**Disciplina de Processamento Estruturado de Informação**

Ano Letivo de 2024/2025

MedSync

8200489 | João Pedro Teixeira

8220360 | Rafael Saraiva

8210666 | Manuel Pereira

Felgueiras, janeiro de 2025

# Resumo

Este trabalho foi realizado para desenvolver uma solução que integre e padronize a partilha de dados médicos entre hospitais MedSync, promovendo a continuidade e a qualidade dos cuidados de saúde. O projeto foi estruturado em três etapas: extração de dados médicos em formato CSV e organização em MongoDB, disponibilização desses dados via API REST em formato JSON e transformação para documentos XML padronizados utilizando XQuery no BaseX. Os principais resultados incluem uma estrutura de dados eficiente, a geração de relatórios clínicos e de transferências em conformidade com o vocabulário XML definido e a validação do mesmo recorrendo a XSD.

Índice

[Resumo 2](#_Toc187097997)

[Introdução 5](#_Toc187097998)

[Motivação e Objetivos 6](#_Toc187097999)

[Apresentação e discussão dos resultados 7](#_Toc187098000)

[MongoDB 7](#_Toc187098001)

[Transformação e organização dos dados 7](#_Toc187098002)

[Pipelines 8](#_Toc187098003)

[API REST 11](#_Toc187098004)

[BASEX 14](#_Toc187098005)

[XSD 16](#_Toc187098006)

[POSTMAN 17](#_Toc187098007)

[Conclusão 18](#_Toc187098008)

Índice de Ilustrações

[Figura 1 - Estrutura da coleção transfers\_by\_date 7](#_Toc187096531)

[Figura 2 - Estrutura da coleção records\_by\_date 8](#_Toc187096532)

[Figura 3 - Conexão à base de dados no servidor em node.js 10](#_Toc187096533)

[Figura 4 - Controlador das transferências (node.js) 11](#_Toc187096534)

[Figura 5 - Pedido HTTP GET ao servidor para obter os registos Clínicos 12](#_Toc187096535)

[Figura 6 - Pedido HTTP GET ao servidor para obter as transferências 13](#_Toc187096536)

[Figura 7 - XQuery usado para os registos clínicos 13](#_Toc187096537)

[Figura 8 - Resposta do BaseX com os Registos Clínicos padronizado 14](#_Toc187096538)

[Figura 9 - Resposta do BaseX com o relatório de transferências padronizado 14](#_Toc187096539)

[Figura 10 - extrato simples do .xsd 15](#_Toc187096540)

[Figura 11 - Extrato da estrtura do ficheiro .xsd 15](#_Toc187096541)

[Figura 12 - Interface do Postman 16](#_Toc187096542)

# Introdução

A partilha eficiente de dados médicos é fundamental para garantir a qualidade dos cuidados de saúde, especialmente em cenários que envolvem múltiplos hospitais. O projeto MedSync surge como uma resposta à necessidade de integração de informações clínicas entre instituições parceiras, promovendo uma abordagem padronizada para a gestão e troca de dados.

Este relatório apresenta o desenvolvimento de um sistema que suporta a extração, transformação e disponibilização de dados médicos. No âmbito deste trabalho, foi implementado um processo composto pelas seguintes etapas principais: a integração e transformação dos dados em MongoDB, o desenvolvimento de uma API REST em node.js, conversão dos dados em formato JSON para um formato XML padronizado utilizando XQuery e validando os mesmos com um XSD.

Além de contextualizar o caso de estudo, este documento descreve a modelação dos dados em MongoDB, as consultas implementadas para extração de informações relevantes, a criação da API REST e os scripts de transformação XML.

# Motivação e Objetivos

O principal objetivo deste projeto é desenvolver um sistema que suporte o fluxo de dados médicos entre hospitais, desde a extração até à apresentação padronizada dos mesmos. Pretendemos atingir este objetivo por meio das seguintes etapas:

1. Extração e Organização dos Dados: Importar dados médicos disponibilizados em formato CSV para uma base de dados MongoDB, garantindo uma estrutura consistente que atenda aos requisitos definidos.
2. Disponibilização via API REST: Implementar uma API REST para permitir o acesso aos dados integrados em formato JSON, facilitando a consulta e manipulação das informações.
3. Transformação para XML: Utilizar scripts XQuery no BaseX para converter os dados JSON em documentos XML padronizados, de acordo com o vocabulário definido pelo MedSync.

As ferramentas escolhidas para alcançar esses objetivos são:

* **MongoDB**: Para armazenar e estruturar os dados devido à sua flexibilidade e capacidade de lidar com documentos heterogêneos.
* **Express.js** e **Node.js:** Para desenvolver a API REST, oferecendo uma interface eficiente e acessível para os dados.
* **BaseX**: Para realizar a transformação dos dados para XML utilizando XQuery, assegurando a conformidade com o vocabulário exigido.
* **Oxygen XML Editor**: Para a criação dos ficheiros .xsd, de forma a validar os documentos XML gerados pelas consultas

Com estas etapas e ferramentas, pretendemos criar uma solução que permita a integração eficiente dos dados médicos

# Apresentação e discussão dos resultados

## MongoDB

O **MongoDB** é uma base de dados de natureza orientada a documentos, que oferece flexibilidade e facilidade de integração de diferentes tipos de dados. Este modelo permite armazenar informações heterogéneas de maneira organizada e eficiente, características essenciais para atender à complexidade dos dados médicos tratados neste projeto.

### Transformação e organização dos dados

Para atender às necessidades específicas deste projeto e facilitar o processo de gerar relatórios padronizados, foram criadas duas novas coleções chamadas ***transfers\_by\_date*** e ***records\_by\_date*** .

***Transfers\_by\_date -*** consolida informações de transferências médicas, registos clínicos e tratamentos prévios, estruturando os dados de forma otimizada para consultas futuras, organizando por ano e mês.

***Records\_by\_date*** - consolida informações consolidar informações de diagnósticos e tratamentos, organizando por ano e mês.

### Pipelines

Na pipeline que transformou os documentos de transferências em documentos otimizados para este projeto foram utilizadas as seguintes estratégias, respetivamente:

1. **$match**: Filtra os documentos para incluir apenas aqueles em que a Data\_Atendimento no clinical\_records é anterior à Data\_Transferencia.
2. **$match**: Filtra os documentos para incluir aqueles onde a Data\_Tratamento é anterior à Data\_Transferencia,.
3. **$group**: Agrupa os documentos por \_id e coleta informações relevantes, como Destino, Motivo, e os diagnósticos e tratamentos anteriores.
4. **$addFields**: Adiciona os campos Ano\_Transferencia e Mes\_Transferencia extraídos da Data\_Transferencia.
5. **$group**: Agrupa os documentos por ano criando uma lista de transferências para cada mês dentro de cada ano, e depois, agrupa novamente, agora por ano, criando uma estrutura hierárquica com meses e transferências.
6. **$sort**: Ordena os documentos agrupados por ano de forma crescente.

Após a execução desta pipeline, a nova coleção é criada com a seguinte estrutura:

{  
 "\_id": 2023,  
 "Meses":[  
 {  
 "Mes": 1,  
 "Detalhes": [  
 {  
 "Destino": "Centro Cardiológico",  
 "ID\_Transferencia": 1638,  
 "ID\_Paciente": 13348,  
 "ID\_Profissional": 159,  
 "Data\_Transferencia": {  
 "$date": "2023-01-15T00:00:00.000Z" },  
 "Motivo": "Exames complementares de diagnóstico",  
 "Tipo\_Transferencia": "Urgente",  
 "Diagnosticos\_Previos": [  
 {  
 "Tipo\_Diagnostico": "Principal",  
 "Codigo\_CID10": "O80",  
 "Descricao\_Diagnostico": "Parto único espontâneo" }  
 ],  
 "Tratamentos\_Previos": [  
 {  
 "ID\_Tratamento": 293849,  
 "Tipo\_Tratamento": "Losartana 50mg",  
 "Realizado": "Sim" }  
 ]  
 }  
 ]  
 }  
 ],  
 "Ano": 2023  
}

Figura 1 - Estrutura da coleção transfers\_by\_date

Na pipeline que transformou os documentos de registos clínicos em documentos otimizados para este projeto foram utilizadas as seguintes estratégias, respetivamente:

1. **$group**: Agrupa os documentos por ID\_Atendimento e coleta informações de diagnósticos, bem como os dados do paciente e profissional.
2. **$addFields (Tratamentos)**: Combina os tratamentos e as atualizações de tratamentos numa única lista.
3. **$addFields (Ano\_Atendimento, Mes\_Atendimento)**: Adiciona campos para o ano e mês extraídos da Data\_Atendimento.
4. **$group**: Agrupa os documentos por ano e mês (Ano\_Atendimento e Mes\_Atendimento), criando uma lista de registos para cada mês, e depois, agrupa os documentos por ano.
5. **$sort**: Ordena os documentos por ano de forma crescente.

Após a execução desta pipeline, a nova coleção é criada com a seguinte estrutura:

{  
 "\_id": 2022,  
 "Meses": [  
 {  
 "Mes": 12,  
 "Registos": [  
 {  
 "ID\_Registo\_Clinico": 31,  
 "ID\_Paciente": 4853,  
 "ID\_Profissional": 67,  
 "Data\_Atendimento": {  
 "$date": "2022-12-24T00:00:00.000Z" },  
 "Diagnosticos": [  
 {  
 "Tipo\_Diagnostico": "Principal",  
 "Codigo\_CID10": "N39",  
 "Descricao\_Diagnostico": "Outros distúrbios do trato urinário" }  
 ],  
 "Tratamentos": [  
 {  
 "\_id": {  
 "$oid": "676724464bf1a1b510c022e4" },  
 "ID\_Tratamento": 174779,  
 "ID\_Registo\_Clinico": 31,  
 "Tipo\_Tratamento": "Aplicação de insulina",  
 "Realizado": "Sim" }  
 ]  
 }  
 ],  
 "Pacientes": [  
 {  
 "\_id": {  
 "$oid": "676723564bf1a1b510bce63a" },  
 "ID\_Paciente": 4853,  
 "Nome\_Completo": "Eric Wilson",  
 "Data\_Nascimento": {  
 "$date": "1984-05-09T00:00:00.000Z" },  
 "Género": "M",  
 "Email": "desconhecido",  
 "Data\_Registo": {  
 "$date": "2019-11-26T00:00:00.000Z" }  
 }  
 ]  
 }  
 ],  
 "Ano": 2022  
}

Figura 2 - Estrutura da coleção records\_by\_date

Mais tarde, devido à necessidade de obter estatísticas mensais referentes ao mês que o utilizador pede, foram criadas duas pipeline capaz de filtrar pelo mês pedido e gerar estatísticas com base no conteúdo da coleção.

No seguinte ponto abordamos o essencial da pipeline que devolve os registos médicos de um mês.

* **$addFields (estatisticas)**: Adiciona um novo campo estatisticas, que contém:
  + **faixaEtaria**: Calcula a distribuição de pacientes por faixa etária, usando a idade calculada com base no ano de nascimento (Data\_Nascimento). A faixa etária é dividida em 3 categorias: 0-18, 19-65 e 65+. O **$reduce** percorre os pacientes e conta quantos se encaixam em cada faixa etária.
  + **porGenero**: Calcula a distribuição de pacientes por género (M para masculino e F para feminino), utilizando o campo Género de cada paciente.
  + **totalTratamentos**: Conta o total de tratamentos realizados durante o mês, somando o número de tratamentos (Tratamentos) presentes no campo Registos.

Nos dois seguintes pontos abordamos o essencial da pipeline que devolve as transferências de um mês.

* **$group**: Reagrupa as transferências e calcula o total de transferências, além de criar dois conjuntos:
  + **transferenciasPorMotivo:** Conjunto de motivos distintos das transferências.
  + **transferenciasPorTipo:** Conjunto de tipos distintos de transferências.
* **$project**: Formata o documento final, incluindo:
  + **totalTransferencias:** Total de transferências no mês.
  + **transferencias:** A lista completa de transferências.
  + **porMotivo**: Para cada motivo distinto de transferência, conta quantas transferências possuem esse motivo.
  + **porTipo**: Para cada tipo distinto de transferência, conta quantas transferências possuem esse tipo.

## API REST

Devido à descontinuação da funcionalidade de DATA API do MongoDB sentimos a necessidade de criar uma API para solucionar este problema. A API REST baseia-se em dois métodos que servem de ligação entre a Base de Dados MongoDB e o BaseX. Esta API faz o papel de servidor sendo que em cada método realiza um pedido http ao MongoDB para obter os dados em formato JSON.

Foi necessário estabelecer uma ligação a base de dados e a API, para a obter as informações corretas fornecidas pelos hospitais. A figura seguinte mostra como foi estabelecida a ligação.

const { MongoClient } = ***require***('mongodb');  
  
const uri = 'mongodb+srv://pedro:pedro@medsync.vziha.mongodb.net/?retryWrites=true&w=majority&appName=MedSync';  
const dbName = 'data';  
let db;  
  
async function connectToDb() {  
 if (!db) {  
 const client = new MongoClient(uri);  
 await client.connect();  
 ***console***.log('Connected to MongoDB');  
 db = client.db(dbName);  
 }  
 return db;  
}  
  
***module***.exports = connectToDb;

Figura 3 - Conexão à base de dados no servidor em node.js

A API tem dois controladores simples com apenas um método que serve para realizar os pedidos de informação à base de dados. Esta informação é devolvida em formato .json e posteriormente desserializada no BaseX.

A figura seguinte mostra um dos controladores criados.

const connectToDb = ***require***('../db');  
  
***exports***.getTransferReportByMonth = async (req, res) => {  
 const {year, month} = req.query;  
  
 if (!year || !month) {  
 return res.status(400).json({success: false, message: "Year and month are required."});  
 }  
  
 const yearInt = parseInt(year, 10);  
 const monthInt = parseInt(month, 10);  
  
 try {  
 const db = await connectToDb();  
 const transfersCollection = db.collection('transfers\_by\_date');  
  
 const pipeline = […];  
  
 const result = await transfersCollection.aggregate(pipeline).toArray();  
  
 res.status(200).json({  
 success: true,  
 data: result,  
 });  
 } catch (error) {  
 ***console***.error(error);  
 res.status(500).json({  
 success: false,  
 message: 'An error occurred while generating the report.',  
 });  
 }  
};

Figura 4 - Controlador das transferências (node.js)

As seguintes figuras mostram como são devolvidos os dados a partir da API.

Uma imagem com texto, captura de ecrã, número, Paralelo

Descrição gerada automaticamente

Figura 5 - Pedido HTTP GET ao servidor para obter os registos Clínicos

Uma imagem com texto, captura de ecrã, número, software

Descrição gerada automaticamente

Figura 6 - Pedido HTTP GET ao servidor para obter as transferências

A informação devolvida é usada posteriormente pelo BaseX.

## BASEX

O papel do BaseX baseia-se a manipulação dos dados, onde estes são entregues no formato .xml padronizado com a utilização do XQuery e uma verificação com XSD.

Com os dados obtidos previamente, o BaseX padroniza-os de modo a devolver um xml com um relatório com estatísticas dos registos clínicos usando o XQuery, como demonstrado na seguinte figura.

1. let $xml := element RegistosClinicos {

2. for $registo in $json/data/\_/Registos/\_

3. return element RegistoClinico {

4. element ID\_Registo\_Clinico { $registo/ID\_\_Registo\_\_Clinico/text() },

5. element ID\_Paciente { $registo/ID\_\_Paciente/text() },

6. element ID\_Profissional { $registo/ID\_\_Profissional/text() },

7. element Data\_Atendimento { substring-before($registo/Data\_\_Atendimento, "T") },

8. element Diagnosticos {},

11. element Tratamentos {}

13. },

14. element Pacientes {

15. },

16. element Estatisticas {

17. element FaixaEtaria {},

19. element PorGenero {},

21. element TotalTratamentos { $json/data/\_/estatisticas/totalTratamentos/text() }

22. }

23. }

Figura 7 - XQuery usado para os registos clínicos

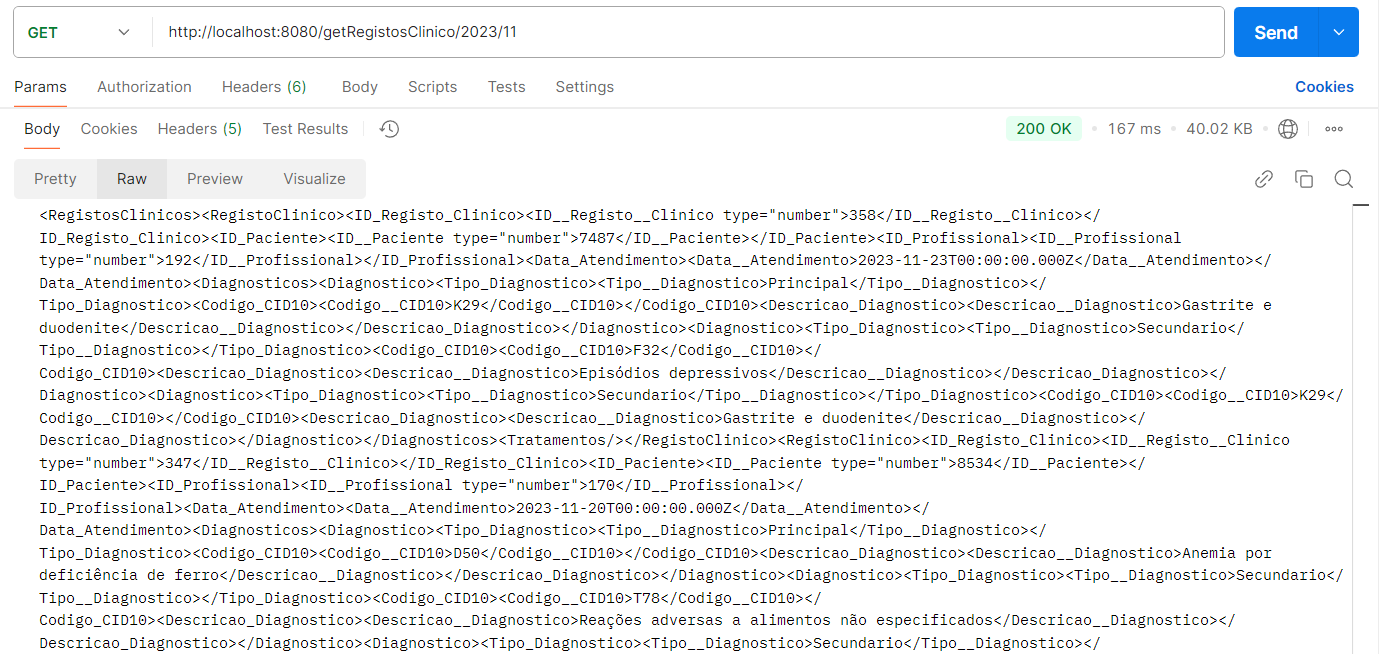


Figura 8 - Resposta do BaseX com os Registos Clínicos padronizado

Como para os registos clínicos, o BaseX padroniza os dados das transferências e devolve-os num ficheiro xml.

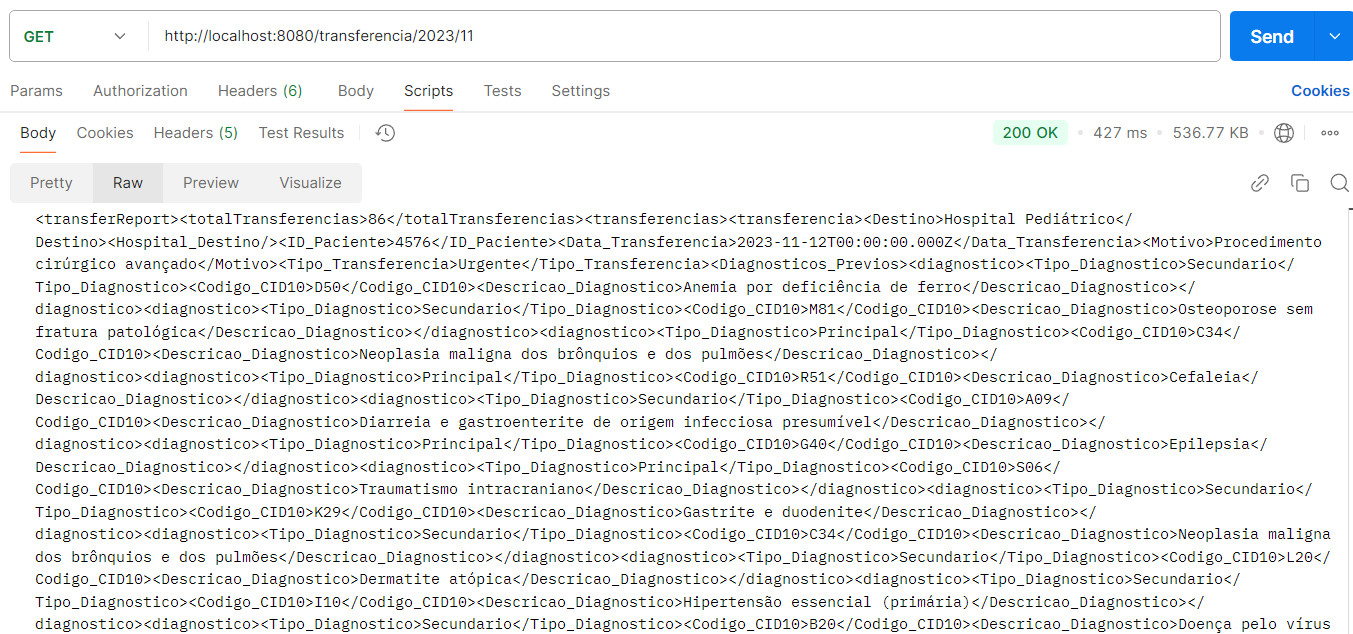


Figura 9 - Resposta do BaseX com o relatório de transferências padronizado

## XSD

O XSD é um formato de ficheiro que serve para a verificação de ficheiros XML.

Os ficheiros XSD usados verificam a integridade da estrutura e a informação introduzida dos ficheiros XML devolvidos pelo BaseX.

As seguintes figuras demonstram como é estruturada a informação dos relatórios de transferências e registos clínicos, respetivamente:

<xs:element name="transferReport">

<xs:complexType>

<xs:sequence>

<xs:element name="totalTransferencias" type="xs:integer"/>

<xs:element name="transferencias">

<xs:element name="porMotivo">

<xs:element name="porTipo">

</xs:sequence>

</xs:complexType>

</xs:element>

Figura 10 - extrato simples do .xsd das transferências

<xs:element name="RegistosClinicos">

<xs:complexType>

<xs:sequence>

<xs:element name="RegistoClinico" maxOccurs="unbounded">

<xs:element name="Pacientes">

<xs:element name="Estatisticas">

<xs:complexType>

<xs:sequence>

<xs:element name="FaixaEtaria">

<xs:element name="PorGenero">

<xs:element name="TotalTratamentos" type="xs:integer"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

Figura 11 - Extrato da estrutura do ficheiro .xsd dos registos clínicos

## POSTMAN

O Postman foi usado para a verificar o bom funcionamento do projeto e nos diferentes pedidos HTTP. Para tal verificação, basta indicar o URL que se quer realizar o pedido HTTP, o método do pedido e nos parâmetros o mês e o ano pretendido. A figura seguinte mostra a interface do Postman.

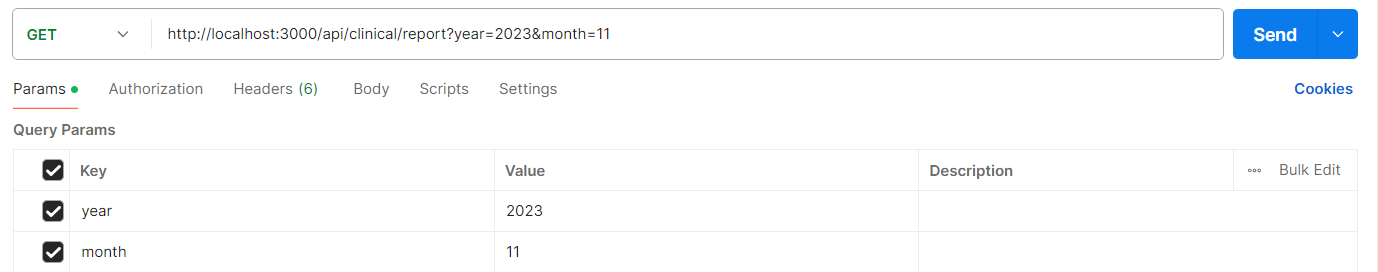


Figura 12 - Interface do Postman

Encontra-se na pasta entregue no moodle a coleção do Postman que foi exportada pela equipa.

# Conclusão

Em suma, o projeto foi finalizado com sucesso, cumprindo os requisitos estipulados desde o início. Todas as funcionalidades foram implementadas conforme o planejado, com um sistema robusto que atende tanto ao lado da BD, consultas e processo de gerar o produto final pretendido, os relatórios em XML.

O grupo manteve um desempenho positivo e mostrou-se interessado durante a execução do projeto, o que foi essencial para a conclusão atempada do mesmo.