**Reunião 27/05/21**

**Adição de um Widget a um Dashboard**

Foram feitas algumas alterações à forma como é criado um widget num dashboard.

O utilizador deve escolher de um conjunto de templates que são disponibilizados.

Estes templates apresentam informação temporária que irá ser atualizada pelo processo ETL, sendo que apenas os campos “function” e “name” são mantidos do template.

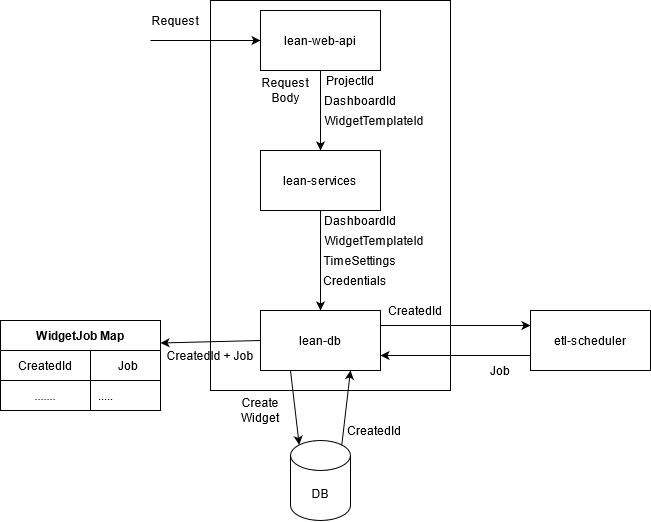
Os templates são armazenados no seu próprio índice no ElasticSearch.

O campo “function” é utilizado para associar o tipo do widget à sua função do ETL.

O utilizador deve também fornecer credenciais, definições de tempo, e, quando necessário, alguns parâmetros extra, como por exemplo um projeto específico, para que o ETL e o scheduler possam ser executados de forma correta.

Assim que um widget é criado, é também criado um job através do módulo etl-scheduler.

Este job é mantido na memória da aplicação, mais especificamente num mapa que associa o ID do widget ao seu job do scheduler para que mais tarde seja possível alterar a cadência do mesmo ou pará-lo completamente.



Surgiram algumas dúvidas quanto ao armazenamento dos jobs num mapa:

* Quando a aplicação é reiniciada a informação deste mapa é apagada. De que formas poderíamos armazenar esta informação? Guardar num ficheiro? Ou recriar o mapa cada vez que a aplicação inicia através dos widgets presentes em base de dados?

**Estrutura do Widget**

O widget contém informação sobre a cadência, função do ETL a ser executada e credenciais necessárias para o acesso às API necessárias.

"\_index": "etl-widgets",

                "\_type": "\_doc",

                "\_id": "lq0-pHkBawRrk9AESr16",

                "\_source": {

                    "name": "Squash test results pie chart",

                    "function": "postSquashTestsPieChart",

                    "source": "Squash",

                    "params": [],

                    "updateTime": {

                        "seconds": "",

                        "minutes": "\*/1",

                        "hours": "\*",

                        "dayOfMonth": "\*",

                        "month": "\*",

                        "dayOfWeek": "\*"

                    },

                    "credentials": {

                        "username": "guest\_tpl",

                        "password": "password",

                        "APIPath": "demo.squashtest.org"

                    },

                    "data": […]

O campo updateTime contém a informação necessária para configurar a cadência do Job criado. Este apresenta um campo para segundos, minutos, horas, dia do mês, mês e dia da semana.

Os campos function e params servem para a identificação da chamada do ETL a ser chamada. O campo function identifica unicamente a função a ser chamada, este é utilizado para obter a função a chamar, fazendo um “get” ao mapa WidgetMap que utiliza a string “function” como key e a função do ETL como value. O campo params contém informação que pode ser necessária para uma chamada correta à função do ETL, como por exemplo um projeto em específico.

O campo credentials contém informação para a chamada à API da fonte de informação pretendida.

Caso a fonte seja JIRA ou Azure contém informação relativa ao email e token, bem como o path do projeto em questão e a versão da API a utilizar.

Se for o Squash contém informação relativa ao username, password e path.

**Scheduler**

O scheduler recebe o Id de um widget e acede à base de dados para obter a informação desse widget que é necessária para criar um job.

O job limita-se a obter a função a chamar de acordo com o mapa WidgetMap e consoante o tamanho do array de params presente no Widget, executa a função com os parâmetros necessários. Caso o array params esteja vazio é apenas passado o Id do widget a atualizar e as credenciais para o ETL, sendo este processo repetido a cada intervalo de tempo predefinido.

Caso seja necessário alterar as informações de cadência deve ser chamada a função do scheduler *reSchedule* que recebe o novo intervalo de tempo e o job a alterar, sendo que esta informação deve mais tarde ficar armazenada no widget.

Para alterar as credenciais deve ser atualizada a informação do widget e depois criado um novo job, sendo que o scheduler teria de fazer uma chamada à DB para que a informação do widget presente em memória para o job seja também atualizada. Isto não é necessário para as definições temporais visto que existe uma função que permita alterar diretamente no job a cadência.

**Autenticação**

Começámos a implementação da autenticação e autorização utilizando o módulo Authization. Numa fase inicial, implementámos operações de login, logout e criação de utilizadores. Conseguindo criar utilizadores, decidimos criar o nosso modelo RBAC, definindo as permissões e papéis existentes na nossa aplicação:

Graphical user interface

Description automatically generated

**Superuser**

* Tem acesso incondicional a todos os recursos e operações
* Ao serem criadas as definições do modelo RBAC, é criado e recebe todas as permissões automaticamente pelo módulo Authization

**Manager:**

* Tem acesso a operações, mas as operações POST, PUT e DELETE são condicionais. Como estas são relativas a projetos, dashboards e widgets, um manager só as poderá realizar se for um projeto da qual seja dono.
* Tem de ser adicionado o role a um Collaborator

**Collaborator:**

* Papel default quando é criado um novo user
* Apenas tem permissão para operações GET, mas apenas se pertencer ao projeto em questão

Seguidamente, procedemos a implementar operações relativas à gestão de utilizadores e autorização.

**Funcionalidades implementadas:**

* Login
* Logout
* Criação de utilizadores
* Modelo RBAC
* Adicionar utilizadores a projetos
* Remover utilizadores de projetos

**Funcionalidades por implementar:**

* Dar roles a utilizadores
* Remover roles de um utilizador
* Editar utilizador
* Eliminar utilizador
* Fazer verificações nos acessos a recursos