Icon

Description automatically generated with medium confidence

Unidade Curricular

Processamento Estruturado de Informação

Trabalho Prático

Docente: Bruno Moisés Oliveira

Alunos:

Rui Alexandre Borba Vitorino (8190479)

Bruno Miguel do Carmo Vieira (8190724)

Gilberto Jorge Da Mota Gomes (8210227)

Licenciatura em Segurança Informática de Redes de Computadores

Índice

[Estrutura XML & XSD 1](#_Toc93653582)

[BaseX API 3](#_Toc93653583)

[Introdução 3](#_Toc93653584)

[Funcionalidades 3](#_Toc93653585)

[Workflow 4](#_Toc93653586)

[Em detalhe 5](#_Toc93653587)

[Cancelamento 10](#_Toc93653588)

[Particularidades 10](#_Toc93653589)

[Verificação de Disponibilidade 11](#_Toc93653590)

[Python Scripts 12](#_Toc93653591)

[generateData.py 12](#_Toc93653592)

[convert.py 13](#_Toc93653593)

[MongoDB 14](#_Toc93653594)

[Modelação dos Dados 14](#_Toc93653595)

[Queries 14](#_Toc93653596)

[Charts 14](#_Toc93653597)

Índice de Figuras

Índice de Tabelas

# Estrutura XML & XSD

A estrutura do trabalho prática assenta-se sobre 4 ficheiros XML estando a sua estrutura e composição definida em ficheiros XSD, que irão ser brevemente explicados.

* Atelier
* Reservation
* Family
* GeographicData
* CommonTypes

Começando a abordagem pelo último ficheiro, visto este ser utilizado por todo os outros; o ficheiro XSD CommonTypes.xsd contêm apenas dois tipos simples de elementos. Tipos estes que definem um padrão (pattern) para determinado tipo de dados, sendo essas padrões expressões regulares:

* Data (pe: 2022-01-15 e não 22/1/15)
* Nome (pe: “John Doe” e não “123John” ou “joão”)

Sendo o seguinte ficheiro utilizado pela Family.xsd e Reservation.xsd; o GeographicData.xsd contém alguns países e relativas informações para que se limite os dados introduzidos no momento da criação de uma reserva. Importante mencionar que não existe um elevado número de países devido a este ser apenas um exemplo, mas a seguinte referência contém o ficheiro base utilizados, este que sim, apresenta todos os dados. (*Core XML Schema - CountryType (Simple Type) | Ed-Fi Tech Docs*, n.d.)

Face às regras impostas, e às especificidades atribuídas aos elementos que compõem uma família, optou-se por criar um XSD para validação dos dados introduzidos, denominado por “Family.xsd ”, podendo depois este ser referenciado aquando necessário.

Como mencionado, existe limitação de:

* Número máximo de famílias por reserva (uma família)
* Número máximo de elementos da família (sete elementos)
* Número de dias preferidos de reserva (cinco dias)

Define a estrutura dos elementos:

* Nome
* Data de nascimento
* País
* Cidade (ficheiro GeographicData)
* Contacto de emergência

Todas as informações referentes a cada reserva são colocadas numa estrutura XML, para ser enviada como “corpo” de um pedido POST para a API, as mesmas são validadas pelo Reservation.xsd.

Informações contidas em cada reserva:

* Todos os dados da família
* Datas de preferência

Este ficheiro é de grande importância pois a correta validação do mesmo vai garantir a conformidade dos dados que posteriormente serão adicionados à Base de Dados.

Por último, o ficheiro Atelier.xsd é aquele que contêm as datas disponíveis para visita e que faz a gestão do limite diário de 50 famílias.

Informações contidas em data:

* Data (validada dentro dos 100 dias)
* Número de visitas disponíveis (slots)

As restrições implementadas neste ficheiro são o formato da data, que é do tipo xs:date, com um padrão definido por uma expressão regular, de forma a garantir a conformidade das datas, pois à posterior, estes dados serão usados numa função de validação de intervalo temporal (100 dias antes do Natal, “2022-09-17 a 2022-12-25”).

Os valores permitidos nas *slots* (0 a 50, ambos inclusivos), é uma validação redundante, uma vez que sempre que é criada ou cancelada uma reserva, o valor da *slot* é alterado por um processo de validação automático.

Inicialmente, este ficheiro contém apenas a data inicial do funcionamento do Atelier do Pai Natal, dia 2022-09-17 com 50 *slots* disponíveis, no entanto, sempre que se realizam novas reservas, as datas serão adicionadas num processo automático e decisão, que será analisado posteriormente na explicação da API.

## Namespaces

asdasad

# BaseX API

## Introdução

Servindo de base para toda a continuação do trabalho, a API escrita em RestXQ e suportada pelo *software* BaseX foi criada tendo as seguintes funcionalidades:

### Funcionalidades

* Submissão de uma reserva
  + Após a verificação de disponibilidade em relação aos dias escolhidos.
* Verificação de disponibilidade para um ou todos os dias
* Cancelamento de uma reserva (usando o seu ID).
* Exportação da Base de Dados

## Workflow

Para que todos os requisitos do enunciado fossem cumpridos, a API tornou-se complexa para quem a observa pela primeira vez, logo uma representação visual do *workflow* de execução e estruturação é essencial.

Diagram

Description automatically generated

Figura 1 Workflow de execução da API

## Em detalhe

Com a ajuda do diagrama torna-se mais simples a explicação do funcionamento da API, que será apresentado de forma detalhada nos pontos seguintes:

1. post-xml

Parâmetro: $xml – Contém o “Body”do POST Request que corresponde ao XML da reserva.

1ª Operação: Validação do XML com o Reservation.xsd utilizando a função “validate:xsd()”, se os dados não cumprirem com a estrutura definida, dá um erro, se estiverem corretamente formatados, então continua para a segunda operação.

2ª Operação: É chamada a função checks(), com a variável $xml passada como argumento com o XML já validado.

1. checks

É a função mais importante da API, pois será a responsável por:

* filtrar e validar as datas
* Adicionar reservas à base de dados
* Fazer updates à base de dados

Variáveis declaradas na função check()

|  |  |
| --- | --- |
| **NOME VARIÁVEL** | **CONTEÚDO/OBJETIVO** |
| $database | Aceder à informação contida na Base de Dados |
| $pfDates | Datas preferidas da família |
| $atelierDates | Datas presentes no ficheiro Atelier.xml |
| $existingDates | Datas escolhidas pela família e que já existem no ficheiro Atelier.xml |
| $validSlots | Datas contidas no $existingDates que têm “slots” com o valor maior do que zero. (filtra os dias escolhidos pela família que têm disponibilidade) |
| $validDate | Primeira data da $validSlots, caso não esteja NULL e passe em todas as validações seguintes, será utilizada para criar uma reserva. |
| $notExistingDates | Datas escolhidas que não constam no Atelier.xml.  Caso esta seja uma data válida, entre 2022-09-15 e 2022-12-23 será usada para a criação automática da data. |
| $checkBetweenDates | Recebe o resultado da validação da $notExistingDates |
| $newDate | Primeira data da $checkBetweenDates caso não esteja NULL, será utilizada para adicionar uma nova data ao ficheiro Atelier.xml e criar uma reserva. |
| $rid | Contém o valor do total de reservas mais um. |

1. Explicação do processo de filtragem da informação

Uma vez que foi tomada a decisão de as datas para os dias das visitas serem criadas num processo automático, será necessário efetuar uma série de verificações, que garantam o sucesso desta opção.

Uma vez que a família pode escolher datas que ainda não existam no ficheiro Atelier.xml, é necessário validar se essas se encontram nos 100 dias definidos para as visitas.

A Base de Dados é inicializada apenas com o primeiro dia do intervalo de tempo definido, ou seja, o ficheiro Atelier.xml contém apenas uma data disponível.

Assim, existem várias possibilidades:

* A família escolher dia(s) ainda não presentes no Atelier.xml, mas válidos no intervalo
* A família escolher dia(s) ainda não presentes no Atelier.xml **e** válidos no intervalo
* A família escolher dias(s) presentes no Atelier.xml com disponibilidade
* A família escolher dias(s) presentes no Atelier.xml sem disponibilidade

Inicia-se então um processo de filtragem que permita validar ou não as datas preferidas da família.

1ª Verificação: Encontrar todas as datas que a família escolheu **e** que estão presentes no ficheiro Atelier.xml, caso não existam datas em comum, a variável $existingDates, estará vazia.

2ª Verificação: Verifica se as datas da família ainda não existem no Atelier ($notExistingDates).

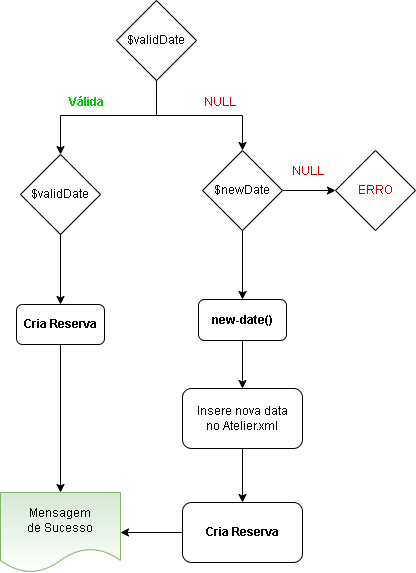
3ª Verificação: Validar se as datas da $notExistingDates estão dentro do intervalo.

1. Processo de tomada de decisão

Se a variável $validDate estiver vazia e a $newDate também é porque a Oficina do Pai Natal atingiu o máximo de capacidade para o(s) dia(s) escolhidos ou as datas escolhidas não estão entre 2022-09-15 e 2022-12-23

Se a variável $validDate estiver vazia e a $newDate não, então é porque a data não existe no Atelier.xml, logo a mesma será adicionada e será criada uma reserva.

Se a variável $validDate não estiver vazia, é porque existia no Atelier.xml uma data coincidente com uma das datas preferidas da família e com slots disponíveis, então é criada uma reserva.



1. check-between-dates(arg1)

Faz-se uma iteração por cada data recebida no argumento e verifica se esta se encontra entre 2022-09-15 e 2022-12-23, retornando as válidas ou retornando a variável vazia se estiverem fora do intervalo.

1. new-date(arg1)

Abre o Atelier.xml e insere um novo elemento "reservations" com a data que é passada no argumento já com as slots a 49, uma vez que vai ser criada a primeira reserva para este dia.

1. return-xml-reservation(arg1,arg2)

Com a informação passada no primeiro argumento elabora e retorna um XML com a estrutura de uma reserva para a data que está no segundo argumento.

Para atribuir o id a esta nova reserva conta as já existentes e incrementa um valor.

1. check-availability(Arg1)

Usada para fazer a consulta de disponibilidade de dias, recebe como argumento os parâmetros passados pelo GET Request.

Se o argumento for "all", verifica todos os dias existentes no Atelier e retorna o número de slots disponíveis por dia.

Se a consulta for para dias específicos, são efetuadas as verificações necessárias, e dependendo da situação é retornada a informação correspondente a cada data.

A screenshot of a computer

Description automatically generated with medium confidence

1. print-all-days()

Como se tomou a decisão, de criar as datas no Atelier.xml de forma automática, quando se faz uma verificação de disponibilidade para os 100 dias, teve de se considerar a possibilidade de ainda não haver reservas para todas as datas possíveis.

Esta função serve para garantir que a informação prestada, incluí as datas já existentes e todas as outras no intervalo de 2022-09-15 a 2022-12-23.

O seu funcionamento é simples, pois contam-se todos os elementos existentes no Atelier.xml, e se o valor for inferior a 100, quer dizer que ainda existem dias que podem ser criados com 50 slots, como tal essa informação é passada para o utilizador. Caso o contrário, não faz nada, pois todos os dias já foram listados anteriormente.

1. cancel-reservation($id as xs:string)

Usada para fazer o cancelamento de uma reserva, recebe como argumento o parâmetro passado pelo GET Request, neste caso é o id da reserva a cancelar.

Abre a reserva, e verifica se esta já se encontra cancelada, se sim retorna um erro ao utilizador a informar desta situação.

Senão, vai tentar alterar o estado para cancelado, no entanto, se não conseguir, é porque a reserva não existe, logo retorna essa informação ao utilizador.

Caso o id corresponda a uma reserva que possa ser cancelada, então altera o estado para cancelado, e vai ao Atelier.txt e aumenta o valor das “slots” na data correspondente com a da reserva e informando o utilizador do sucesso da operação.

1. exportdatabase()

Exporta todos os ficheiros em formato XML.

1. reservationok($date, $id)

Função usada para enviar mensagem de sucesso na criação de uma reserva.

1. canceledok($id)

Função usada para enviar mensagem de sucesso no cancelamento de uma reserva.

1. page:exportdone()

Função usada para enviar mensagem de sucesso na exportação da Base de Dados

1. deleteAndCreateDB()

Função que serve para apagar a Base de Dados e criar uma nova, usada pelo script “generateData.py que é usado para introduzir dados aleatórios na mesma.

# Python Scripts

Foram desenvolvidos dois *scripts* na linguagem de programação Python devido a dois elementos do grupo terem conhecimento sobre a linguagem e as capacidades da mesma pudessem ser enquadradas e utilizadas durante a realização do trabalho, nomeadamente nas tarefas:

* Geração aleatória de dados
* Transformação de ficheiros XML para JSON
* Importação dos ficheiros JSON para MongoDB
* Embutimento de informações em ficheiros JSON

## generateData.py

Este *script* tem como funcionalidade gerar dados aleatórios para que se possa tornar as reservas a analisar relativamente mais próximas a uma realidade.

Dados estes que são compreendidos entre uma quantidade aleatória de elementos da família (entre 1 a 7), sendo os seus nomes e datas de nascimento também aleatórias, e um número de datas preferidas de visita ao Atelier (entre 1 a 5), compreendidas dentro do intervalo de tempo de funcionamento do Atelier.

O *script* é executado através da linha de comandos, tendo três parâmetros:

1. Número de reservas a gerar
2. Número de reservas a cancelar
3. Opção de execução do segundo *script* que transforma e realiza o envio dos dados para o MongoDB

*Nota: o segundo e terceiro parâmetro são opcionais*

## convert.py

Este *script* tem três propósitos, estando estes interligados.

O primeiro é a remoção de *namespaces* e prefixos dos ficheiros XMLs exportados pelo BaseX.

Embora estes não sejam utilizados nem reconhecidos pelo MongoDB e os seus formatos (JSON/BSON), decidiu-se os remover para que existe uma conversão limpa e simples entre formatos, utilizando expressões regulares para tal.

O segundo é a conversão dos agora “limpos” e puros ficheiros XMLs para formato JSON, para que estes possam ser enviados corretamente para o MongoDB, conversão esta que é realizada utilizado o módulo xmltodict. (*Xmltodict · PyPI*, n.d.)

Após a conversão, o *script* irá ler os ficheiros presente na diretoria de nome, JSON, e enviá-los para uma definida *collection* existente num Cluster Atlas, utilizando o módulo pymongo. (*PyMongo 4.0.1 Documentation*, n.d.)

São ainda efetuados dois embutimentos em cada ficheiro JSON a ser enviado, nomeadamente as contagens do número de reservas ativas e canceladas, e o número de pessoas relativamente a cada estado de reserva. Isto, pois, embutir a informação é bastante mais eficaz a nível de *performance*, em comparação ao embutimento da mesma informação no MongoDB usando*queries*.

De qualquer forma, as referentes *queries* em sintaxe MongoDB encontram-se presentes na diretoria MONGOQUERIES/extra.

# MongoDB

## Modelação dos Dados

Inicialmente a base de dados, assenta em duas estruturas XML, e ao ser importada para o MongoDB, optou-se por agrupar ambas as estruturas.

|  |  |
| --- | --- |
| **Campo** | **Tipo de dado** |
| Data | Date |
| Slots | Integers |
| Reservas Ativas |
| Reservas Canceladas |
| Pessoas nas Reservas Ativas |
| Pessoas nas Reservas Canceladas |
| Reservas, contendo o ID, estado, e os dados relativos à família | Array |

Tabela Modelação dos dados

## Queries

## Charts