Gestão de Propostas e Projetos Curriculares

Integração de Sistemas

Mestrado Engenharia Informática

Sónia Ferreira - 1161491 | Sérgio Ferreira – 1161393| Vera Dias – 1160941

Abstract

Este documento está desenvolvido dentro do escopo da unidade curricular de Integração de sistemas do Mestrado de Engenharia Informática (MEI).
O objetivo deste projeto é integrar sistemas de várias camadas. A primeira camada foi implementada em BPMN, a segunda camada foi em ESB e finalmente foram implementados um conjunto de microserviços na nuvem com escalabilidade e elasticidade.

Keywords: WSO2, NodeJS, Java, Microservices, Kubernetes, Cloud

Índice

1 Introdução

Este documento descreve a elaboração de um projeto que gere as provas de tese de mestrado.

Este capítulo tem como objetivo apresentar o contexto do projeto, o problema e a estrutura do documento.

1.1 Contexto

O projeto foi desenvolvido dentro do escopo da unidade curricular de Integração de Sistemas do Mestrado de Engenharia Informática no ISEP. O objetivo deste projeto é produzir um *software* que suporte o processo de gestão de propostas e projetos curriculares.

Na primeira iteração foi necessário delinear todos os processos de negócio em BPMN, na segunda foi necessário alterar todos os endereços que são chamados nos serviços de BPMN serem reformulados e finalmente na terceira iteração foram feitos os microserviços que criam os objetos que são usados nos processos do BPMN.

1.2 Estrutura do documento

Neste relatório estão definidas todas as iterações do projeto. Em cada iteração é definido os objetivos e uma breve demonstração do que foi feito. Na última iteração é definido as linguagens que foram usadas, o que foi implementado e algumas decisões arquiteturais que foram feitas.

2 Primeira Iteração

Nesta secção é definido um resumo dos objetivos e uma demonstração do que foi feito em BPMN.

2.1 Objetivos

De acordo com o enunciado disponibilizado, existem vários processos de negócio para suportar o processo de gestão de propostas e projetos curriculares. O primeiro processo é o processo de **gestão de proposta**, em que um proponente insere uma proposta que deve ser revista pelo responsável pela cadeira curricular ou RUC, dependendo da revisão é feito um conjunto de decisões. Depois disto é iniciado o processo de **aceitação de proposta**, o estudante começa por selecionar a proposta e solicita ao proponente a sua atribuição e solicita também a orientação de um docente. Quando o docente aceita o pedido de orientação é inicializado o processo de **formalização de proposta**, este é espoletado pelo estudante preenchendo um formulário, a formalização é enviada ao orientador, que deve aceitar e ao RUC que deve definir docente revisor. Os docentes revisores devem aceitar o convite e aceitar a formalização. Finalmente é inicializado o processo de **gestão de prova**, que inicia quando o estudante submete o relatório. Neste processo, o RUC define arguentes e também deve ser definida a data e local da prova. No final o RUC introduz uma nota atribuída e notifica o estudante.

Nesta iteração, o objetivo é que todos estes processos sejam definidos e implementados usando o WSO2 e o BPMN.

2.2 Demonstração

Esta iteração foi atingida de forma bem-sucedida e todos os objetivos foram implementados.
Nesta secção é feita uma breve demonstração de alguns dos processos que foram implementados. A seguinte figura representa o diagrama de negócio aceitação de proposta.

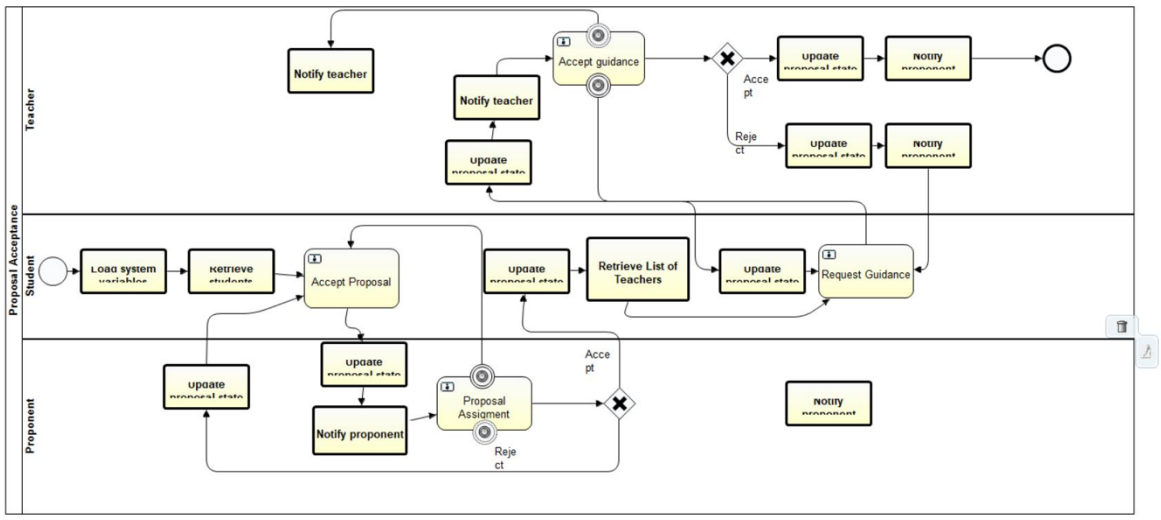

Figura 1 - Aceitação de proposta BPMN.

Também foi definido o diagrama de negócio que representa a **gestão de proposta** representado pela seguinte figura.

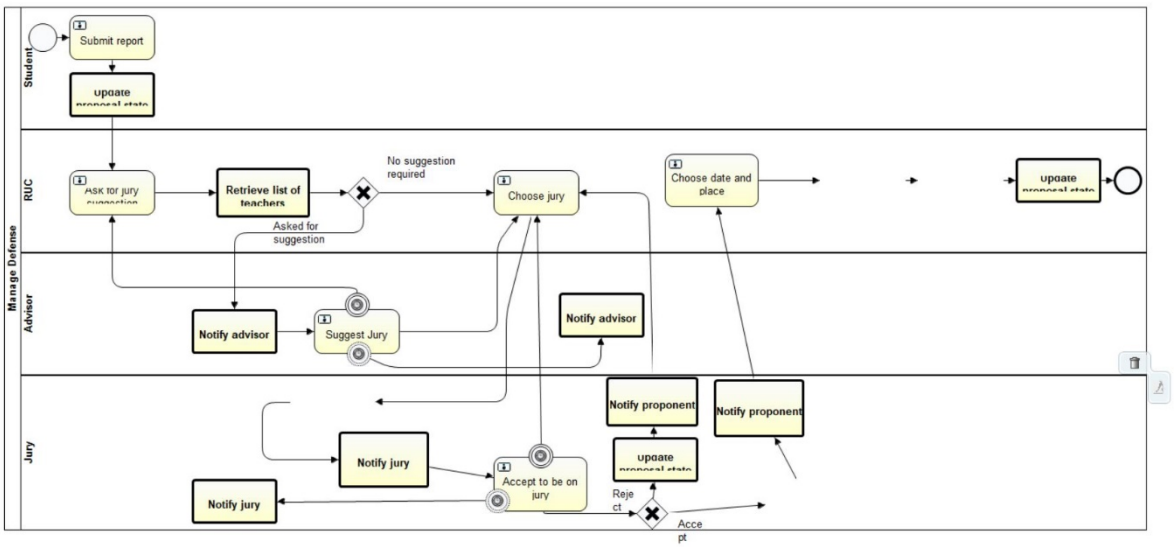

Figura 2 - Gestão de prova BPMN

Outros processos que foram implementados nesta iteração estão definidos nos diretórios que começam por **p1** no diretório que contém todo o código fonte deste projeto.

3 Segunda Iteração

Nesta secção será apresentado os objetivos delineados para a segunda iteração e algumas implementações feitas usando ESB.

3.1 Objetivos

Na segunda iteração foi necessário construir sobre o projeto feito previamente. Neste momento foi necessário convergir diferentes pedidos REST ou SOAP em objetos que alimentam os processos de negócio definidos em cima. O que na primeira iteração usava URLs de pedidos *Mock* agora é substituído por endereços disponibilizados pelo ESB.

A representação da ligação entre os componentes da iteração 1 e 2 está representada pela seguinte Figura 3.

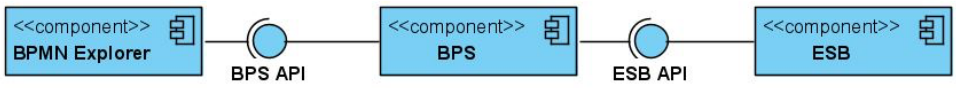

Figura 3 - Diagrama de Componentes

3.2 Demonstração

Nesta secção é demonstrada algumas das funcionalidades implementadas na segunda iteração no ESB. A Figura 4 contém a implementação de um dos pedidos que produz objetos do tipo palavra-chave que foi disponibilizado para ser consumidos pela implementação em BPS.

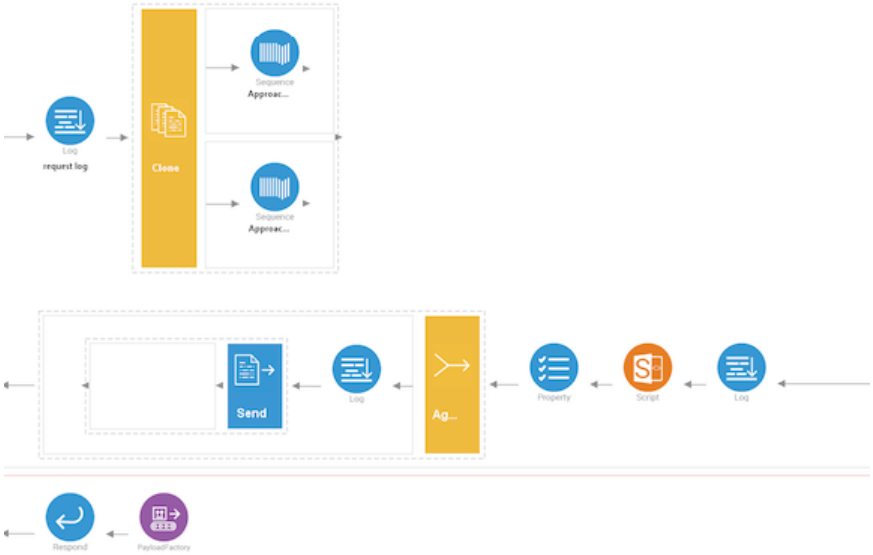

Figura 4 - Implementação de um pedido que produz palavras-chave.

4 Terceira Iteração

Nesta terceira e última iteração é definido todos os componentes que foram feitos, nomeadamente, como foi feita a implementação de microserviços, como é que foi feita a implementação em *Kubernetes*, como é que foi feita a implementação na nuvem e o que foi feito usando o serviço de API Management.

4.1 Microserviços

Na terceira iteração era expectável que certos componentes deste *software* fossem implementados numa arquitetura orientada a microserviços. De acordo com Martin Fowler, a arquitetura orientada a microserviços é uma abordagem de desenvolvimento de uma aplicação única com uma construção de pequenos serviços, cada um destes executa os seus conjuntos dos seus próprios processos e comunicação com mecanismos leves (Fowler, s.d.).

Idealmente, como não foi implementado nada previamente, a decisão lógica seria decompor por funcionalidades de negócio. A partir disto foi necessário também definir as comunicações que seriam feitas entre os microserviços, que neste caso devem ser feitas de forma assíncrona.

Também foi necessário aplicar CQRS aos microserviços, isto significa que os pedidos de escrita devem estar num microserviços diferente dos pedidos de leitura, ou seja, para cada microserviços existe a componente *Query* e a componente *Command*. Este modelo é muito benéfico pois é indicado para modelos de negócio em que o número de escritas e o número de leituras é distinto e esta divisão permite uma maior escalabilidade.

Para cada microserviços de componente *Query* definido existe uma base de dados por serviço, as linguagens usadas para implementar os microserviços foram Java e NodeJS e para base de dados foi Postgresql e MongoDB respetivamente.

O uso de mensagens assíncronas necessita da adoção de um padrão *publish/ subscribe*. Isto é benéfico pois permite o desacoplamento entre *publisher* e *consumer*.

A comunicação entre os microserviços é feita por partilha de eventos através do uso de um *Message Broker* (MB). O MB usado neste trabalho foi *RabbitMQ*.

A ideia principal no modelo de mensagens do *RabbitMQ* é que o *producer* nunca envia mensagens diretamente para uma *queue,* mas sim para uma *Exchange*. A *Exchange* recebe as mensagens do *producer* e distribui as mesmas para as *queues* segundo as regras definidas pelo tipo de *Exchange*.

Finalmente, o tipo de *Exchange* que foi usada foi do tipo *fannout*, que permite que a mesma mensagem seja distribuída por várias *queues* em simultâneo e consequentemente que vários *consumers* possam consumir a mesma.

A seguinte Figura 5 representa todos os componentes definidos por uma arquitetura de microserviços nesta iteração.

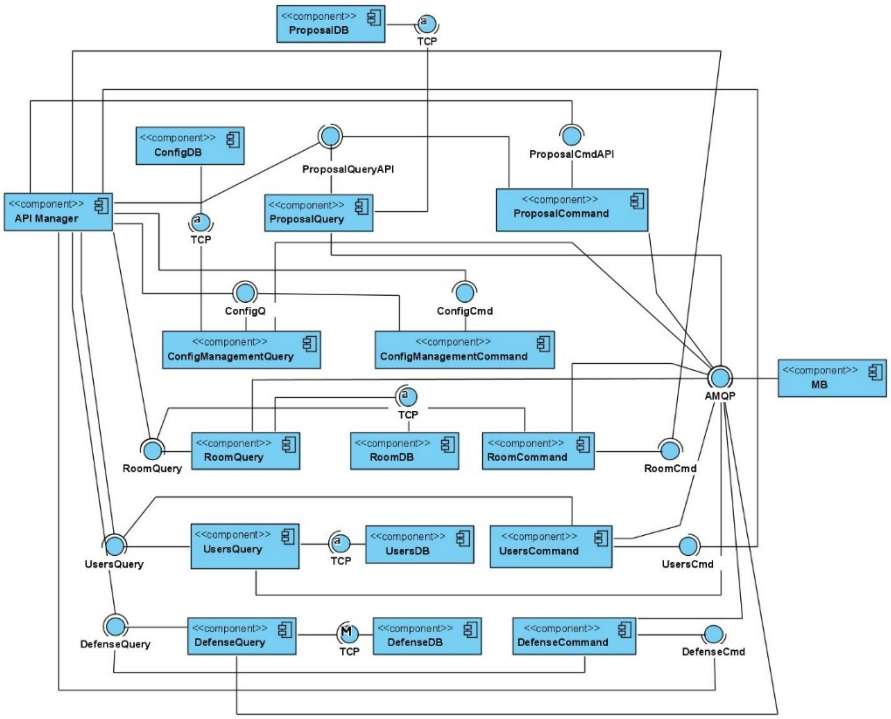

Figura 5 - Diagrama de Componentes

4.2 *Deployment*

No *deploy* dos microservicos em Kubernetes, foi usado principalmente os objectos *Deployment* (para definir pods e as suas propriedades), *Service* (para expor microservicos para fora do cluster) e *ConfigMap* para definir valores que foram posteriormente expostos aos pods como variáveis de ambiente. Também foi usado um *Horizontal Pod Autoscaler (HPA)* para escalar o número de *pods* por microserviço com base na utilização de CPU de cada microserviço individual. No entanto faltou tempo para implementar HPA para todos os microserviços.

Em termos de infraestrutura utilizamos GKE na *Google Cloud Platform* para lançar um pequeno *cluster* com 3 *nodes*.

Os serviços estão expostos via *NodePort*, em que é aberta uma porta alta nos nós do *cluster*. Caso a equipa tivesse mais tempo poderia ter sido usado o tipo *LoadBalancer* e aceder ao microserviço de entrada via *LoadBalancer* lançado no GCP o que tornaria o acesso aos microserviços mais simples e aumentaria a segurança do próprio cluster (devido a não ser necessário ter portas abertas para a internet).

Em termos de melhorias a efectuar, destaca-se a questão de mudar os serviços de *NodePort* para *LoadBalancer* de forma a poder fechar as portas abertas nos nos do cluster, e a implementação do *HorizontalPodAutoscaler* nos restantes microserviços de forma a garantir escalabilidade em face a cargas elevadas.

De igual forma seria ideal implementar volumes persistentes nas bases de dados, algo que a equipa não teve tempo para implementar.

Na seguinte figura é possível verificar o diagrama de implantação da aplicação. O componente *API Manager* comunica com todos os microserviços através de uma API Gateway. Todos os serviços do tipo *Query* têm a sua própria base de dados (em Postgres ou MongoDB). O *Message Broker* escolhido para implantação foi o *RabbitMQ*, que comunica através do protocolo AMQP com todos os microserviços desenvolvidos.

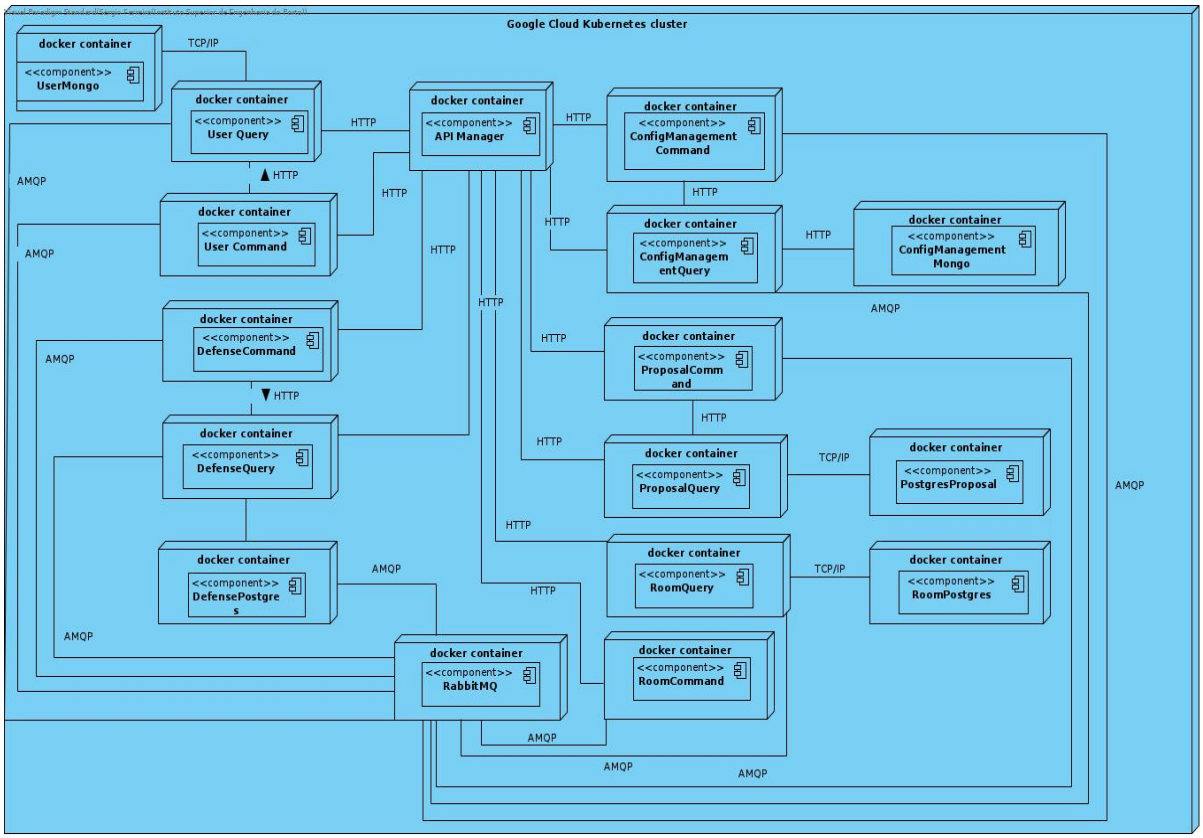

Figura 6 - Diagrama de Implantação.

4.3 WSO2 API Management

Nesta iteração foi usado o WSO2 API Manager como gateway de entrada para os microserviços. Sendo este o componente de entrada, é também ele responsável por garantir a autenticação no acesso a todos os endpoints de todos os microserviços.

Para cada API adicionada no API Manager é possível escolher o plano de negócio que nos permite indicar o número de requests/ms. Outra forma de limitar o número de pedido é através das subscrições, pois ao gerar um token de acesso é possível indicar uma quota de requests por token de acesso que é partilhada por todos os subscritores do mesmo.

Permite integrar os vários microserviços e expô-los como um único serviço conforme podemos ver na imagem abaixo, permitindo escolher os recursos a agrupar.

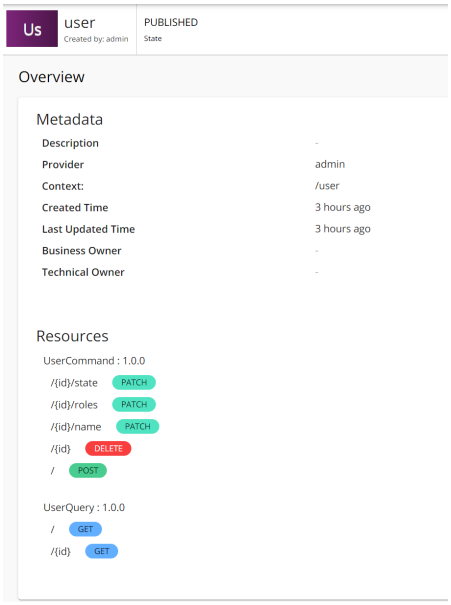

Figura 7 - API product no WSO2 API Management

Permite importar a documentação Swagger criada nos microserviços Spring Boot ou gerar a mesma.

5 Conclusão

Este capítulo encerra este relatório resumindo brevemente o que foi feito em todas as iterações, melhorias e a conclusão final.

5.1 Melhorias

Embora a primeira e segunda iteração terem sido bem-sucedidas existe vários aspetos que podem ser melhorados na terceira iteração, nomeadamente:

- Melhoria dos microserviços, o microserviços gestão de sala e gestão de prova;
- Integração com o projeto das iterações anteriores;
- Adicionar testes de carga;
- Adicionar análise a partir do *API Management*;
- Fazer testes com *JMeter*;
- Adicionar o requisito de negócio de ao fim de três propostas rejeitadas ou canceladas o utilizador deve ficar inativo durante 4 meses.

5.2 Conclusão Final

De uma forma geral este projeto foi implementado de uma forma bem-sucedida. O grupo trabalhou bastante bem em equipa e demonstrou conhecimento durante as várias entregas. No entanto, a terceira iteração necessita de algumas melhorias que não foram atingidas por falta de tempo.

6 Bibliography

Fowler, M. (n.d.). *Microservices*. Retrieved Junho 20, 2021, from https://martinfowler.com/articles/microservices.html