P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

-

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Relatório do Trabalho de Processamento Estruturado de Informação

Licenciatura em Engenharia Informática 2022/2023

Grupo 21

(8210193) Diogo Cardoso
(8210390) Pedro Nunes
(82010399) Rui Ferreira

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Conteúdo

-

1 - Objetivo

Este trabalho tem como objetivo o desenvolvimento de um projeto para a Unidade Curricular de PEI, com o objetivo de criar uma REST API, conectada a uma base de dados com informações relativas ás marcações e peritagens de uma oficina, oficina esta de agora em diante designada por “nCar”.
Para isso, fizemos uso dos conhecimentos obtidos ao longo do semestre, tais como:

- Criação de vocabulário com o uso de schemas,
- Criação de documentos XML de teste para os schemas,
- Criação de API’s REST com o uso de XQuery.

Para além das hard-skills que se adquiriram durante a realização deste trabalho foram ainda adquiridas soft-skills, isto é, capacidades não técnicas que enquanto não essenciais para a realização da nossa profissão são boas adições ao nosso set de skills, exemplo de uma soft-skill é a capacidade de trabalho em equipa, uma soft-skill que é cada vez mais indispensável no mercado de trabalho em que o curso de Engenharia Informática se encontra inserido.

2 – Vocabulário

2.1. – Divisão de Ficheiros e Namespaces

Antes de proceder á "construção" do vocabulário foi necessário a discussão da divisão de ficheiros, facilitando assim a divisão do trabalho.

O vocabulário foi dividido nos seguintes ficheiros:

Appointment.xsd – Este documento apresenta o vocabulário relativo ás marcações e possui o seguinte Namespace: *"http://www.nCar.pt/appointment".*
Expertise.xsd – Este documento apresenta o vocabulário relativo ás peritagens e possui o seguinte Namespace: *"http://www.nCar.pt/expertise".*
Local.xsd – Este documento apresenta o vocabulário relativo ao local a visitar e será utilizado pelo vocabulário de marcações, tem o seguinte Namespace: *"http://www.nCar.pt/local".*
Vehicle.xsd – Este documento apresenta o vocabulário relativo ao veículo a inspecionar e será utilizado pelo vocabulário de marcações, tem o seguinte Namespace: *"http://www.nCar.pt/vehicle".*
Params.xsd – Este documento apresenta o vocabulário relativo aos parâmetros da peritagem e será utilizado pelo vocabulário das peritagens, tem o seguinte Namespace: *"http://www.nCar.pt/params".*
Partner.xsd – Este documento apresenta o vocabulário relativo a um parceiro e possui o seguinte Namespace: *"http://www.nCar.pt/partner".*
Expert.xsd – Este documento apresenta o vocabulário relativo a um perito e possui o seguinte Namespace: *"http://www.nCar.pt/expert".*

2.2. – Vocabulário Utilizado

Para explicarmos o vocabulário utilizado iremos dividir o mesmo entre **Marcações** e **Peritagens** uma vez que são os principais elementos deste projeto.

-

2.2.1 – Marcações

As marcações terão a seguinte estrutura:

Code – Aqui será armazenado o código de marcação, este elemento que é um número **inteiro** (xs:int) irá ser importante para validar as Peritagens na API.
idExpert – Este elemento que é do tipo **String**, uma vez que se trata de um código alfanumérico, irá armazenar o ID do perito que realizará esta marcação.
Local – Este elemento trata-se de um *complexType* e tem por sua vez outro *complexType* com o nome **Residence**, aqui ficam os elementos da cidade, do código postal, da rua, do numero da porta e, embora não obrigatório, das coordenadas geográficas, passando de novo para o Local, para além do Residence irá ainda armazenar um elemento **tipoCliente** do tipo *complexType* que nos indicará se se trata de uma Oficina ou um Cliente, um elemento **Name** que armazena o nome da Oficina/Cliente e ainda o elemento **Owner**, um elemento do tipo **Boolean** que apenas é requerido quando se trata de um Cliente e que específica se o Cliente é o dono do veículo.
Vehicle – Aqui estão as informações do veículo ao qual se irá proceder á peritagem, nele são utilizados os seguintes elementos: **Marca**, um elemento do tipo string e que armazena a marca do carro, **Modelo**, também do tipo string e armazena o modelo do veículo, **Combustível**, o usuário pode escolher dentro de uma lista de opções pré-definidas, **Transmissão**, aqui o usuário pode escolher entre Manual e Automática, **Ano,** um elemento do tipo inteiro que se refere ao ano do veículo e, por fim, a **Potência** em quilómetros (inteiro).
Date – elemento do tipo ***xs:date*** que guarda a data da marcação.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<appointment xmlns="http://www.nCar.pt/appointment"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.nCar.pt/appointment ../appointment.xsd">
        <Code>1344</Code>
        <idExpert>E23290F402</idExpert>
        <Local>
            <Residence xmlns="http://www.nCar.pt/local">
                <City>Lisboa</City>
                <postalCode>4534-404</postalCode>
                <Road>Rua de Santa Catarina</Road>
                <doorNumber>200</doorNumber>
            </Residence>
            <Manufactory xmlns="http://www.nCar.pt/local">true</Manufactory>
            <Name xmlns="http://www.nCar.pt/local">Oficina do Colas</Name>
        </Local>
        <Vehicle>
            <Marca xmlns="http://www.nCar.pt/vehicle">Toyota</Marca>
            <Modelo xmlns="http://www.nCar.pt/vehicle">Corolla</Modelo>
            <Combustivel xmlns="http://www.nCar.pt/vehicle">Diesel</Combustivel>
            <Transmissão xmlns="http://www.nCar.pt/vehicle">Auto</Transmissão>
            <Ano xmlns="http://www.nCar.pt/vehicle">2019</Ano>
            <Potência xmlns="http://www.nCar.pt/vehicle">1200</Potência>
        </Vehicle>
        <date>2022-06-19</date>
    </appointment>
```

Figura 1: Exemplo de um ficheiro válido de marcação

-

2.2.2 – Marcações

As peritagens terão a seguinte estrutura:

Code – Aqui será armazenado o código da Peritagem, este elemento que é um número **inteiro** (xs:int).
idExpert – Este elemento que é do tipo **String**, uma vez que se trata de um código alfanumérico, irá armazenar o ID do perito que realizou esta peritagem.
idPartner – Este elemento que é do tipo **String**, uma vez que se trata de um código alfanumérico, irá armazenar o ID do parceiro que realizou esta peritagem.
State – Este elemento, que poderá tomar o valor de "YES" ou "NO" e tem como objetivo informar se a peritagem foi realizada ou não.
expertiseInformation – Este elemento, do tipo *complexType*, irá armazenar os elementos de **startTime**, **endTime, Entity** (Oficina ou Cliente) e **Parameters**, este elemento do tipo *complexType* armazena os resultados da peritagem, o usuário pode escolher se o componente se encontra OK e caso não esteja pode escolher uma das opções pré-definidas, podem ser selecionadas mais que uma opção quando pedido pelo formulário.
Category – Caso a peritagem não tenha sido realizada terá que ser selecionada uma das opções pré-definidas.
Reason – Caso não se encontre a razão para a não realização da peritagem no elemento Category o usuário poderá usar 400 caracteres para explicar o motivo da não realização da peritagem.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Expertises xmlns="http://www.nCar.pt/expertise"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.nCar.pt/expertise ../expertise.xsd">
    <idPartner>123</idPartner>
    <idExpert>123</idExpert>
    <date>2019-06-20</date>
    <expertise>
        <Code>1</Code>
        <idPartner>123</idPartner>
        <idExpert>123</idExpert>
        <State>NO</State>
        <Category>Missing Vehicle</Category>
        <date>2019-06-12</date>
    </expertise>
</Expertises>
```

Figura 2:Exemplo de um ficheiro XML válido para uma peritagem

-

3 – API

Neste capítulo iremos dissecar a nossa API, iremos explicar as funções utilizadas e os seus casos de utilização.

3.1. – Funções Utilizadas e Explicação

3.1.1 – validateXML($appointment), validateExpertiseXML($expertise), validatePartnerXML($partner), validateExpertXML($expert)

Apesar de nomes diferentes, todas estas funções fazem, essencialmente, o mesmo, recebem como argumento um ficheiro xml, têm dentro de uma variável o *path* para o schema pelo qual aquele xml tem que passar e retorna se o xml é valido ou não.

3.1.2 – addEx($expertise as item())

Esta função, do tipo **POST**, irá receber um ficheiro xml com um conjunto de peritagens realizadas num determinado dia, esta função irá fazer a validação do ficheiro (através de uma das funções referidas anteriormente), após isso serão criadas algumas variáveis que armazenarão:

- Se todas as peritagens possuem um Parceiro válido;
- Se todas as peritagens possuem um Perito válido;
- Se todas as peritagens têm um Código igual ao de uma Marcação;
- Se todas as peritagens têm a mesma data;
- Número de Peritagens;
- Se todas as peritagens têm códigos diferentes entre si;
- Se existe alguma Peritagem duplicada.

Caso o segundo ponto seja igual a todos os acima e os dois restantes forem iguais a 0, esses conjuntos de peritagens serão adicionados á base de dados através da função **addExpertise($expertise),** caso contrário, o utilizador será informado de um conflito.

3.1.3 – findExpertiseByDate($PartnerCode , $expertiseDate, $state)

Esta função, do tipo **GET,** irá retornar ao usuário as peritagens realizadas ou não realizadas pelo Parceiro indicado na data pedida.

Este pedido pode ser enviado através do seguinte endereço:
http://localhost:8080/expertises/find/partner/{$partnerCode}/{$state}/?expertiseDate={$expertiseDate}

Como é possível verificar, enquanto que o ID do parceiro e o Estado da Peritagens são dados no *path*, a data da peritagem é dada como *query*.

A função irá verificar se todos os parâmetros são validos, isto é, se o código do parceiro existe, se a variável $state está definida como "YES" ou "NO" e se existe uma peritagem realizada pelo parceiro na data inserida, caso não exista nenhuma peritagem nos parâmetros pedidos ou se algum parâmetro utilizar valores não pretendidos é informado ao utilizador a existência de um conflito, caso contrário, irá ser fornecida ao utilizador a informação pedida.

3.1.4 – showExpertise($expertiseCode)

Esta função do tipo **GET**, irá retornar ao usuário uma peritagem com o código enviado no *path.*

Este pedido pode ser enviado pelo seguinte endereço:

http://localhost:8080/expertises/{$expertiseCode}

O que esta função vai fazer é criar uma variável que armazena uma peritagem com esse código, caso ela exista essa será retornada ao utilizador, caso não exista é apresentada ao utilizador a mensagem de conflicto.

-

3.1.5 – replaceEx($expertise as item())

Esta função, do tipo **PUT**, irá substituir uma peritagem por outra enviada pelo usuário.

Será verificado se as seguintes condições se confirmam:

- Existe uma Marcação com o mesmo código;
- Existe um Parceiro com o mesmo código do exibido na Peritagem;
- Existe um Perito com o mesmo código do exibido na Peritagem;
- Se ainda não existe uma peritagem com o mesmo código.

Caso todas estas condições se confirmem esta peritagem será substituída, caso contrário o utilizador será avisado sobre um conflito.

Este pedido pode ser enviado pelo seguinte endereço:

http://localhost:8080/expertises/replace

3.1.6 – findAppointmentByExpert($expertCode)

Esta função, do tipo **GET**, irá apresentar ao usuário as marcações de um perito para o dia atual.

Em primeiro lugar é criada uma variável que armazena o dia atual, de seguida é guardada numa outra variável as peritagens que confiram ambos os parâmetros (lembrando que o código do perito é passado através do *path*).

Caso esta varável não seja nula esta será retornada ao usuário, caso contrário será apresentada ao utilizador uma mensagem de conflito.

Este pedido pode ser enviado pelo seguinte endereço:

http://localhost:8080/appointment/find/expert/{$expertCode}

4 – Apreciação Critica

4.1. – Principais Dificuldades

As principais dificuldades foram encontradas durante a criação da API, a introdução do XQuery com a do BaseX, uma ferramenta com documentação reduzida, parecia assustadora a princípio e demorou até conseguirmos "dominar" as ferramentas que tínhamos a nosso dispor mas através da prática, pesquisa e um pouco de tentativa e erro fizeram com que ganhássemos algum conhecimento (mesmo de domínios que não foram tocados neste trabalho) que nos serão uteis no futuro, relativamente ao vocabulário era algo em que já tínhamos alguma experiencia obtida com os exercícios dados nas aulas teóricas e teórico-práticas sendo apenas necessários alguns ajustes á sua estrutura durante a criação da API.

4.2. – Aspetos a melhorar

Alguns aspetos a melhorar no nosso trabalho seriam provavelmente mudarmos o nome de algumas funções e torna-las mais "entendíveis" para quem vê o trabalho de fora, apesar disso, o maior aspeto a melhorar será talvez a forma como armazenamos as marcações, ao contrário das Peritagens estas são armazenadas de forma separada, esta opção teve como objetivo demonstrar duas formas de armazenar a base de dados mas é claro que a forma de apresentar a informação mais organizada foi a utilizada nas peritagens.