Politécnico do Porto
Instituto Superior de Engenharia do Porto

Developing applications with Ballerina

Tiago Nora

LETI
Licenciatura em Engenharia de Telecomunicações e Informática

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Março, 2023

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Projeto/Estágio, do 3º ano, da Licenciatura em Engenharia de Telecomunicações e Informática.

Candidato: Tiago Nora, Nº 1201050, 1201050@isep.ipp.pt

Orientação Científica: Isabel Azevedo, ifp@isep.ipp.pt

Empresa: GILT - Games, Interaction and Learning Technologies

Orientador: Isabel Azevedo, ifp@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Março, 2023

(Opcional) Poderá usar esta secção para dedicar o trabalho a alguém...

Agradecimentos

(Opcional) Agradecimentos que sejam devidos...

Resumo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean ultricies mauris ut auctor consequat. Aliquam quis viverra risus. Duis pulvinar vehicula gravida. Etiam pellentesque est leo, a porta velit lobortis a. Donec pretium ante augue, in tincidunt augue vestibulum at. Integer libero urna, vehicula in egestas sit amet, porttitor sit amet nunc. Curabitur non congue urna. Nulla fringilla tellus quis cursus blandit. Etiam iaculis viverra libero sit amet ultrices. Nullam eget eleifend risus. Cras porttitor at ligula non sagittis. Fusce sit amet neque quis ex rhoncus sodales. Fusce luctus turpis ex, ut interdum massa hendrerit in. Vestibulum ultricies dolor diam, eu consectetur est dapibus nec. Integer interdum ex sed urna faucibus aliquet.

Palavras-Chave: Lista, separada por vírgulas, de palavras, frases, ou acrónimos chave no âmbito do trabalho descrito neste texto.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean ultricies mauris ut auctor consequat. Aliquam quis viverra risus. Duis pulvinar vehicula gravida. Etiam pellentesque est leo, a porta velit lobortis a. Donec pretium ante augue, in tincidunt augue vestibulum at. Integer libero urna, vehicula in egestas sit amet, porttitor sit amet nunc. Curabitur non congue urna. Nulla fringilla tellus quis cursus blandit. Etiam iaculis viverra libero sit amet ultrices. Nullam eget eleifend risus. Cras porttitor at ligula non sagittis. Fusce sit amet neque quis ex rhoncus sodales. Fusce luctus turpis ex, ut interdum massa hendrerit in. Vestibulum ultricies dolor diam, eu consectetur est dapibus nec. Integer interdum ex sed urna faucibus aliquet.

Keywords: Comma separated list of words, phrases, or key acronyms within the scope of your developed work.

Índice

Lista de Figuras

Lista de Tabelas

Listagens

Lista de Acrónimos

CQRS	*Command Query Responsibility Segregation*
CRUD	*Create, Read, Update and Detroy*
gRPC	*Google Remote Procedure Call*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
MariaDB	Maria Database
MySQL	My Structured Query Language
OAuth2	Open Authorization 2.0
RabbitMQ	*Rabbit Message Queue*
REST	*Representational State Transfer*
RPC	*Remote Procedure Call*
SOAP	*Simple Object Access Protocol*
TCP	Transmission Control Protocol
XML	*Extensible Markup Language*
YAML	*YAML ain't markup language*

Capítulo 1

Introdução

1.1 Contextualização

1.2 Descrição do Projeto

1.2.1 Objetivos

1.3 Calendarização

1.4 Organização do Relatório

Capítulo 2

Introdução teórica

Neste capitulo é descrito e apresentado o estado da arte em relação à presente matéria em estudo, considerando os termos e conceitos que estão relacionados com a mesma, refletindo no que diz respeito as suas qualidades, importância, imperfeições e caraterísticas.

2.1 Arquiteturas de software

A arquitetura de software é um termo que é usado para descrever as caraterísticas, estrutura e comportamento dos componentes de software. As quatro principais arquiteturas de software são: arquitetura monolítica, arquitetura orientada a serviços, arquitetura de microsserviços e arquitetura orientada a eventos. A arquitetura monolítica é a mais antiga, passando pela arquitetura orientada a serviços e acabando pelas duas restantes, respetivamente em ordem cronológica. Cada uma das arquiteturas segue uma abordagem diferente, apesar que, a arquitetura de microsserviços e arquitetura orientada a eventos podem coexistir na mesma aplicação.

2.1.1 Arquitetura Monolítica

Na arquitetura monolítica, o código e as suas funcionalidades estão contidas e consolidas em uma única aplicação. Este método de desenvolvimento de software é indicado para pequenos projetos, dado que, torna-se complicado escalar e manter em grandes projetos, devido as constantes mudanças dos mesmos e outros fatores. A arquitetura monolítica tem várias vantagens, tais como um desenvolvimento inicial

rápido, custos de desenvolvimento mais baixos, e testes de funcionalidade apenas num local. Além disso, a comunicação entre módulos é facilitada e ocupa menos espaço. Contudo, as suas desvantagens incluem manutenção difícil da funcionalidades, escalabilidade comprometida, dificuldade em fazer alterações ao código, existência de um único ponto de erro, e pouca flexibilidade.

2.1.2 Arquitetura orientada a serviços

A arquitetura orientada a serviços é uma evolução da arquitetura monolítica, pois divide a aplicação em serviços reutilizáveis que podem ser excetuados e implementados independentemente uns dos outros. Sendo que, estes podem estar conectados entre si, com recurso a protocolos de comunicação. A escalabilidade é facilitada pela arquitetura, pois ela possibilita o baixo acoplamento entre os componentes, o que facilita a manutenção, reduz os custos e a complexidade.

2.1.3 Arquitetura de microsserviços

No caso, da arquitetura de microsserviços, este divide o sistema em pequenas aplicações, independentes umas das outras, ou seja, cada uma pode ser implementada e gerida individualmente, a essas aplicações dá se o nome de microsserviços. Hoje em dia, esta arquitetura é muito utilizada devido ao crescente numero de utilizadores, aumento de tráfego necessário e diferentes localizações desses mesmos usuários.

As mais importantes caraterísticas de uma arquitetura de microsserviços são: a escalabilidade e a elasticidade. Sendo que, escalabilidade refere-se à capacidade de um sistema conseguir ajustar se ao crescimento no número de usuários ou no volume de dados sem comprometer o desempenho ou a disponibilidade. A escalabilidade pode ser alcançada sem interromper o normal funcionamento do serviço através de processos de escalabilidade horizontal, adicionar mais nós, ou vertical, aumentar a capacidade de processamento ou armazenamento de um ou mais nós. Por outro lado, a elasticidade refere-se à capacidade de um sistema conseguir ajustar-se com as variações no tráfego ou carga de forma automática, aumentando ou diminuindo a alocação de recursos para algum serviço conforme os requisitos no momento. Normalmente esta gestão de alocar automaticamente recursos é feita através de containers.

Normalmente esta arquitetura está associada a padrões de software como por exemplo *Command Query Responsibility Segregation* (CQRS) (Segregação de Responsabilidade de Comando e Consulta), onde as responsabilidades da aplicação são divididos em dois blocos, o bloco das operações que modificam dados e o bloco que apresenta os dados da aplicação e *Database Per Service* (base de dados por serviço), em que cada aplicação contem uma base de dados, com os dados estritamente necessários para o seu correto funcionamento e funciona como um tipo de *Backup*, dado a redundância de dados.

A arquitetura de microsserviços apresenta flexibilidade em relação a modificações, dado o seu baixo acoplamento entre módulos e previne o colapso total do sistema no caso de erro ou falha. No entanto, isso exige um trabalho extra na implementação da comunicação entre serviços, o que pode causar uma complexibilidade na implementação e construção dos serviços, como também na própria implementação dos serviços devido a sua quantidade.

2.1.4 Arquitetura orientada a eventos

Numa arquitetura orientada a eventos, os diferentes componentes do sistema, comunicam entre si com recurso a eventos, eventos esses que podem ser alterações ou criação de elementos da lógica de negocio. Esta arquitetura recorre ao modelo de publicação e subscrição, onde os eventos são enviados através dos publicadores e recebidos pelos subscritores que subscreveram esse mesmo tipo de evento.

2.2 Comunicação entre serviços

Numa arquitetura que tem várias aplicações muita vezes é preciso fazer passar dados entre elas ou mesmo verificar a existência das mesma e por isso tem se que recorrer os vários tipos de comunicações, como por exemplo, comunicação síncrona e comunicação assíncrona.

2.2.1 Comunicação síncrona

A comunicação síncrona é denominada de comunicação bloqueante, dado que esta, está sempre a espera de uma resposta por parte do recetor do pedido, independentemente do tipo de dados seja ele uma mensagem de erro como uma mensagem a confirmar a criação de algum tipo de dado e os respetivos atributos. Exemplos desse tipo de comunicação são:

- *Representational State Transfer* (REST)
- *Google Remote Procedure Call* (gRPC)
- *Remote Procedure Call* (RPC)
- *Simple Object Access Protocol* (SOAP)

Começando com o RPC, dado que é o mais antigo deles todos, neste caso a comunicação é feita através de procedimento remoto entre servidor e cliente. Visto que o RPC é uma comunicação de alto nível, esta facilita a implementação da mesma pois a comunicação ao nível do protocolo Transmission Control Protocol (TCP) é escondida, fazendo com que a performance seja maior. Sendo que é feita uma chamada

de procedimento remoto é necessário voltar a desenvolver ou escrever código, mas o esforço necessário é mínimo, devido que se pode reutilizar funcionalidades.

Passando ao SOAP, este já é um protocolo especifico para comunicação de dados estruturados, com a limitação de só poder receber dados do tipo *Extensible Markup Language* (XML). Comparando com RPC, este apresenta um complexidade maior na sua arquitetura.

No que diz respeito ao gRPC, este é uma evolução do RPC, de outro modo, uma *framework*(ferramenta que auxilia na desenvolvimento de aplicação visto já conter código já definido) do RPC tendo sido desenvolvido para combater alguns dos problemas do seu antecessor. A empresa GOOGLE, a empresa que a desenvolveu, decidiu implementar, *Protocol Buffers*(descreve a estrutura dos dados) para serializar informação e fazendo assim que microsserviços consigam comunicar entre si com auxilio desta ferramenta.

Através de REST, o servidor disponibiliza ao cliente uma interface de operações bem definidas, com ajuda do protocolo HTTP. Normalmente são usadas operações *Create, Read, Update and Detroy* (CRUD), ou seja, operações de criação, leitura, atualização e destruição. Uma das vantagens é que o servidor permite que o cliente acesse as funcionalidades sem que este tenha conhecimento da implementação da mesma.

2.2.2 Comunicação assíncrona

A comunicação assíncrona é denominada de uma comunicação não bloqueante, sendo que a aplicação não fica a espera de uma resposta imediata e prossegue com o normal funcionamento do programa. Exemplos desse tipo de comunicação são:

- *Rabbit Message Queue* (RabbitMQ)
- Apache Kafka

Sendo o RabbitMQ um método de comunicação assíncrona, a sua utilização é normalmente associada a uma arquitetura orientada a eventos, sendo que, a esta pode também estar associada a uma arquitetura de microsserviços. Neste método a mensagem pode ser propagada ou enviada para um consumidor em especifico. No caso de ser enviada para vários remetentes, a mensagem parte de um produtor, no qual, envia para uma *exchange*, que por sua vez vai contar o número de serviços que subscreveram aquele tipo de mensagem, cria *queues* respondente ao número contado com a informação enviada pelo produtor, os consumidores detetam uma nova entrada na *queue*, consumem a e tratam a de seguida. Por outro lado se a mensagem for enviada em que só vai haver um remetente, o processo é semelhante, a mensagem parte do consumidor é enviado para a *exchange*, a *exchange* cria só uma *queue* com a informação, o consumidor deteta a nova entrada na *queue*, consume a e trata a de seguida.

2.3 Tecnologias

Introdução as tecnologias em estudo

2.3.1 Gestor de contentores

Exemplos de gestores de contentores são:

- Podman
- Docker

2.3.2 Formato de troca de dados

Exemplos de formatos de troca de dados são:

- XML
- *YAML ain't markup language* (YAML)
- *JavaScript Object Notation* (JSON)

2.3.3 Linguagem de programação

ballerina nodejs python

2.3.4 Autenticação

Exemplos de formatos de autenticação são:

- *JSON Web Token* (JWT)
- Open Authorization 2.0 (OAuth2)

2.3.5 Base de dados

Alguns exemplos de base de dados são:

- My Structured Query Language (MySQL)
- Maria Database (MariaDB)

2.3.6 Cliente web

Alguns exemplos de clientes web são:

- Postman
- Insomnia

2.3.7 Ferramenta de teste de performance

jmeter

Capítulo 3

Conclusões

3.1 Trabalho Futuro

Referências

Anexo A

Título do Anexo

A.1 Secção

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

A.2 Mais uma secção do Anexo A

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida

placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.