INSIS

Hugo Vinhal, *1161007*

José Mota, *1161263*

Parte 1

Modelação de processos

De forma a cumprir os requisitos enunciados no projeto 1, foi realizada a modelação de todo o processo de negócio a partir da utilização do *Business Process Server* (BPS), configurado por BPMN.

Posto isto, o processo foi repartido em 4 grandes principais processos:

- Gestão de proposta – figura 1.
- Aceitação de proposta – figura 2.
- Gestão de formalização – figura 3.
- Gestão de prova – figura 4.

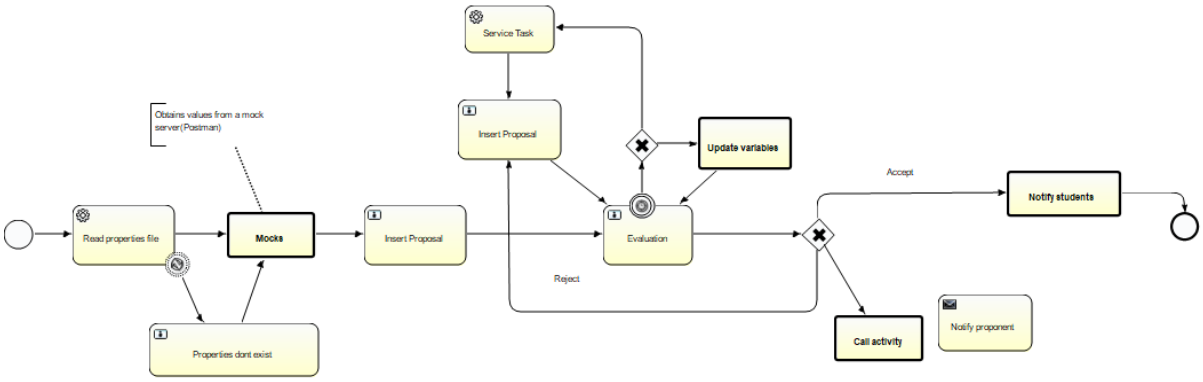

Figura 1 - Gestão de proposta

O diagrama é o ponto de entrada no processo geral, onde o utilizador escolhe os diferentes atributos necessários para criar a proposta. O proponente (docente ou externo, e, portanto, não pode ser estudante) introduz o título, problema, objetivos, palavras-chave, lista de abordagens e de tecnologias prováveis. O RUC aceita, rejeita ou cancela a proposta. Em caso de aceitação, é publicada e enviada notificação a estudante. Se rejeitada, permite ao proponente rever e ressubmeter. Se cancelada, o proponente é notificado com o comentário à decisão, mas não é possível rever e ressubmeter tal proposta.

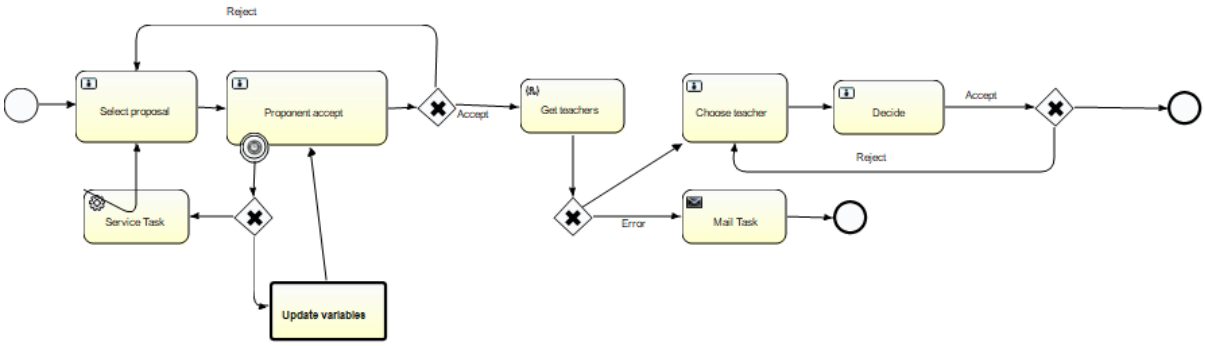

Figura 2 - Aceitação de proposta

Os estudantes selecionam a proposta, solicitando ao proponente a atribuição, se e quando receber atribuição solicita orientação de docente.

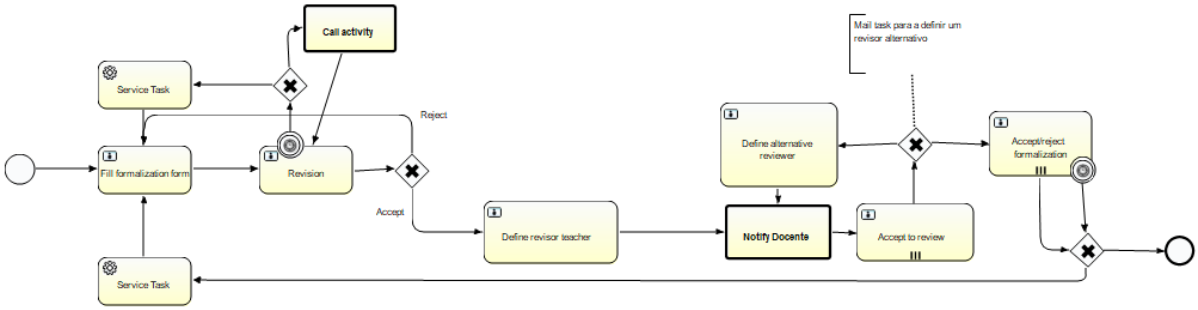

Figura 3 - Gestão de formalização

Iniciado apoós o estudante preencher o formulário, sendo revista pelo orientador. De acordo com as revisões o fluxo varia, em caso de aceitação notifica os docentes para rever o formulário, estes podem aceitar ou recusar.

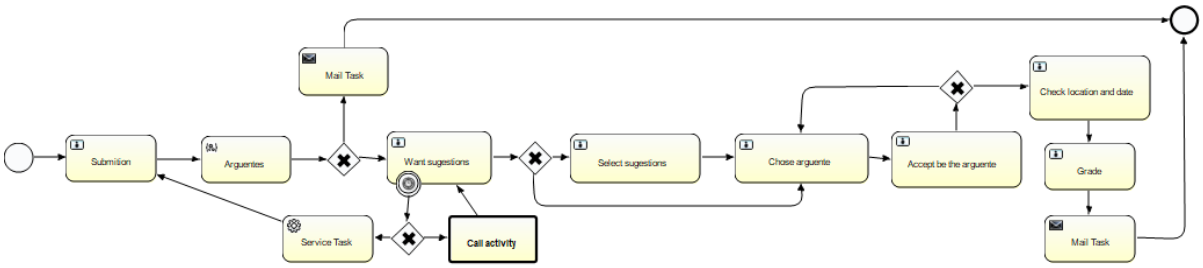

Figura 4 - Gestão de prova

A submissão pelo estudante despoleta o processo, o RUC deve escolher utilizadores com o cargo de docentes para ser arguente. Com o fim do processo é indicada a nota e enviada ao estudante.

Para além dos diagramas apresentados foram desenvolvidos alguns com o propósito de reutilização e/ou tornar mais legível os diagramas.

Parte 2

A utilização do *Integration Studio* permitiu modelar novos endpoints de acordo com as necessidades e os recursos disponibilizados, tanto em HTTP como em SOAP. Apesar de existirem mais e diversos *resources* apenas estão demonstrados alguns, dada a repetição de alguns dos processos.

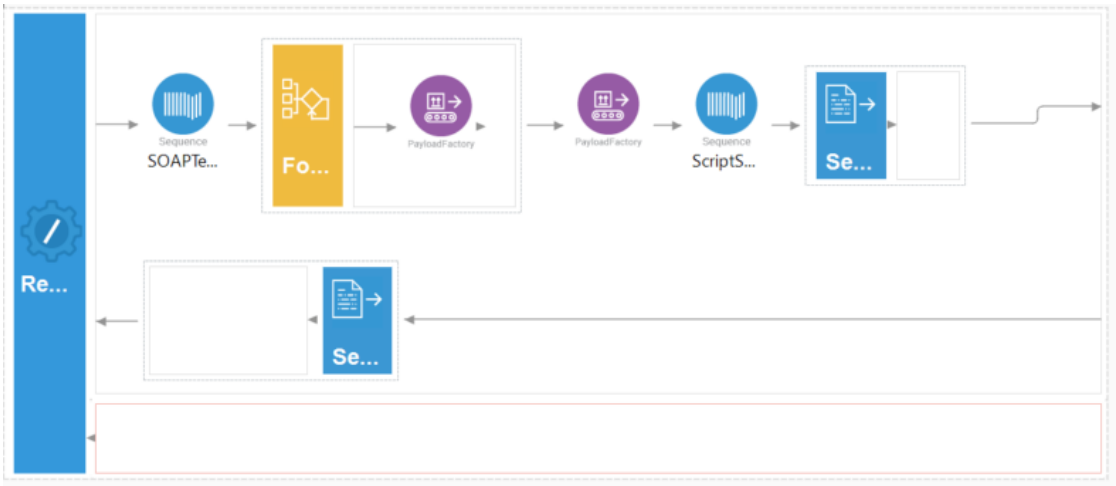

Figura 5 - Endpoint Temas

O *endpoint* representado acima, procura apresentar uma comunicação de apenas um pedido SOAP e extrair os dados relevantes de Temas, após a extração utiliza uma sequência de forma a formatar e retirar dados repetidos.

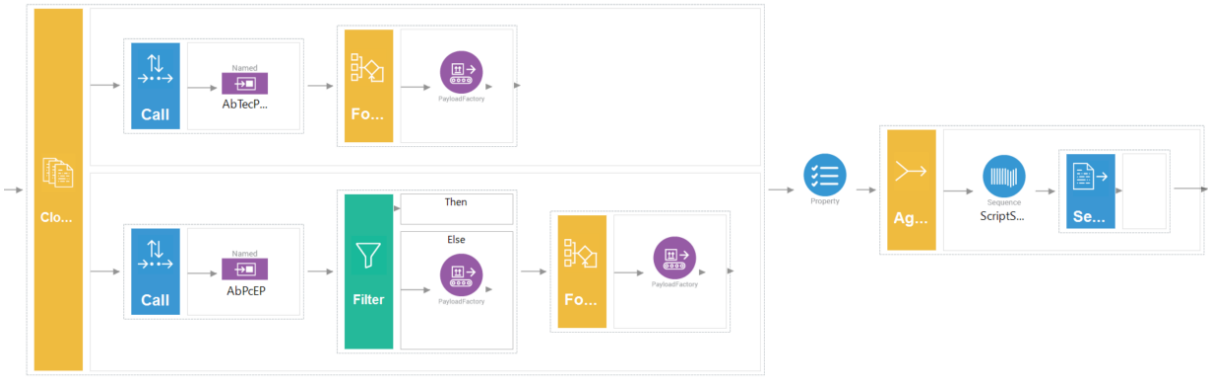

Figura 6 - Endpoint abordagens

Desenvolvido de forma a alcançar dois *endpoints* REST, obter a informação pertinente, no segundo caso verifica se o pedido obteve alguma resposta, em caso de negação contém informação para alimentar a sequência.

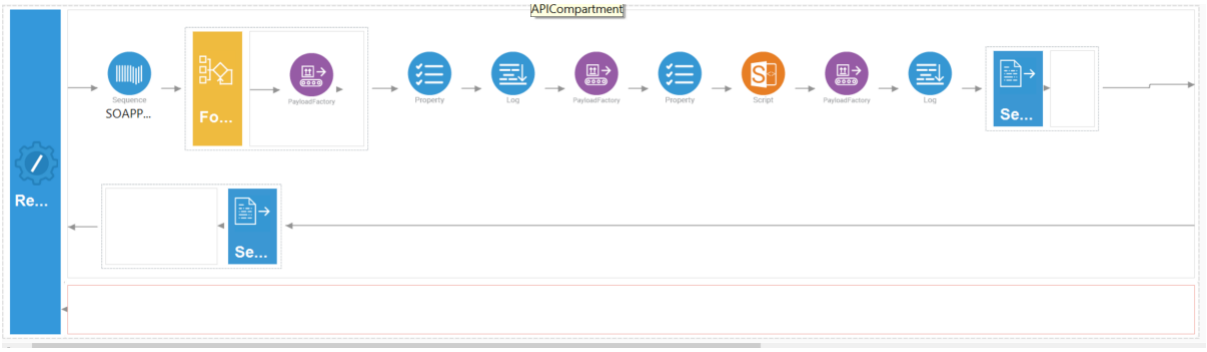

Figura 7 - Endpoint Filtro

Novamente é realiza um pedido inicialmente, aos quais os dados serão processados, obtendo o filtro introduzido no pedido "/filtro/{open}" é possível na *script task* obter apenas a informação de acordo com o estado.

Parte 3

Linguagem

De acordo com os critérios descritos, devem ser implementados serviços em diferentes linguagens. Utilizou-se Java com apoio da *framework* Spring-Boot e Node.js com a *framework* Nestjs.

- Spring-Boot – Microserviços de configurações, propostas e provas.
- Nestjs – Microserviço de pessoas.

Software de *messaging*

É utilizado o serviço de mensagens RabbitMQ, o qual permite a comunicação inter-microserviços.

Domínio da aplicação

De forma a entender todas as secções seguintes deste documento e consolidar o problema, neste capítulo é abordado o domínio da aplicação desenvolvida. De um modo geral o seguinte diagrama representa o mesmo:

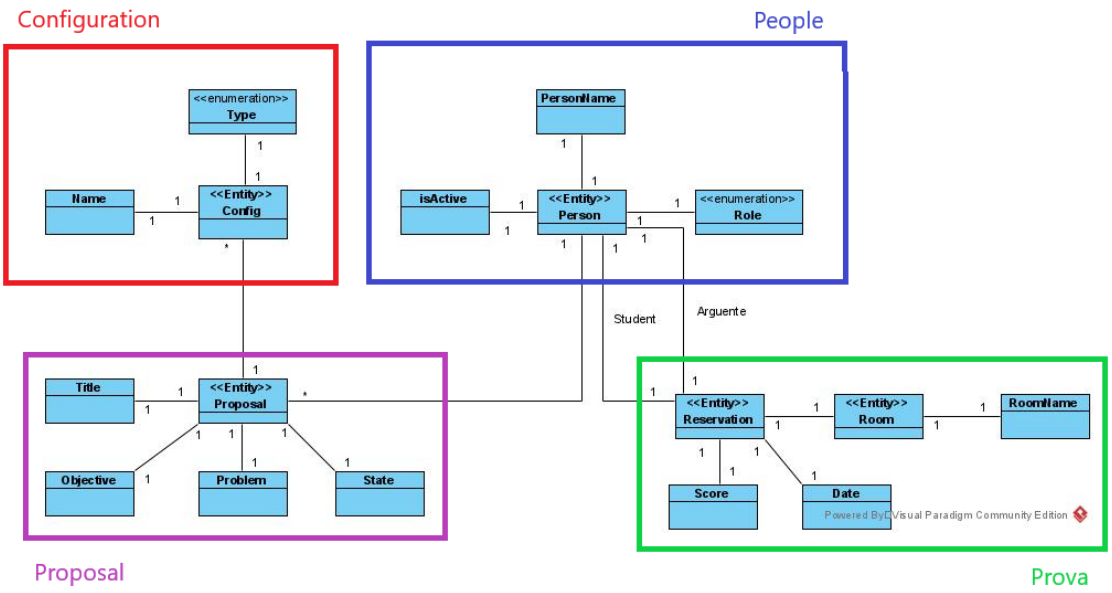

Figura 8 - Modelo de domínio

Dados os agregados apresentados, segue uma resumida explicação dos mesmos.

People:

- Responsável pela criação e gerenciamento de todos os utilizadores da aplicação.
- Os *Roles* são atribuídos aos utilizadores pelo RUC.
- Os novos estudantes encontram-se ativos no ano letivo corrente. No entanto, no fim do ano estes passam a inativos, sendo que apenas o RUC pode voltar a passá-los a ativos.

Proposal:

- Responsável pela gestão de todas as propostas da aplicação.

Prova:

- Responsável pela gestão de todas as provas e reservas de sala da aplicação.
- O objeto de domínio correspondente à reserva representa também a prova a que a mesma lhe está associada.
- As reservas de sala apenas podem ser realizadas até 24h antes do horário desejado e até uma semana de antecedência.
- Estas reservas apenas podem ser realizadas pelo RUC.

Configuration:

- Responsável pela gestão de todas as abordagens, tecnologias e palavras-chave da aplicação.

Estilos, padrões e boas-práticas

Na figura apresentada de seguida é possível perceber de melhor forma a arquitetura de todo o sistema. Dada a dimensão do diagrama, este foi também incluído nos anexos deste documento.

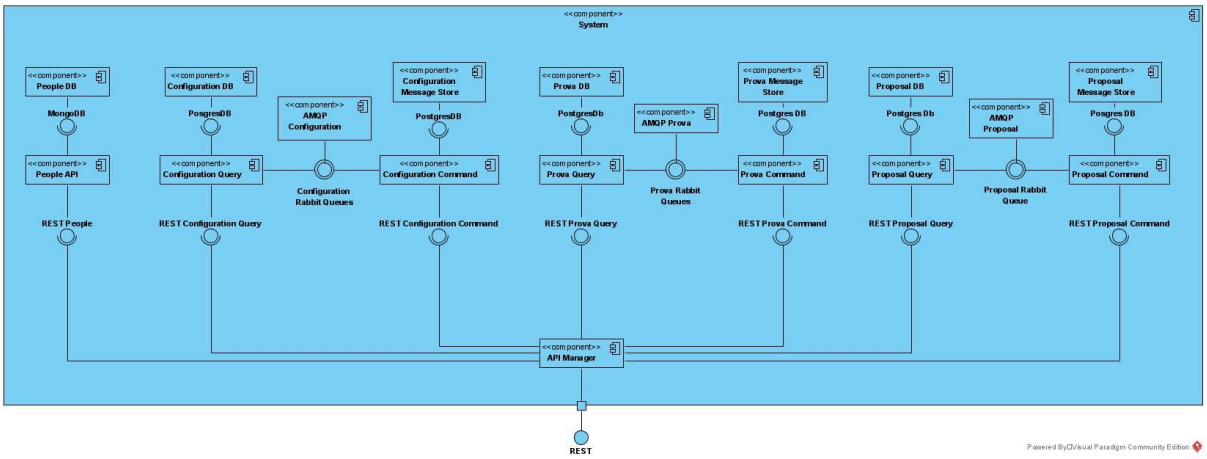

Figura 9 - Diagrama de componentes

A partir do diagrama de componentes acima apresentado, é possível realizar uma análise mais aprofundada do mesmo:

- Dada a reduzida complexidade do serviço de gestão de pessoas, foi implementado um microserviço independente que permite consultar, criar, alterar ou remover pessoas através de pedidos HTTP.
- ***Database per service***, como o próprio nome indica, faz uso de uma única base de dados por cada um dos microserviços. Para os serviços desenvolvidos em Spring-boot, fez-se uso de PostgresSQL e para os serviços implementados em NestJs utilizou-se MongoDB.
- Relativamente aos serviços de configurações, propostas e provas, foram implementados um conjunto de padrões, conforme eram requisitados, expostos nos pontos seguintes.
- **Padrão CQRS,** ao terem sido criados dois microserviços para cada um dos softwares de gestão, onde um (*Command*), é responsável por receber todos os pedidos relativos à criação, eliminação ou alteração dos dados, e um segundo (*Query*), responsável por receber todos os pedidos de leitura e retornar os dados pretendidos.
- ***Event Sourcing***, utilizado em todos os microserviços de *Command*, no qual são registados todos os eventos que passam pelo serviço e, desta forma, obter não só estado atual do sistema, assim como todos os passos que o levaram a esse ponto.
- ***Messaging* por MB,** assim como já foi referido e é evidenciado no diagrama apresentado, todos os microserviços (à exceção do serviço de pessoas), fazem uso de *queues* de mensagens, de forma a estabelecer a comunicação entre os diferentes microserviços.
- ***Saga/Eventual consistency***, de forma a assegurar a consistência dos dados entre os dois microserviços de cada software de gestão, foi criada uma outra queue responsável por apenas receber os eventuais erros que ocorram no serviço de *query*. Desta forma o serviço de comando, ao ler desta *queue,* regista os eventos relativos aos erros ocorridos de forma a manter o estado dos dados atualizado e consistente.

Ainda sobre o diagrama apresentado na figura 5, as ligações estabelecidas entre os 4 serviços foram desprezadas na representação de forma a tornar o diagrama menos complexo e de mais fácil leitura.

No entanto estas ligações são apresentadas na figura 6, onde a comunicação é realizada a partir dos pedidos HTTP que cada um dos serviços fornece.

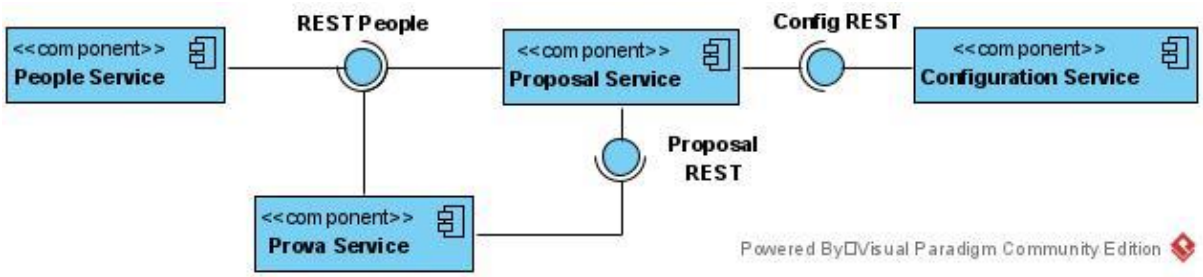

Figura 10 - Ligações entre serviços

Relativamente à elasticidade e escalabilidade do sistema fez-se uso de *containers* de Docker. Para além do serviço de *queues* do RabbitMQ estar a correr desta forma, para todas as bases de dados foram criados contentores específicos, assim como para as aplicações desenvolvidas em Spring-Boot e NestJs foram criadas as imagens e os respetivos contentores. Na figura seguinte são apresentados todos os contentores referenciados a correr.

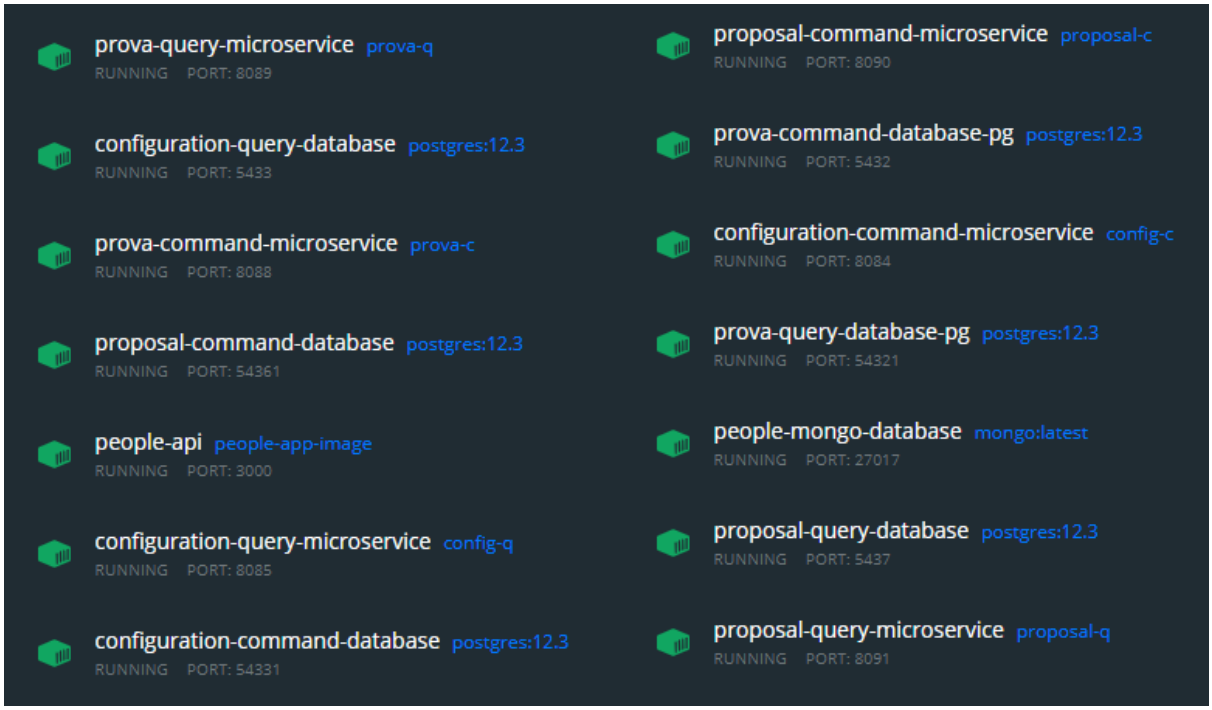

Figura 11 - Contentores em Docker

Testes automáticos e experiências à API e microserviços

Nesta secção são apresentados os testes automáticos realizados no Postman aos endpoints criados. Para isto são essencialmente testados os dados de resposta produzidos por cada um dos endpoints e se os respetivos códigos HTTP estão em conformidade com aquilo que é pretendido.

Na seguinte figura são apresentados os resultados obtidos (à parte deste documento segue o ficheiro JSON com todos os detalhes relativos aos testes apresentados):

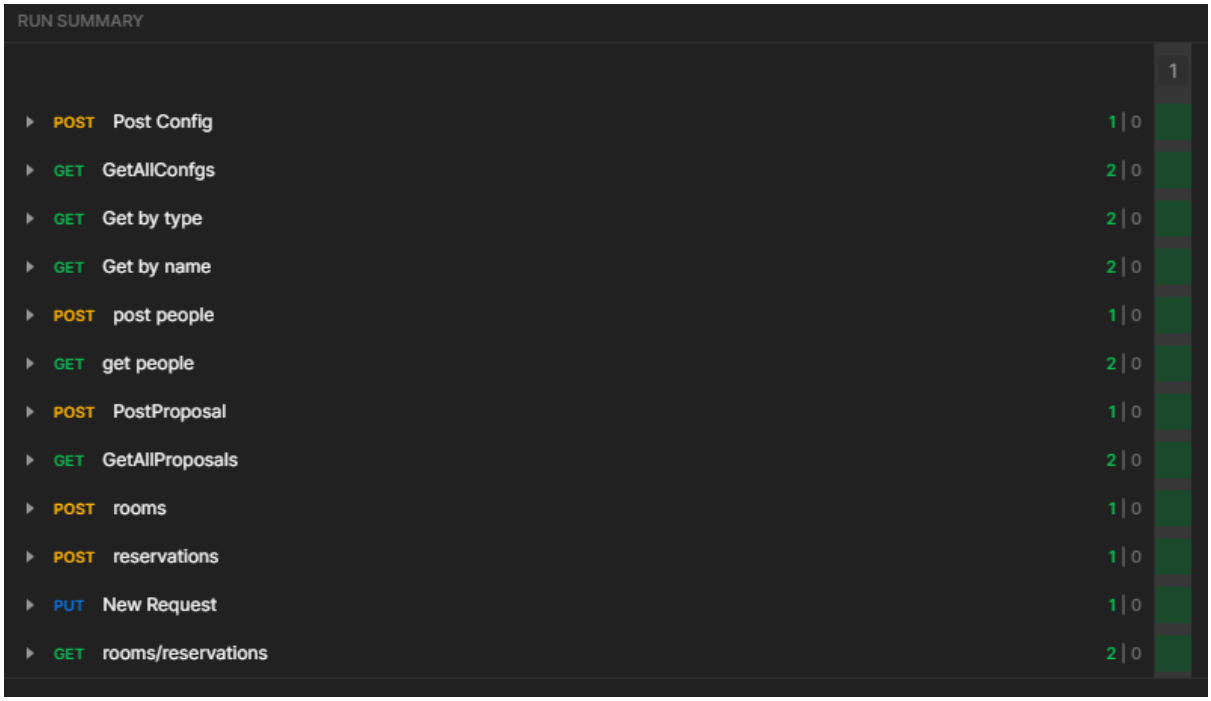

Figura 12 - Resultados dos testes

Levantamento de todos os serviços a partir de Docker

Assim como já foi referido, todos os microserviços criados, assim como as respetivas bases de dados, contêm o seu contentor respetivo de forma a poder agilizar todo o processo de demonstração e eventual *deploy* do sistema. Posto isto, os seguintes passos devem ser seguidos de forma a conseguir testar toda a aplicação.

1. Correr o RabbitMQ, da maneira desejada, na porta 5672.
2. Navegar até ao diretório "Project3".
3. Entrar no diretório "People-Microservice" e correr o comando "docker build -t people-app-image ."
4. Voltar ao diretório "Project3".
5. Para todos os restantes diretórios, os seguintes passos devem ser seguidos:
    - Correr o comando "mvn package".
    - Correr o comando "docker build -t ***nome-da-imagem*** .". O nome da imagem a inserir para cada um dos diretórios é listada mais abaixo.
6. Voltar ao diretório "Project3" e correr o comando "Docker-compose up".

Ao seguir estes 6 passos, todo o sistema fica disponível para utilização. É importante referir que neste processo não se encontra a instanciação da API Manager, pelo que os pedidos aos diferentes endpoints serão realizados diretamente aos serviços instanciados.

São de seguida apresentados os nomes para a variável ***nome-da-imagem*** utilizada no quinto passo:

- Diretório "Configuration_Microservice_Command" – **config-c**.
- Diretório "Configuration_Microservice_Query" – **config-q**.
- Diretório "Proposal_Microservice_Command" – **proposal-c**.
- Diretório "Proposal_Microservice_Query" – **proposal-q**.
- Diretório "Prova_Microservice_Command" – **prova-c**.

- Diretório “Prova_Microservice_Query” – **prova-q**.

API Manager

A utilização de API manager do WSO2 permitem implementar diversas funcionalidades e requisitos pretendidos da última iteração.

- Swagger é uma ferramenta de documentação usual em API’s Restful, com o objetivo de implementar a documentação utilizou-se as dependências disponíveis em Maven ou NPM, dependendo dos projetos. A documentação das Api’s encontra-se disponível nos projetos locais e no API Manager. A imagem abaixo representa a documentação obtida do serviço Prova_Query, contendo os endpoints e os modelos presentes no serviço.

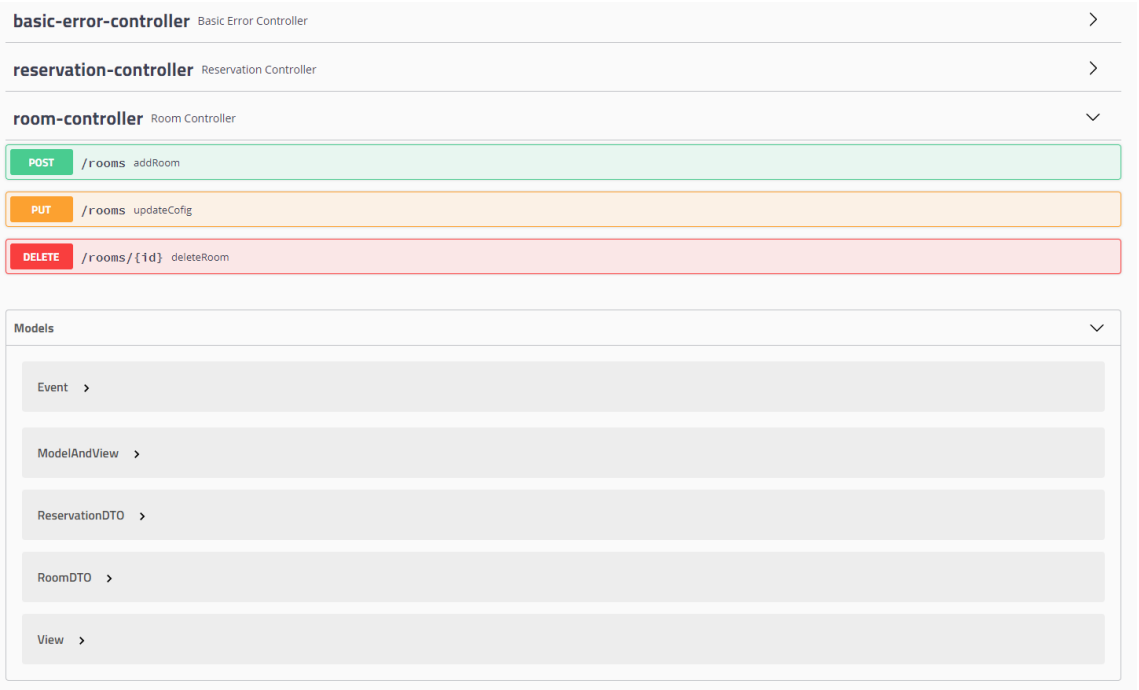

Figura 13 - Swagger Prova

- Utilização de scopes definidas no API, que permitem indicar quais os utilizadores ou os roles que podem realizar determinadas operações.
- Throttling definido novamente com apoio das funcionalidades disponibilizadas no próprio API Manager pode ser configurável de acordo com as necessidades do serviço.

Figura 14 - Throttling

- Analytics apesar de ter sido realizada uma pesquisa para alcançar o requisito, não foi possível cumprir com a necessidade pois este ponto fora o último a ser desenvolvido e dada a instalação realizada do API Manager em Docker a gestão da funcionalidade é algo complexo e poderia causa o trabalho desenvolvido para os pontos anteriores.

Melhorias/Alternativas

- Relativamente aos tipos de bases de dados selecionados, podia ter sido decido utilizar, por exemplo no caso dos serviços de *Query*, uma base de dados que permitisse uma leitura dos dados mais eficiente (Apache Solr, ElasticSearch, etc.). No entanto foi decidido fazer uso de tipos de bases de dados que os membros do grupo se sentissem mais confortáveis de utilizar de forma a agilizar todo o processo de desenvolvimento de todos os serviços.
- Dada a arquitetura apresentada na figura 5, são várias as alternativas ao sistema evidenciado. Uma alternativa, a qual iria contra grande parte dos requisitos enunciados para este trabalho, seria a omissão de todos os padrões e tecnologias orientados a eventos. Esta seria uma alternativa muito simplificada relativamente à que foi implementada.

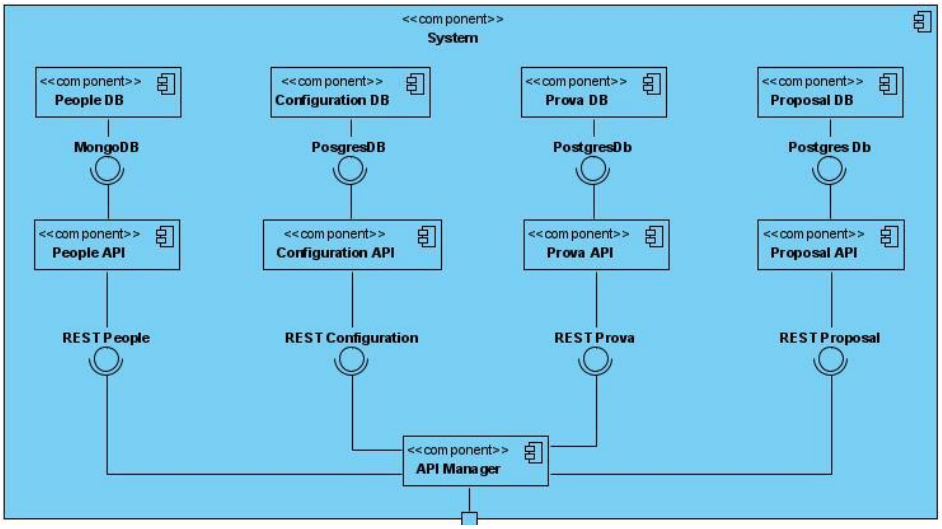

Figura 15 - Diagrama de componentes (alternativa)

- Relativamente às ligações entre serviços, apresentadas na figura 6, ao contrário do que é apresentado no ponto anterior no que toca a omissão de todos os padrões orientados a eventos, neste caso, visto que as ligações entre os serviços são estabelecidas via HTTP diretamente uns entre os outros, aqui seria proposta implementar uma comunicação a partir de *Message Queueing*.
- Ainda sobre a utilização de uma arquitetura orientada a eventos, poderia também ser considerada a utilização de uma tecnologia diferente ao RabbitMQ, como por exemplo, kafka. Dado o desconhecimento dos membros do grupo na implementação desta última, foi decidido utilizar RabbitMQ.

ANEXOS

A. Diagrama de componentes