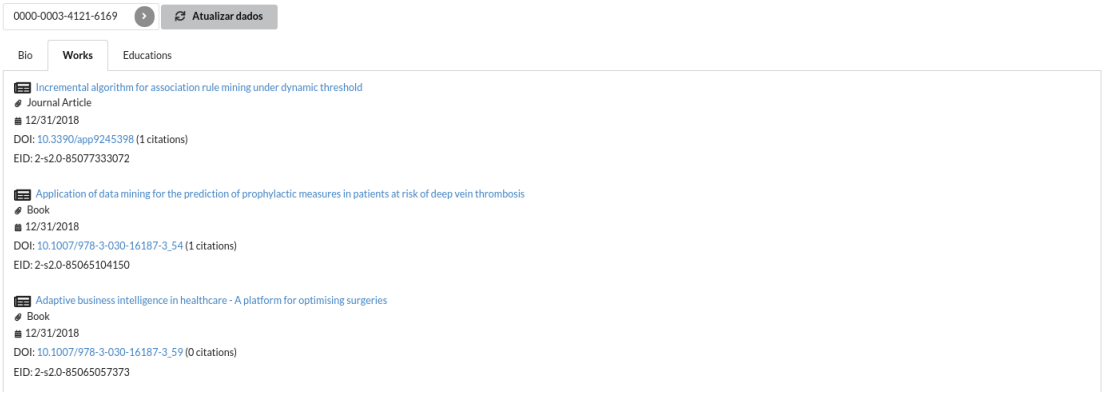

Figura 3: Trabalhos

Cada trabalho apresentado, mostra informações do mesmo, o título e o tipo de publicação, o ano em que esta foi feita, o EID e o DOI caso existam, e, ainda, o número de citações feitas desse trabalho.

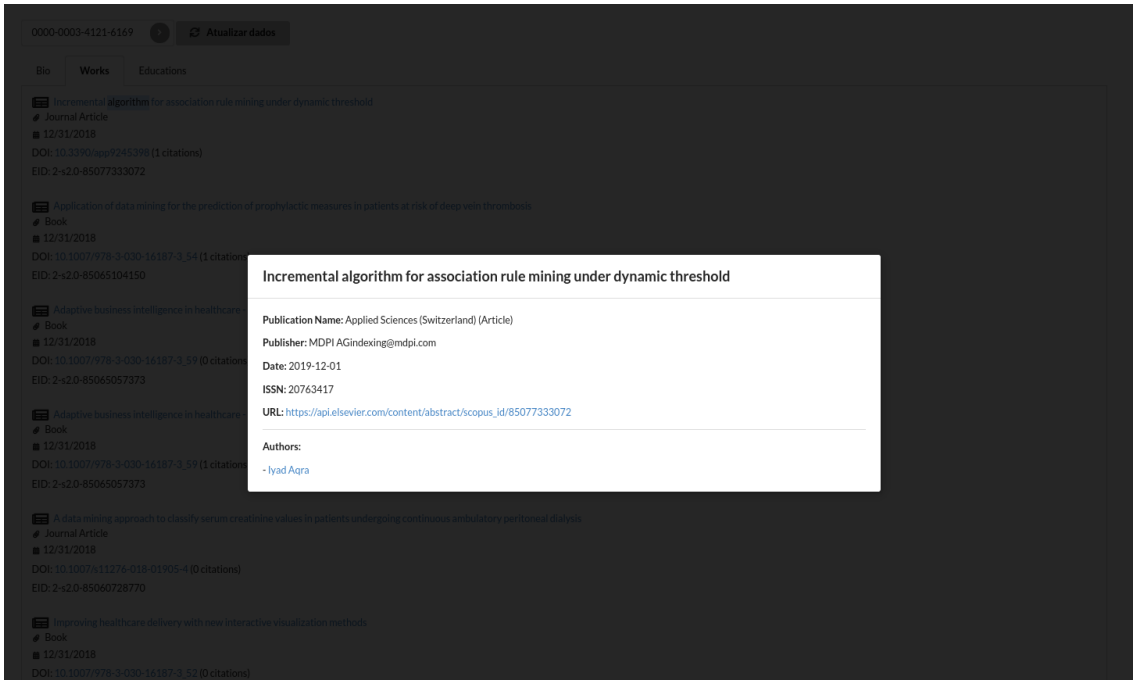

Figura 4: Trabalho

5.3 Educação

Numa outra TAB é também possível ver o percurso académico de um investigador, onde estão os seus graus académicos, local onde foi conseguido, e entre que anos.

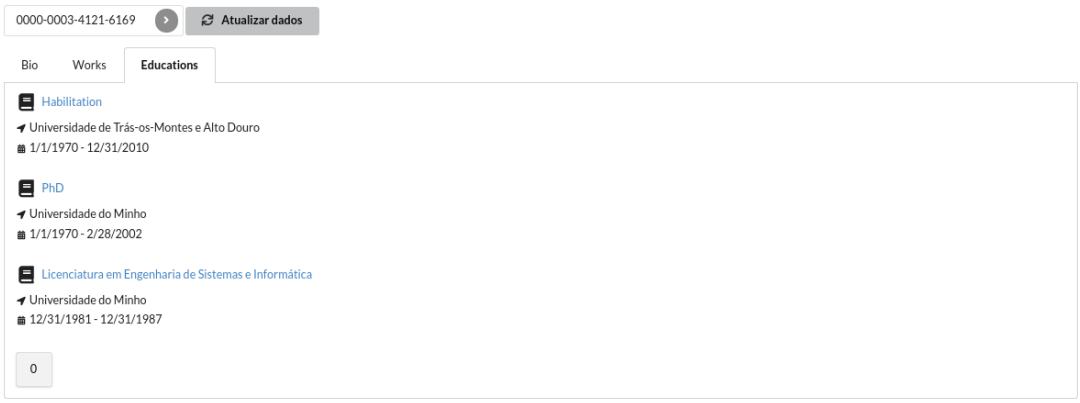

Figura 5: Educação

6 Análise de Desempenho

6.1 Quantidade e Topologia de Dados

Em primeiro lugar, é necessário fazer um análise do tipo de dados que recebemos para armazenar na base de dados e perceber o seu tipo e a sua quantidade de modo a ter o máximo de informação possível no momento de decidir onde e como dedicar tempo e recurso na optimização de processos. Para isso, fez-se uma avaliação sobre dados recolhidos na API da Orcid, sobre vários utilizadores distintos, para recolher a quantidade de dados de cada espécie.

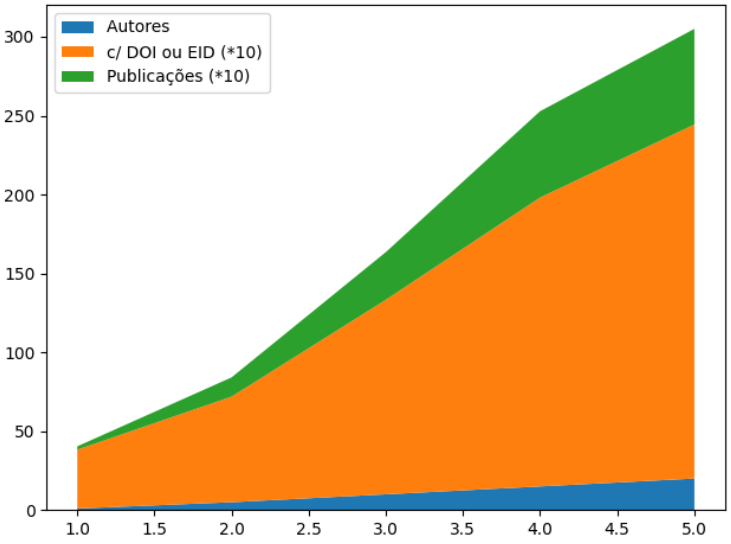

Figura 6: Incremento de Dados

Tal como discutido, feita a recolha de dados, podemos observar que apesar de fazermos queries a apenas 5 utilizadores distintos, a quantidade de publicações que cada um tem, principalmente com DOI e EID (ou seja, vai haver uma recolhar extra de dados sobre estas publicações na API

do Scopus) está cerca de 3 ordem de grandeza superior ao número de autores. Ou seja, para cada autor, existem, grosso modo, cerca de 1000 publicações.

Com esta análise, podemos tirar algumas conclusões. Em primeiro lugar, não vale a pena perder tempo em tentar optimizar as queries sobre informações do utilizadores, o seu número é bastantes reduzido, as informações são mínimas e, simplesmente, o custo/benefício não compensa.

6.2 Latência de Pedidos

Uma das decisões mais importantes e com mais impacto na performance do sistema será decidir como e quando consultar os dados e se faz sentido armazená-los. Para isso, é essencial medir o tempo de consulta aos vários possíveis serviços. Por exemplo, quando o cliente pretende obter todas as pulicações de um utilizador, como é que nós vamos responder a isso? (1) O cliente consulta a API da Orcid para ir buscar as publicações, consulta; (2) O cliente consulta o Backend que faz de intermediário entre o cliente e a API da Orcid?; (3) O cliente faz o pedido ao Backend que lhe responde com os dados armazenados.

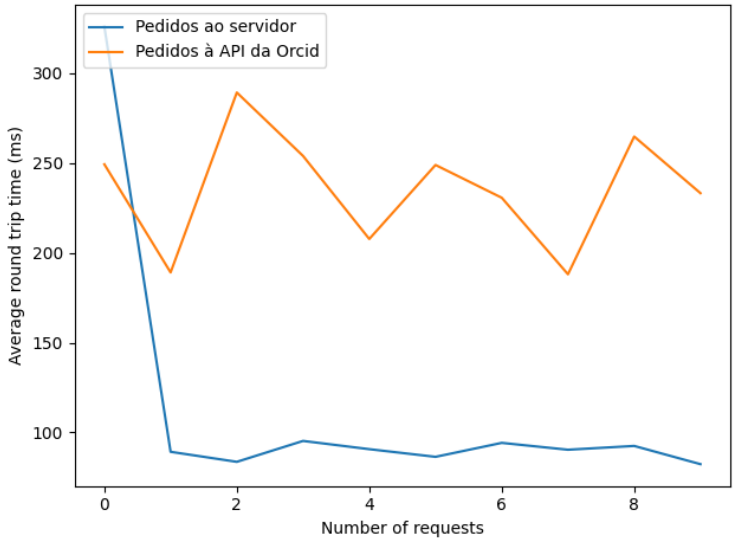

Figura 7: Análise de Latência

Com a análise de latência média para as várias opções, é seguro excluir logo à priori a opção (1), não faz sentido, em termos de usabilidade para o utilizador final, ele ficar sempre à espera cerca de 250ms por cada pedido realizado à API, esta solução não é de todo viável visto que rapidamente se deterioraria a experiência do cliente à medida que cada vez mais e mais complexos dados fossem adicionados ao serviço.

Também não é ideal a opção (2), por causa de se manterem os problemas da solução (1) apesar de ser mais fácil a sua refinação por ser feito do lado do servidor, havendo mais controlo.

A solução adequada é então a solução (3), na qual, o servidor é responsável por armazenar em base de dados as informações eventualmente atualizadas. Ou seja, existe um mecanismo de atualização de dados (discutido anteriormente), em que o cliente é responsável por pedir que os dados guardados no servidor sejam atualizados. A partir disso, todas as queries de leitura desses dados estão armazenadas localmente no servidor, o que permite reduzir a latência do serviço para cerca de 70ms, exceto obviamente no pedido 0, em que é necessário consultar a API da Orcid para recolher os dados.

6.3 Tamanho de mensagens e dos dados transferidos

Com a optimização anterior, é possível responder não só mais rápido ao cliente mas enviando muitos menos dados na rede. Uma vez que do lado do servidor armazenamos todos os dados necessários, se não forem pedidas atualizações de dados, existe uma poupança enorme de recursos de rede, contribuindo para uma maior disponibilidade do sistema para atender cada vez mais clientes, mas também possibilita a poupança de dinheiro em termos de limites de uso da API e em termos de esforço computacional, uma vez que guardar os dados é muito mais barato.

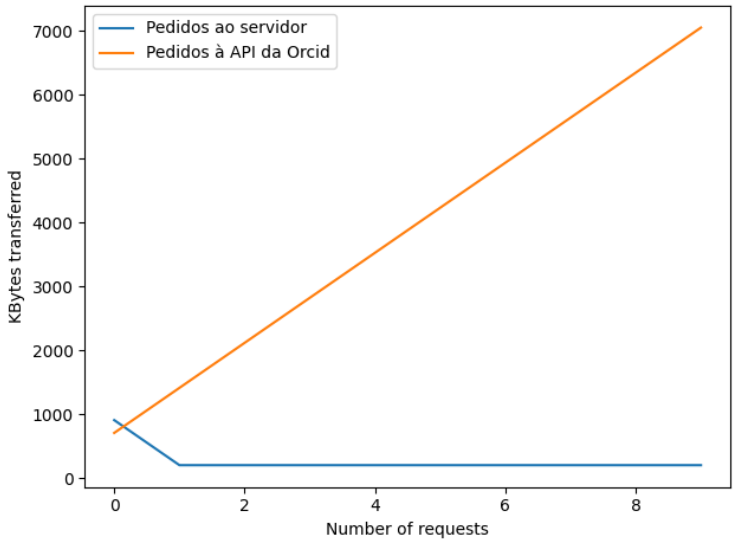

Figura 8: Análise de Transferência de Dados Acumulados

Neste gráfico, podemos então observar a quantidade acumulada em KBytes que se verificaria caso o servidor tivesse constantemente a pedir os dados à API, como podemos verificar, ao fim de 10 pedidos de clientes, a diferença já é avassaladoramente significativa. Esta retirada de carga do servidor permite então uma poupança em termos de recursos disponíveis e em termos monetários muito grandes porque, tal como referido em cima, o pedido dos dados à API, a receção e consequente processamento destes dados e o seu reenvio ao cliente, seria muito mais caros, do que processá-los uma vez e depois só quando necessário, e mantê-los em memória prontos a ser consultados.

7 Conclusão

A facilidade na interoperabilidade semântica entre sistemas que estas APIs representam, tornaram possível de forma rápida e eficiente construir um serviço nelas baseado capaz de consultar todos os dados necessários, fazer o seu processamento e conjugação de modo a produzir dados úteis e com alto valor acrescentado para os clientes do serviço.

Da API da ORCID são por exemplo consultadas as informações pessoais de um determinado utilizador, a sua lista de publicações e o percurso académica, que mais tarde são combinados com os detalhes de cada uma dessas publicações que pode, por exemplo, ser consultado na API da SCOPUS. Esta inter-ajuda permitiu que facilmente fossem alcançados todos os objetivos propostos para este projeto.

Passou-se então para uma fase posterior de análise do sistema e da sua complexidade, da qual foi possível tirar inúmeras ilações e debater sobre as melhores optimizações em termos de custo/benefício que foram posteriormente implementadas para reduzir o custo de manutenção do serviço, da sua operação mas ao mesmo tempo aumentando a sua eficiência e ainda melhorar a experiência do utilizador final.

8 Bibliografia

Referências

[1] REST
https://en.wikipedia.org/wiki/REST

[2] Knex.js - A SQL Query Builder for Javascript
http://knexjs.org/

[3] Objection.js - An SQL-friendly ORM for Node.js
https://vincit.github.io/objection.js/

[4] TypeScript Docs
https://www.typescriptlang.org/docs/home.html