ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Unidade Curricular de Processamento Estruturado de Informação

Ano Letivo de 2022/2023

StepUp Shoes

Luís Oliveira 8190370

05 de Setembro, 2023

Data de Receção	
Responsável	
Avaliação	
Observações	

Luís Oliveira 8190370

05 de Setembro, 2023

Resumo

Este relatório descreve as várias fases da criação e implementação de vocabulário XML e de uma API através do BaseX.

Este projeto é relativo à componente prática da Unidade Curricular de Processamento Estruturado de Informação.

Índice

1. Introdução

1.1 Contextualização

O projeto é relativo à componente prática da Unidade Curricular de Processamento Estruturado de Informação e tem como objetivo principal o desenvolvimento de vocabulário XML e de uma API através do BaseX.

1.2 Apresentação do Caso de Estudo

A StepUp Shoes é uma renomada empresa de calçado que oferece uma variedade diversificada de produtos. Durante sua trajetória, estabeleceu várias parcerias ao nível nacional, variando de vendedores individuais a grandes lojas de varejo que comercializam seus produtos.

Ao longo dos anos, a empresa presenciou um crescimento significativo e, com esse aumento, surgiu a necessidade de um sistema mais eficiente de rastreamento e relatórios de vendas. Atualmente, a StepUp Shoes exige que cada um dos seus vendedores e parceiros apresente mensalmente um relatório detalhado de vendas.

O objetivo principal deste estudo é examinar e propor uma solução eficaz para consolidar esses relatórios. Dada a diversidade de parceiros e sistemas que eles possam usar, a empresa optou por um vocabulário XML uniformizados.

1.3 Motivação e Objetivos

A crescente complexidade e diversidade das operações comerciais da StepUp Shoes trouxeram consigo desafios na forma de gestão de dados. Com vários parceiros usando sistemas diferentes e, possivelmente, formatos de relatório variados, tornou-se um desafio integrar todos esses dados de uma maneira uniforme e eficiente. Além disso, a análise manual de relatórios de vendas de diferentes parceiros pode levar a discrepâncias e ineficiências, possivelmente afetando a tomada de decisão estratégica na empresa.

Objetivos:

- Uniformização de Relatórios: Implementar um sistema uniformizado onde todos os parceiros submetam seus relatórios de vendas num formato XML uniformizado.

- Automatização e Eficiência: Criar uma solução que permita a importação automática de dados de vendas dos parceiros em um banco de dados centralizado (MongoDB) e, posteriormente, permitir a criação de relatórios consolidados.

- Flexibilidade e Acessibilidade: Usar o Mongo Atlas e a Data API para garantir que os dados possam ser acedidos e manipulados através de chamadas HTTP, proporcionando uma camada adicional de flexibilidade e interoperabilidade.

- Transformação de Dados: Através do BaseX, implementar uma API que possa transformar e fornecer os dados necessários em formato XML conforme o vocabulário fornecido pela StepUp Shoes.

1.4 Estrutura do Relatório

De forma a facilitar a sua consulta, a estrutura do relatório é dividida em capítulos e subcapítulos, onde se descreve os passos para a realização deste projeto.

É apresentado o tema, a contextualização e o caso de estudo deste projeto na introdução. De seguida, descreve-se o processo de desenvolvimento deste projeto e por fim, uma reflexão sobre o projeto.

2 Desenvolvimento do Vocabulário XML

Esta secção aborda a estrutura e o desenvolvimento de um vocabulário XML destinado a representar relatórios de vendas para a StepUp Shoes. Além do documento XML, um esquema XSD foi criado para validar a estrutura e o conteúdo do XML. Devido ao fim da licença do Oxygen, foi utilizado o VS Code para o desenvolvimento dos XMLs e XSD. Foi utilizado o XMLSpy Altova para a criação da documentação disponível na pasta “documentação XSD”.

2.1 Estrutura do Documento XML

O documento XML fornece uma representação estruturada de um relatório de vendas, *SalesReport*, de um determinado mês do vendedor da StepUo Shoes. Ele contém informações detalhadas sobre a StepUo Shoes, os detalhes do relatório de vendas (como ano fiscal e mês) e uma lista detalhada de vendas individuais. Cada venda individual é uma fatura, que contém informação sobre o cliente e os produtos.

2.1.1 Detalhes do Relatório de Vendas (*SalesReport*)

Aqui foram definidos os parâmetros do relatório de venda:

- *FiscalYear*: O ano fiscal do relatório.
- *Month*: O mês do relatório.
- *DistinctProductCount*: Número de produtos distintos vendidos.
- *TotalSalesValue*: Valor total das vendas.
- *DistinctClientCount*: Número de clientes distintos.

2.1.2 Detalhes do Vendedor (*Seller*)

Este elemento encapsula informações relacionadas com o vendedor, como:

- *VATNumber*: Número de Identificação Fiscal.
- *Name*: Nome da empresa.
- *Address, City, Country, PostalCode*: Detalhes do endereço do vendedor.

2.1.3 Detalhes de Venda Individual (*SaleDetail*)

O elemento "SaleDetail" detalha cada venda individual, ou seja, uma fatura, incluindo:

- InvoiceID, *Date* : Detalhes da fatura.
- Client : Detalhes do cliente.
- *LineItem*: linhas da venda, onde contém os detalhes do produto vendido (*Product*)

2.2 Esquema XSD

O esquema XSD, *SalesReport.xsd*, permite validar os documentos XML para garantir que sigam a estrutura correta e contenham valores válidos.

2.2.1 Tipos Definidos: *vatNumberType* e *clientVATType*

Estes tipos personalizados foram desenvolvidos para gerir os números de contribuinte. O *vatNumberType* define um padrão de 9 dígitos. Como o número de contribuinte de um cliente também pode assumir o valor “"Consumidor Final, foi criado o *clientVATType.*

2.2.2 Validações e Restrições

O esquema XSD implementa várias validações, incluindo:

- Padrões para números de contribuinte e códigos postais.
- Intervalos de valores para meses (1-12).
- Padrões de enumeração para determinados campos, como "Consumidor Final" para o número de IVA do cliente.

3 MongoDB

Esta secção aborda a importação dos dados fornecidos, formatação dos mesmos e ativação da Data API. Foi também ativada a Data API no Mongo Atlas para permitir o acesso aos dados armazenados na base de dados do Atlas utilizando HTTP.

3.1 Importação de dados

A importação dos dados CSV foi feita através da interface do MongoDB Compass. Foi criada a base de dados *SalesData* com quatro coleções: Customers, Products, SalesHeader e "SalesLines".
Os dados apresentavam algumas incongruências e para as tentar resolver foi utilizado um script (que se encontra no ficheiro *FormatarDados_Script.txt*). Foram formatadas três das quatro coleções criadas:

```
db.SalesLines.find({}).forEach(function(doc) {
  var valorEUR = doc.valorEUR && typeof doc.valorEUR === 'string' ? parseFloat(doc.valorEUR.replace(/,/g, '.')) : doc.valorEUR;
  var descontoEUR = doc.descontoEUR && typeof doc.descontoEUR === 'string' ? parseFloat(doc.descontoEUR.replace(/,/g, '.')) : doc.descontoEUR;
  db.SalesLines.updateOne({_id: doc._id}, {$set: {valorEUR: valorEUR, descontoEUR: descontoEUR}});
});
```

- Para cada documento da coleção *SalesLines*, verifica se o campo *valorEUR* existe e é do tipo *string*. Se sim, converte esse valor para *float*, substituindo vírgulas por pontos (útil para converter valores como "1,25" para 1.25). Faz o mesmo para o campo *descontoEUR*.
- Atualiza o documento atual com os novos valores de *valorEUR* e *descontoEUR*.

```
db.Products.find({}).forEach(function(doc) {
  var Dimensao = doc.Dimensao ? doc.Dimensao.toString() : "N/A";
  db.Products.updateOne({_id: doc._id}, {$set: {Dimensao: Dimensao}});
});
```

- Para cada documento da coleção Products, verifica se o campo *Dimensao* existe e, se existir, converte-o para uma *string*. Caso contrário, define o valor como "N/A".
- Atualiza o documento atual com o novo valor de *Dimensao*.

```
        db.Customers.find({}).forEach(function(doc) {
            var CodigoPostal = doc.CodigoPostal ? doc.CodigoPostal.toString() : "N/A";
            var Cidade = doc.Cidade ? doc.Cidade.toString() : "N/A";
            var NIF = doc.NIF ? doc.NIF.toString() : "Consumidor Final";
            db.Customers.updateOne({_id: doc._id}, {$set: {CodigoPostal: CodigoPostal, Cidade: Cidade, NIF: NIF}});
        });
```

- Para cada documento da coleção *Customers*, verifica se o campo *CodigoPostal* existe e, se existir, converte-o para uma *string*. Caso contrário, define o valor como "N/A".

- Faz o mesmo para os campos *Cidade* e *NIF*. No entanto, se o campo *NIF* não existir, é definido como "Consumidor Final".

- Atualiza o documento atual com os novos valores dos campos mencionados.

A formatação do NIF foi feita para ir de encontro com o requisito do enunciado, “se o cliente não forneceu NIF deverá surgir: “Consumidor Final”).

A substituição dos campos vazios ou *NULLs* para “N/A” tem como o objetivo facilitar a nidificação dos campos cuja informação estava em falta.

A formatação de campos com valores numéricos como *valorEUR* e *descontoEUR* teve como objetivo permitir que estes valores sejam usados mais facilmente em futuras operações que serão precisas de forma a cumprir os requisitos.

3.2 Organização e transformação de dados

O enunciado refere que se pretende obter o relatório de vendas de um determinado mês e a lista de clientes que realizaram compras num determinado período. De maneira a possibilitar estas consultas, realizaram-se as operações necessárias (usando métodos como *find()* e *aggregate()*) para transformar e uniformizar os dados nas coleções do MongoDB.

3.2.1 *SalesReport*

Foi realizada a operação de agregação na coleção *SalesHeader* com o objetivo de combinar dados de três coleções diferentes: *Customers*, *SalesLines* e *Products*, para obter um relatório de vendas, ainda sem o filtro do mês. Isto foi feito no MongoDB Compass, seguindo as seguintes etapas (stages):

- Etapa 1 - Junção com a coleção *Customers*: utiliza a operação *$lookup* para combinar os documentos da coleção *SalesHeader* com documentos da coleção *Customers*, com base nos campos *id_cliente* e *id_Cliente*. Como resultado, cada documento na coleção *SalesHeader* terá um novo campo chamado *clienteInfo*, um array que contém informações do cliente correspondente.

- Etapa 2 - Descompactar *clienteInfo*: *$unwind* é utilizado para transformar cada elemento *clienteInfo* num documento separado.

- Etapa 3 - Junção com a coleção *SalesLines*: junção dos documentos resultantes da etapa anterior com a coleção *SalesLines* usando os campos *codigoFatura*.

- Etapa 4 - Descompactar *linhasFatura*: *$unwind* é usado para descompactar o array *linhasFatura*.

- Etapa 5 - Junção com a coleção *Products*: os documentos são combinados com a coleção *Products* usando os campos *linhasFatura.id_Produto* e *id_Artigo*.

- Etapa 6 - Descompactar *linhasFatura.produtoInfo*: *$unwind* usado para descompactar *linhasFatura.produtoInfo*.

- Etapa 7 - Agrupar por código de fatura: *$group* é usado para agrupar os documentos pelo campo *codigoFatura*. Vários campos são extraídos usando o operador *$first* para reter informações relevantes como data, cliente e detalhes do produto. Além disso, é criado um campo *linhasFatura* que é um array contendo detalhes de cada produto vendido, e um campo *totalFatura* que calcula o total da fatura subtraindo os descontos do valor total.

```
db.getCollection('SalesHeader').aggregate(
 [
  {
   $lookup: {
    from: 'Customers',
    localField: 'id_cliente',
    foreignField: 'id_Cliente',
    as: 'clienteInfo'
   }
  },
  {
   $unwind: {
    path: '$clienteInfo',
    preserveNullAndEmptyArrays: false
   }
  },
  {
   $lookup: {
    from: 'SalesLines',
    localField: 'codigoFatura',
    foreignField: 'codigoFatura',
    as: 'linhasFatura'
   }
  },
  {
   $unwind: {
    path: '$linhasFatura',
    preserveNullAndEmptyArrays: false
   }
  },
  {
   $lookup: {
    from: 'Products',
    localField: 'linhasFatura.id_Produto',
    foreignField: 'id_Artigo',
    as: 'linhasFatura.produtoInfo'
   }
  },
  {
   $unwind: {
```

```
      path: '$linhasFatura.produtoInfo',
      preserveNullAndEmptyArrays: false
    }
  },
  {
    $group: {
      _id: '$codigoFatura',
      Data: { $first: '$Data' },
      IdCliente: {
        $first: '$clienteInfo.id_Cliente'
      },
      NomeCliente: {
        $first: '$clienteInfo.Nome'
      },
      Cidade: { $first: '$clienteInfo.Cidade' },
      Pais: { $first: '$clienteInfo.Pais' },
      NIF: { $first: '$clienteInfo.NIF' },
      CodigoPostal: {
        $first: '$clienteInfo.CodigoPostal'
      },
      linhasFatura: {
        $push: {
          IdArtigo:
            '$linhasFatura.produtoInfo.id_Artigo',
          DescArtigo:
            '$linhasFatura.produtoInfo.DescArtigo',
          Quantidade:
            '$linhasFatura.quantidade',
          Valor: '$linhasFatura.valorEUR',
          Desconto: '$linhasFatura.descontoEUR',
          StockAtual:
            '$linhasFatura.produtoInfo.StockAtual'
        }
      },
      totalFatura: {
        $sum: {
          $subtract: [
            '$linhasFatura.valorEUR',
            '$linhasFatura.descontoEUR'
          ]
        }
      }
    }
  }
 ],
 { maxTimeMS: 60000, allowDiskUse: true }
);
```

Ao executar esta agregação, obtemos um conjunto de documentos que detalham cada venda, o cliente que a fez, os produtos comprados, bem como o total da fatura. Os documentos resultantes foram exportados para o ficheiro “SalesReport.json”. De seguida foi criada a coleção “SalesReport” e foi importado este ficheiro. Esta coleção será usada para obter o relatório de vendas de um determinado mês bem como a lista de clientes.

4 API no Basex

Este código representa a integração de uma API RESTful, “api.xqm”, em XQuery no ambiente BaseX, com uma base de dados MongoDB.

```
declare
%rest:path("getClientsByPurchaseDate")
%rest:GET
%rest:query-param("startDate", "{$startDate}")
%rest:query-param("endDate", "{$endDate}")
function page:getClientsByPurchaseDate($startDate as xs:string, $endDate as xs:string) {
   (: consulta mongoDB :)
   let $jsonData := '{
      "collection": "SalesReport",
      "database": "SalesData",
      "dataSource": "PEIEE",
      "filter": {"Data":{' ||
         '"$gte": { "$date": "' || $startDate || 'T00:00:00.000Z" },' ||
         '"$lt": { "$date": "' || $endDate || 'T00:00:00.000Z" }' ||
      '}' ||
   '},' ||
   '"projection": {}' ||
   '}'

   let $req := http:send-request(
     <http:request method='post'>
       <http:header                                                                                    name="api-key"
value="JzJgIoy9Lgcp8LTY4oXJ2dHcGRBsqUmEpFLaq50fsY77rJUbcf5hMWI6sNCexPQB"/>
       <http:body media-type='application/json'/>
     </http:request>,
     "https://eu-west-2.aws.data.mongodb-api.com/app/data-swqwv/endpoint/data/v1/action/find",
     $jsonData
   )

   return
     for $doc in $req[2]/json/documents/_
     return
       <Client>
          <ClientID>{ data($doc/IdCliente) }</ClientID>
          <ClientName>{ data($doc/NomeCliente) }</ClientName>
          <City>{ data($doc/Cidade) }</City>
          <Country>{ data($doc/Pais) }</Country>
          <VATNumber>{ data($doc/NIF) }</VATNumber>
       </Client>
};

declare
%rest:path("getSalesReport")
%rest:GET
%rest:query-param("year", "{$year}")
%rest:query-param("month", "{$month}")
function page:getSalesReport($year as xs:integer, $month as xs:integer) {

   (: Cálculo das datas :)
   let $start-date := fn:string-join(($year, format-number($month, '00'), '01T00:00:00.000Z'), '-')
   let $end-date := fn:string-join(($year, format-number($month + 1, '00'), '01T00:00:00.000Z'), '-')

   (: consulta mongoDB :)
   let $jsonData := '{
      "collection": "SalesReport",
      "database": "SalesData",
      "dataSource": "PEIEE",
      "filter": {"Data": {
```

```
            "$gte": { "$date": "" || $start-date || "" },
            "$lt": { "$date": "" || $end-date || "" }
          }},
      "projection": {}
   }'

   let $req := http:send-request(
     <http:request method='post'>
       <http:header                                                                                    name="api-key"
value="JzJgIoy9Lgcp8LTY4oXJ2dHcGRBsqUmEpFLaq50fsY77rJUbcf5hMWI6sNCexPQB"/>
       <http:body media-type='application/json'/>
     </http:request>,
     "https://eu-west-2.aws.data.mongodb-api.com/app/data-swqwv/endpoint/data/v1/action/find",
     $jsonData
   )

   (: Cálculo de distinctProducts, distinctClients e  totalValue :)
   let $documents := $req[2]/json/documents/_
   let $distinctProducts := distinct-values($documents/linhasFatura/*/IdArtigo)
   let $distinctClients := distinct-values($documents/IdCliente)
   let $totalValue := sum($documents/totalFatura)

   return
  <SalesReport>
   <Seller>
      <VATNumber>509765432</VATNumber>
      <Name>StepUp Shoes</Name>
      <Address>Rua Principal, 50</Address>
      <City>Lousada</City>
      <Country>Portugal</Country>
      <PostalCode>4620-001</PostalCode>
   </Seller>
   <FiscalYear>{$year}</FiscalYear>
   <Month>{$month}</Month>
   <DistinctProductCount>{count($distinctProducts)}</DistinctProductCount>
   <TotalSalesValue>{$totalValue}</TotalSalesValue>
   <DistinctClientCount>{count($distinctClients)}</DistinctClientCount>
   {
     for $doc in $documents
     return
       <SaleDetail>
          <InvoiceID>{data($doc/__id)}</InvoiceID>
          <Date>{data($doc/Data)}</Date>
          <Client>
          <ClientID>{data($doc/IdCliente)}</ClientID>
          <VATNumber>{data($doc/NIF)}</VATNumber>
          <ClientName>{data($doc/NomeCliente)}</ClientName>
          <City>{data($doc/Cidade)}</City>
          <Country>{data($doc/Pais)}</Country>
          <PostalCode>{data($doc/CodigoPostal)}</PostalCode>
          </Client>
             {
             for $line in $doc/linhasFatura/_
             return
             <LineItem>
             <Product>
                <ArticleID>{data($line/IdArtigo)}</ArticleID>
                <ArticleDescription>{data($line/DescArtigo)}</ArticleDescription>
                <CurrentStock>{data($line/StockAtual)}</CurrentStock>
                </Product>
                <Quantity>{data($line/Quantidade)}</Quantity>
                <Value>{data($line/Valor)}</Value>
                <DiscountAmount>{data($line/Desconto)}</DiscountAmount>
             </LineItem>
             }
           <InvoiceTotal>{data($doc/totalFatura)}</InvoiceTotal>
       </SaleDetail>
```

```
    }
  </SalesReport>
};
```

4.1 Lista em XML de clientes num determinado período

4.1.1 Definição e Endpoint

O código acima define uma função chamada *getClientsByPurchaseDate*, que é um serviço RESTful. O cliente pode aceder esta função através do endpoint getClientsByPurchaseDate, fornecendo duas datas como parâmetros (*startDate* e *endDate*). O método "%rest:GET" indica que esta função é acedida usando o método HTTP GET.

4.1.2 Consulta ao MongoDB

A função constrói dinamicamente uma consulta JSON para o MongoDB, que procura os registos na coleção *SalesReport* da base de dados *SalesData* onde a data da venda está entre *startDate* e *endDate*.

4.1.3 Pedido à API

Um pedido POST é enviado para a MongoDB Data API com a consulta construída. Uma chave de API é incluída no cabeçalho no pedido para autenticação.

4.1.4 Transformação dos Resultados

A função processa a resposta recebida da API e transforma os resultadosnuma estrutura XML semelhante com o vocabulário desenvolvido. Para cada documento de resultado, os detalhes do cliente são extraídos e retornados como um elemento <Client>.

4.2 Relatório em XML de vendas num determinado mês

4.2.1 Definição e Endpoint

O código define uma segunda função chamada *getSalesReport*. Os usuários podem acessar essa função via o endpoint *getSalesReport*, fornecendo um ano e um mês como parâmetros. O método “%rest:GET” indica que esta função é acedida usando o método HTTP GET.

4.2.2 Cálculo das Datas

Antes da consulta ao MongoDB, a função calcula as datas de início e término com base no ano e mês fornecidos.

4.2.3 Consulta ao MongoDB

Assim como na primeira função, uma consulta foi construída dinamicamente para procurar registos na coleção *SalesReport*. A consulta filtra as vendas que ocorreram dentro do mês especificado.

4.2.4 Pedido à API e Transformação dos Resultados

Uma consulta JSON é construída e uma solicitação POST é enviada à API MongoDB. Após enviar a consulta para a MongoDB API e receber os resultados, são realizados os cálculos:

- Produtos distintos vendidos no mês.
- Clientes distintos que fizeram compras no mês.
-Valor total de vendas no mês.

4.2.5 Transformação dos Resultados

Os resultados são então estruturados em um formato de relatório XML, de acordo com o vocabulário desenvolvido. O relatório contém detalhes do vendedor, informações fiscais, contagem de produtos distintos, valor total de vendas, contagem de clientes distintos e detalhes de cada venda feita no mês.

4.3 Postman

No postman, preenchendo o tipo de autorização “Basic Auth” com “admin” tanto no username como na password. Exemplos de pedidos do cliente no postman:

http://localhost:8984/getClientsByPurchaseDate?startDate=2018-07-01&endDate=2018-07-31

http://localhost:8984/getSalesReport?year=2018&month=08

5. Conclusões e Trabalho Futuro

O trabalho proposto pela desafiou o grupo de formas que inicialmente não antecipava, exigindo uma abordagem multidisciplinar que abarcou desde a importação de dados, criação de uma API e até a criação de um vocabulário XML específico. Esta experiência serviu para ilustrar a interconexão e a complexidade dos sistemas de informação na prática real.

Durante o desenvolvimento do projeto, o grupo enfrentou obstáculos significativos. A adaptação a ferramentas alternativas, devido à expiração da licença do Oxygen, foi uma lição prática da resiliência e adaptabilidade necessárias no mundo real da tecnologia. Da mesma forma, familiarizar-se e dominar o BaseX, uma ferramenta que não era do domínio anterior do grupo, reforçou a importância do aprendizagem contínuo e da capacidade de se adaptar rapidamente a novos ambientes e tecnologias.

O grupo está confiante de que alcançou os objetivos definidos e, mais importante, expandiu o conjunto de habilidades e compreensão no campo da informática.

Referências WWW

https://www.mongodb.com/docs/atlas/app-services/data-api/generated-endpoints/#call-a-data-api-endpoint

https://www.mongodb.com/docs/atlas/app-services/data-api/examples/

https://docs.basex.org/wiki/JSON_Module

Anexos

api.xqm

documentação XSD

agreagação ClientsReport.txt

agregação SalesReportFINAL.txt

FormatarDados_Script.txt

ClientsReportFinal.json

SalesReport.json

exampleSaleReport.xml

exampleSaleReport2.xml

SalesReport.xsd

user e pass MongoDB.txt