ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

P.PORTO

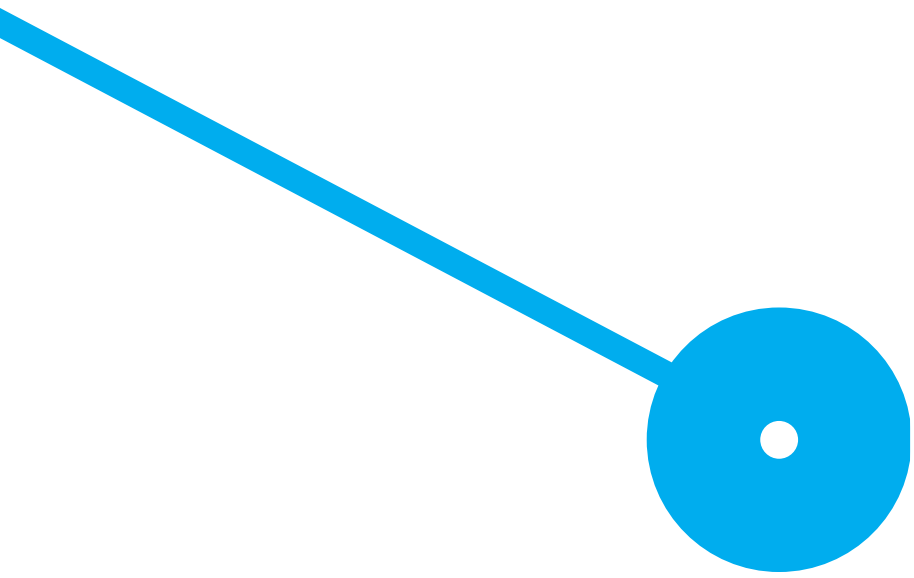
CTeSP CURSO TÉCNICO SUPERIOR PROFISSIONAL
Desenvolvimento para a Web e Dispositivos Móveis

Hotel Reservas

(8240558) Tiago Queirós

(8240544) Diogo Costa

2025/2026



1. Índice

Conteúdo

2. Introdução

2.1.Objetivos Gerais

O presente projeto tem como objetivo o desenvolvimento de uma aplicação web completa para gestão de reservas de uma cadeia hoteleira, demonstrando a integração de múltiplas tecnologias de estruturação e processamento de dados.

A aplicação permite:

· Estruturar dados de reservas em formato XML com validação XSD rigorosa

· Consultar e transformar dados usando XQuery em ambiente BaseX

· Persistir e agregar dados em MongoDB Atlas (base de dados NoSQL)

· Expor funcionalidades através de uma API RESTful em Node.js/Express

· Interagir com o sistema através de uma interface web moderna e responsiva

2.2.Objetivos específicos

- Definir esquema de validação - XML Schema Definition (XSD)Validar em tempo real
- Implementar consultas declarativas - XQuery 3.1
- Criar API REST para consultas - RESTXQ Module / BaseX
- Migrar dados para NoSQL - MongoDB Atlas
- Implementar agregações - MongoDB Aggregation Framework
- Desenvolver backend robusto - Node.js / Express
- Criar interface responsiva - HTML5 / CSS3 / JavaScript
- Visualizar estatísticas - Chart.js
- Documentar API - Swagger / OpenAPI 3.0
- Deploy em cloud - Vercel / Render

2.3.Motivação do Projeto

O setor hoteleiro necessita de sistemas eficientes para gerir reservas,

acompanhar estatísticas e garantir a integridade dos dados e o professor é muito fixe por isso aplicamo-nos mais neste projeto para dar um trabalho inovador .

Este projeto simula um cenário real onde:

Dados são estruturados em XML para interoperabilidade

Validações garantem a consistência das reservas

Análises agregadas ajudam na tomada de decisões

Uma API permite integração com outros sistemas

Uma interface facilita a gestão diária

3. Âmbito do Sistema

O Sistema de Gestão de Reservas de Hotel é uma plataforma web que permite gerir todas as reservas de uma cadeia hoteleira com presença em 5 cidades portuguesas.

O sistema suporta operações CRUD completas, visualização de estatísticas e análise de dados agregados.

3.1.Unidades Hoteleiras

Lisboa(LS) - Unidade principal, zona centro

Porto (PO) - Unidade norte, zona ribeirinha

Coimbra (CB) - Unidade centro, proximidade universidade

Faro (FR) - Unidade sul, zona turística

Braga (BR) - Unidade minho, centro histórico

3.2.Entidades do Sistema

HÓSPEDE
NumeroCliente
Nome
Nif
Email
Telefone

RESERVA
NumeroReserva
Unidade
CheckIn
Checkout
ValorTotal
Quarto

SERVIÇO ADICIONAL
Tipo (Enum)
Nome
Preco
Quantidade

3.3.Entidades do Sistema-JSON

```
{
  "reserva": {
    "numeroReserva": "R001",
    "hospede": {
      "numeroCliente": "CLI123",
      "nome": "string",
      "nif": "string",
      "email": "string",
      "telefone": "string"
    },
    "unidade": "LS|PO|CB|FR|BR",
    "checkIn": "date",
    "checkOut": "date",
    "valorTotal": "decimal",
    "servicosAdicionais": [
      {
        "tipo":
"spa|restaurante|transporte|outros",
        "nome": "string",
        "preco": "decimal",
        "quantidade": "integer"
      }
    ]
  }
}
```

Figura 1-JSON

3.4.Entidades do Sistema - Tipos de Serviços Adicionais

- spa - Massagens, tratamentos de beleza, relaxamento
- restaurante - Refeições, room service, buffet
- transporte - Transfers, táxi, aluguer de veículos
- outros - Lavandaria, mini-bar, experiência

3.5.Divisão da equipa

Nome	Função	Desenvolvimento
Tiago Queirós	Frontend+ Backend Developer + MONGO MASTER	API Node.js, MongoDB, Deployment, Interface web, CSS, JavaScript
Diogo Costa	Database Constructor + API Builder + XML MASTER	Arquitetura ,Schema XSD, Consultas,XQuery, BaseX, RESTXQ , Chart.js ,Mongo Charts

Metodologia de Trabalho:

- Controlo de versões com Git/GitHub
- Divisão de tarefas por sprints
- Revisão de código entre membros

4. Arquitetura do Sistema

4.1.Visão Geral

O sistema combina duas abordagens complementares:

PARTE 1 - XML e BaseX (Estruturação e Consultas):

- Vocabulário XML para estruturar dados de reservas
- Esquema XSD para validação rigorosa dos dados
- Consultas XQuery para extração de informação
- API RESTXQ para expor endpoints via BaseX

PARTE 2 - MongoDB e Node.js (Persistência e Web):

- Camada de Apresentação (Frontend - Vercel)
- Camada de Lógica de Negócio (Backend API - Render)
- Camada de Dados (MongoDB Atlas)
- Dashboard com gráficos e estatísticas

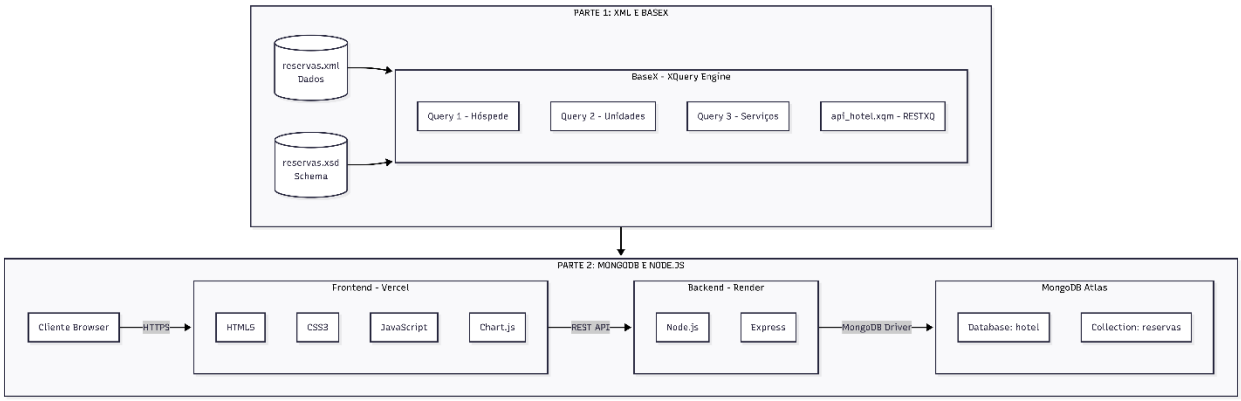


Figura 2-Arquitetura do Sistema

4.2.Fluxo de Dados

- 1. Utilizador acede à interface web (Vercel)
- 2. Frontend faz pedidos fetch() à API (Render)
- 3. Backend valida e processa o pedido
- 4. MongoDB executa queries/aggregations
- 5. Resposta JSON retorna ao frontend
- 6. Interface atualiza dinamicamente (sem refresh)

4.3. Decisões Tecnológicas

- Node.js escolhido pela eficiência com I/O assíncrono
- Express pela simplicidade e middlewares disponíveis
- MongoDB pela flexibilidade do schema e aggregation framework
- Vercel/Render pelo free tier generoso e deploy automático
- Chart.js pela integração simples e gráficos responsivos

5. MODELAÇÃO DE DADOS XML

5.1.Estrutura XML (reservas.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<reservas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="reservas.xsd">

 <reserva numeroReserva="R001">
  <hospede>
   <numeroCliente>CLI123</numeroCliente>
   <nome>João Silva</nome>
   <nif>123456789</nif>
   <email>joao.silva@example.com</email>
   <telefone>912345678</telefone>
  </hospede>
  <unidade>LS</unidade>
  <checkIn>2025-01-10</checkIn>
  <checkOut>2025-01-15</checkOut>
  <valorTotal>550.00</valorTotal>
  <servicosAdicionais>
   <servicoAdicional>
    <tipo>spa</tipo>
```

```
        <nome>Massagem Relaxante</nome>
        <preco>60.00</preco>
        <quantidade>2</quantidade>
      </servicoAdicional>
      <servicoAdicional>
        <tipo>restaurante</tipo>
        <nome>Jantar Gourmet</nome>
        <preco>40.00</preco>
        <quantidade>1</quantidade>
      </servicoAdicional>
    </servicosAdicionais>
  </reserva>
</reservas>
```

5.2.Esquema XSD (reservas.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <!-- Tipo para código de cliente: 3 letras + 3 dígitos -->
 <xs:simpleType name="tipoNumeroCliente">
  <xs:restriction base="xs:string">
   <xs:pattern value="[A-Z]{3}[0-9]{3}"/>
  </xs:restriction>
 </xs:simpleType>

 <!-- Tipo para NIF: 9 dígitos -->
 <xs:simpleType name="tipoNIF">
  <xs:restriction base="xs:string">
   <xs:pattern value="[0-9]{9}"/>
  </xs:restriction>
 </xs:simpleType>

 <!-- Tipo para telefone: 9 dígitos começando por 9 -->
 <xs:simpleType name="tipoTelefone">
  <xs:restriction base="xs:string">
   <xs:pattern value="9[0-9]{8}"/>
```

```
  </xs:restriction>
</xs:simpleType>

<!-- Tipo para email -->
<xs:simpleType name="tipoEmail">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"/>
  </xs:restriction>
</xs:simpleType>

<!-- Tipo para número de reserva: R seguido de 3 dígitos -->
<xs:simpleType name="tipoNumeroReserva">
  <xs:restriction base="xs:string">
    <xs:pattern value="R[0-9]{3}"/>
  </xs:restriction>
</xs:simpleType>

<!-- Tipo para serviços adicionais (Enumeration) -->
<xs:simpleType name="tipoServico">
  <xs:restriction base="xs:string">
    <xs:enumeration value="spa"/>
    <xs:enumeration value="restaurante"/>
    <xs:enumeration value="transporte"/>
```

```
    <xs:enumeration value="outros"/>
  </xs:restriction>
</xs:simpleType>

<!-- Unidades: LS, PO, CB, FR, BR -->
<xs:simpleType name="tipoUnidade">
  <xs:restriction base="xs:string">
    <xs:enumeration value="LS"/>
    <xs:enumeration value="PO"/>
    <xs:enumeration value="CB"/>
    <xs:enumeration value="FR"/>
    <xs:enumeration value="BR"/>
  </xs:restriction>
</xs:simpleType>

<!-- Elemento raiz -->
<xs:element name="reservas">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="reserva" maxOccurs="unbounded">
        <!-- ... estrutura completa ... -->
      </xs:element>
    </xs:sequence>
```

```
  </xs:complexType>

  </xs:element>


</xs:schema>
```

5.3.Tipos de Dados Personalizados

Nome do Tipo	Expressão Regular	Exemplos Válidos
tipoNumeroCliente	[A-Z]{3}[0-9]{3}	CLI123, ABC456, XYZ789
tipoNIF	[0-9]{9}	123456789, 987654321
tipoTelefone	9[0-9]{8}	912345678, 963852741
tipoEmail	regex padrão email	user@example.com
tipoNumeroReserva	R[0-9]{3}	R001, R123, R999
tipoServico	enum	spa, restaurante, outros
tipoUnidade	enum	LS, PO, CB, FR, BR

5.4.Restrições e Validações

- numeroReserva - atributo obrigatório (use="required")
- servicoAdicional - máximo 4 por reserva (maxOccurs="4")
- unidade - apenas valores do enum (xs:enumeration)
- hóspede - todos os campos são obrigatórios
- datas - tipo xs:date (YYYY-MM-DD)
- valores - tipo xs:decimal

6. CONSULTAS XQUERY

6.1. Ambiente BaseX

As consultas XQuery foram desenvolvidas e testadas no BaseX:

Versão: BaseX 10.x

Interface: BaseX GUI e BaseX HTTP Server

Ficheiros: .xq para consultas autónomas, .xqm para módulos

6.1. Query 1 - Reservas por Hóspede

Ficheiro: query1_reservas_hospede.xq

Query 1: Apresentar as reservas de um dado hóspede

Parâmetro: numeroCliente (ex: CLI123)

```
declare variable $numeroCliente as xs:string external := "CLI123";

let $reservas := doc("reservas.xml")//reserva[hospede/numeroCliente = $numeroCliente]

return
 <resultado>
  <hospede>{$numeroCliente}</hospede>
  <totalReservas>{count($reservas)}</totalReservas>
```

```
<reservas>
{
 for $reserva in $reservas
 return
  <reserva numero="{$reserva/@numeroReserva}">
   <nome>{$reserva/hospede/nome/text()}</nome>
   <unidade>{$reserva/unidade/text()}</unidade>
   <checkIn>{$reserva/checkIn/text()}</checkIn>
   <checkOut>{$reserva/checkOut/text()}</checkOut>
   <valorTotal>{$reserva/valorTotal/text()}</valorTotal>
   <servicosAdicionais>
   {
    for $servico in $reserva/servicosAdicionais/servicoAdicional
    return
     <servico>
      <tipo>{$servico/tipo/text()}</tipo>
      <nome>{$servico/nome/text()}</nome>
      <preco>{$servico/preco/text()}</preco>
      <quantidade>{$servico/quantidade/text()}</quantidade>
     </servico>
   }
   </servicosAdicionais>
  </reserva>
```

```
  }
  </reservas>
 </resultado>
```

Resultado Exemplo:

```
<resultado>
 <hospede>CLI123</hospede>
 <totalReservas>2</totalReservas>
 <reservas>
  <reserva numero="R001">
   <nome>João Silva</nome>
   <unidade>LS</unidade>
   <checkIn>2025-01-10</checkIn>
   <checkOut>2025-01-15</checkOut>
   <valorTotal>550.00</valorTotal>
  </reserva>
  <!-- ... mais reservas ... -->
 </reservas>
</resultado>
```

6.2. Query 2 - Reservas por Unidade

Ficheiro: query2_reservas_por_unidade.xq

Query 2 : Agrupa e conta reservas por cada unidade hoteleira

```
let $reservas := doc("reservas.xml")//reserva
return
 <resultado>
  <totalGeral>{count($reservas)}</totalGeral>
  <unidades>
  {
   for $unidade in distinct-values($reservas/unidade)
   let $quantidade := count($reservas[unidade = $unidade])
   order by $unidade
   return
    <unidade codigo="{$unidade}">
     <nome>
     {
      switch($unidade)
       case "LS" return "Lisboa"
       case "PO" return "Porto"
       case "CB" return "Coimbra"
```

```
            case "FR" return "Faro"
            case "BR" return "Braga"
            default return "Desconhecida"
        }
        </nome>
        <quantidade>{$quantidade}</quantidade>
      </unidade>
  }
  </unidades>
 </resultado>
```

6.3.Query 3 - Total de Serviços Vendidos

Ficheiro: query3_total_servicos.xq

Query 3: Calcula estatísticas de serviços adicionais por tipo

```
let $servicos := doc("reservas.xml")//servicoAdicional
return
 <resultado>
  <totalServicos>{count($servicos)}</totalServicos>
  <totalQuantidade>{sum($servicos/quantidade)}</totalQuantidade>
  <valorTotal>{sum($servicos/preco * $servicos/quantidade)}</valorTotal>
  <servicosPorTipo>
```

```
 {
  for $tipo in distinct-values($servicos/tipo)
  let $servicosDoTipo := $servicos[tipo = $tipo]
  let $quantidadeTotal := sum($servicosDoTipo/quantidade)
  let $valorTotal := sum($servicosDoTipo/preco * $servicosDoTipo/quantidade)
  order by $quantidadeTotal descending
  return
    <tipo nome="{$tipo}">
      <quantidade>{$quantidadeTotal}</quantidade>
      <valor>{$valorTotal}</valor>
    </tipo>
 }
 </servicosPorTipo>
</resultado>
```

6.4.Módulo RESTXQ (API REST em XQuery)

Ficheiro: api_hotel.xqm

O módulo RESTXQ permite expor consultas XQuery como endpoints REST:

```
module namespace api = "http://hotel.com/api";

(:~ GET /api/reservas/hospede/{numeroCliente} :)
declare
  %rest:GET
  %rest:path("/api/reservas/hospede/{$numeroCliente}")
  %rest:produces("application/xml", "text/xml")
function api:reservas-hospede($numeroCliente as xs:string) {
  (: implementação :)
};

(:~ GET /api/reservas/unidade :)
declare
  %rest:GET
  %rest:path("/api/reservas/unidade")
  %rest:produces("application/xml", "text/xml")
function api:reservas-por-unidade() {
  (: implementação :)
```

```
};


(:~ GET /api/servicos/total :)
declare
  %rest:GET
  %rest:path("/api/servicos/total")
  %rest:produces("application/xml", "text/xml")
function api:total-servicos() {
  (: implementação :)
};
```

7. MONGODB E AGGREGATION FRAMEWORK

7.1.Configuração MongoDB Atlas

- Cluster: M0 Sandbox (Free Tier)
- Região: AWS eu-west-1 (Ireland)
- Database: hotel
- Collection: reservas
- Network Access: IP Whitelist configurado

7.2.Migração XML para JSON/BSON

Os dados XML foram convertidos para o formato MongoDB:

```
{
  "_id": ObjectId("..."),
  "numeroReserva": "R001",
  "hospede": {
    "numeroCliente": "CLI123",
    "nome": "João Silva",
    "nif": "123456789",
    "email": "joao.silva@example.com",
    "telefone": "912345678"
  },
```

```
  "unidade": "LS",
  "checkIn": "2025-01-10",
  "checkOut": "2025-01-15",
  "valorTotal": 550.00,
  "quarto": "101",
  "servicosAdicionais": [
    {
      "tipo": "spa",
      "nome": "Massagem Relaxante",
      "preco": 60.00,
      "quantidade": 2
    }
  ]
}
```

7.3.Consultas de Agregação (Aggregation Pipeline)

7.3.1. Query 1: Serviços Vendidos por Tipo

```
db.reservas.aggregate([
  { $unwind: "$servicosAdicionais" },
  {
    $group: {
```

```
    _id: "$servicosAdicionais.tipo",
    totalServicos: { $sum: 1 },
    quantidadeTotal: { $sum: "$servicosAdicionais.quantidade" },
    valorTotal: {
      $sum: {
        $multiply: ["$servicosAdicionais.preco", "$servicosAdicionais.quantidade"]
      }
    }
  }
 },
 { $sort: { quantidadeTotal: -1 } },
 {
  $project: {
    _id: 0,
    tipo: "$_id",
    totalServicos: 1,
    quantidadeTotal: 1,
    valorTotal: { $round: ["$valorTotal", 2] }
  }
 }
])
```

7.3.2. Query 2: Reservas por Unidade

```
db.reservas.aggregate([
 {
  $group: {
   _id: "$unidade",
   quantidadeReservas: { $count: {} },
   valorTotalReservas: { $sum: "$valorTotal" }
  }
 },
 { $sort: { quantidadeReservas: -1 } },
 {
  $project: {
   _id: 0,
   unidade: "$_id",
   nomeUnidade: {
    $switch: {
     branches: [
      { case: { $eq: ["$_id", "LS"] }, then: "Lisboa" },
      { case: { $eq: ["$_id", "PO"] }, then: "Porto" },
      { case: { $eq: ["$_id", "CB"] }, then: "Coimbra" },
      { case: { $eq: ["$_id", "FR"] }, then: "Faro" },
      { case: { $eq: ["$_id", "BR"] }, then: "Braga" }
```

```
        ],
        default: "Desconhecida"
      }
    },
    quantidadeReservas: 1,
    valorTotalReservas: { $round: ["$valorTotalReservas", 2] }
  }
 }
])
```

7.3.3. Query 3: Valor Médio das Reservas

```
db.reservas.aggregate([
 {
  $group: {
    _id: null,
    valorMedio: { $avg: "$valorTotal" },
    totalReservas: { $sum: 1 },
    valorTotalGeral: { $sum: "$valorTotal" }
  }
 },
 {
```

```
  $project: {
    _id: 0,
    valorMedio: { $round: ["$valorMedio", 2] },
    totalReservas: 1,
    valorTotalGeral: { $round: ["$valorTotalGeral", 2] }
  }
 }
])
```

7.3.4. Query 4: Maior Reserva (Top 1)

```
db.reservas.aggregate([
 { $sort: { valorTotal: -1 } },
 { $limit: 1 },
 {
  $project: {
    _id: 0,
    numeroReserva: 1,
    "hospede.nome": 1,
    "hospede.numeroCliente": 1,
    unidade: 1,
    checkIn: 1,
```

```
    checkOut: 1,
    valorTotal: 1,
    totalServicos: { $size: "$servicosAdicionais" }
   }
  }
])
```

7.4.Operadores Utilizados

Operador	Descrição
$unwind	Deconstrução de arrays para documentos individuais
$group	Agrupamento com acumuladores
$sum	Soma de valores
$avg	Média de valores
$count	Contagem de documentos
$sort	Ordenação (ascendente/descendente)
$limit	Limitar número de resultados
$project	Projeção/formatação de campos
$round	Arredondamento de decimais
$multiply	Multiplicação de valores
$switch	Condicional (similar a switch/case)
$size	Tamanho de um array

8. API REST (Node.js/Express)

8.1.Configuração do Servidor

server.js

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const { MongoClient } = require('mongodb');

const app = express();
const PORT = process.env.PORT || 3000;

app.use(cors());
app.use(express.json());


app.use((req, res, next) => {
    console.log(`[Request] ${req.method} ${req.path}`);
    next();
});
```

Figura 3-Configuração do Servidor

Este código configura o servidor Node.js/Express:

- **dotenv** - Carrega variáveis de ambiente do ficheiro .env(ex: MONGODB_URI)
- **express** - Framework web para criar a API REST
- **cors** - Permite pedidos de outros domínios (frontend Vercel → backend Render)
- **MongoClient** - Driver para conectar ao MongoDB Atlas
- **PORT 3000** - Porta onde o servidor escuta

- **app.use(cors())** - Ativa CORS para todos os pedidos
- **app.use(express.json())** - Permite receber JSON no body dos pedidos
- **Middleware de logging** - Regista todos os pedidos (método + rota) na consola para debug

8.2.Conexão à Base de Dados

server.js

```
let db;

async function connectDB() {
    try {
        const client = new MongoClient(process.env.MONGODB_URI);
        await client.connect();
        db = client.db('hotel');
        console.log('Conectado ao MongoDB Atlas!');
    } catch (error) {
        console.error('Erro ao conectar ao MongoDB:', error);
        process.exit(1);
    }
}
```

Figura 4-Conexão à Base de dados

Função assíncrona que estabelece ligação ao MongoDB Atlas usando a URI definida nas variáveis de ambiente, seleciona a base de dados "hotel" e encerra o servidor em caso de erro de conexão.

8.3.Endpoints Implementados (11 Rotas)

Método	Endpoint	Descrição
GET	/api/reservas	Listar todas as reservas
GET	/api/reservas/:numeroReserva	Obter reserva específica
GET	/api/reservas/hospede/:id	Reservas por hóspede
GET	/api/reservas/unidade	Estatísticas por unidade
GET	/api/reservas/media	Valor médio das reservas
GET	/api/reservas/maior	Maior reserva
GET	/api/servicos/total	Serviços por tipo
GET	/api/estatisticas	Dashboard completo
POST	/api/reservas	Criar nova reserva
PUT	/api/reservas/:numeroReserva	Atualizar reserva
DELETE	/api/reservas/:numeroReserva	Eliminar reserva

8.4.Validações XSD no Backend

```
const XSD_RULES = {
    numeroReserva: /^RES\d{3}$/,
    numeroCliente: /^[A-Z]{3}\d{3}$/,
    unidades: ['LS', 'PO', 'CB', 'FR', 'BR'],
    maxServicos: 4
};
```

8.4.1. Implementação de validação

server.js

```
app.post('/api/reservas', async (req, res) => {
    try {
        const novaReserva = req.body;


        if (!XSD_RULES.numeroReserva.test(novaReserva.numeroReserva)) {
            return res.status(400).json({
                sucesso: false,
                erro: 'Validação XSD falhou: numeroReserva deve seguir o padrão RESxxx (ex: RES001).'
            });
        }


        if (!XSD_RULES.unidades.includes(novaReserva.unidade)) {
            return res.status(400).json({
                sucesso: false,
                erro: `Validação XSD falhou: Unidade inválida. Permitido: ${XSD_RULES.unidades.join(', ')}`
            });
        }


        if (novaReserva.hospede && !XSD_RULES.numeroCliente.test(novaReserva.hospede.numeroCliente)) {
            return res.status(400).json({
                sucesso: false,
                erro: 'Validação XSD falhou: numeroCliente deve ter 3 letras e 3 números (ex: CLI123).'
            });
        }


        if (novaReserva.servicosAdicionais && novaReserva.servicosAdicionais.length > XSD_RULES.maxServicos) {
            return res.status(400).json({
                sucesso: false,
                erro: `Validação XSD falhou: Máximo de ${XSD_RULES.maxServicos} serviços adicionais permitidos.`
            });
        }

        const existe = await db.collection('reservas').findOne({ numeroReserva: novaReserva.numeroReserva });
        if (existe) {
            return res.status(409).json({ sucesso: false, erro: 'Reserva já existe com esse número.' });
        }

        const queryColisao = {
            unidade: novaReserva.unidade,
            quarto: novaReserva.quarto,
            $or: [
                {
                    checkIn: { $lt: novaReserva.checkOut },
                    checkOut: { $gt: novaReserva.checkIn }
                }
            ]
        };

        const conflito = await db.collection('reservas').findOne(queryColisao);

        if (conflito) {
            return res.status(409).json({
                sucesso: false,
                erro: `O quarto ${novaReserva.quarto} na unidade ${novaReserva.unidade} já está reservado nesse período
(${conflito.checkIn} até ${conflito.checkOut}).`
            });
        }

        const resultado = await db.collection('reservas').insertOne(novaReserva);
        res.status(201).json({
            sucesso: true,
            mensagem: 'Reserva criada com sucesso!',
            dados: novaReserva
        });
    } catch (error) {
        res.status(500).json({ sucesso: false, erro: error.message });
    }
});
```

Figura 5- Validações XSD no Backend

8.5.Sistema de Deteção de Conflitos

Verificar se o quarto está ocupado no período

server.js

```
const queryColisao = {
            unidade: novaReserva.unidade,
            quarto: novaReserva.quarto,
            $or: [
                {
                    checkIn: { $lt: novaReserva.checkOut },
                    checkOut: { $gt: novaReserva.checkIn }
                }
            ]
        };

        const conflito = await db.collection('reservas').findOne(queryColisao);

        if (conflito) {
            return res.status(409).json({
                sucesso: false,
                erro: `O quarto ${novaReserva.quarto} na unidade ${novaReserva.unidade} já está reservado nesse período
(${conflito.checkIn} até ${conflito.checkOut}).`
            });
        }
```

Figura 6-Verificar se o quarto está ocupado no período

8.6.Tratamento de Erros HTTP

Código	Descrição / Cenário
200	OK - Operação bem sucedida
201	Created - Reserva criada com sucesso
400	Bad Request - Validação XSD falhou
404	Not Found - Reserva não encontrada
409	Conflict - Reserva duplicada ou conflito de quarto
500	Internal Server Error - Erro de base de dados

9. INTERFACE WEB (Frontend)

9.1. Tecnologias Utilizadas

- HTML5 - Estrutura semântica (1674 linhas)
- CSS3 - Estilos e animações (3500+ linhas)
- JavaScript ES6+ - Lógica e fetch API (1400+ linhas)
- Chart.js - Gráficos interativos
- SweetAlert2 - Alertas personalizados
- Font Awesome - Ícones vetoriais
- ·Google Fonts - Inter, Fira Code

9.2.Layout e Design System

Paleta de cores (CSS Variables):

- --bg-primary: #0a0a0f;
- --bg-secondary: rgba(58, 40, 85, 0.95);
- --bg-card: rgba(45, 35, 65, 0.6);
- --text-primary: #fbeaff;
- --text-secondary: #e0d0f0;
- --accent-primary: #a855f7;
- --accent-secondary: #5d3e8f;
- --accent-purple: #d8b4fe;

9.3.Dashboard de Estatísticas

- 4 cards com métricas animadas (countUp animation)
- Indicador de status da API (online/offline)
- Badges dinâmicos com contagem de serviços
- Atualização em tempo real via fetch()

9.4.Gestão de Reservas (CRUD Completo)

CREATE:

- Formulário multi-step com validação
- Seleção de serviços adicionais (máx. 4)
- Preview antes de submeter
- Feedback visual de sucesso

READ:

- Listagem em cards expansíveis
- Filtros por unidade e hóspede
- Detalhes completos ao expandir
- Indicadores visuais de serviços

UPDATE:

- Formulário de edição com dados pré-preenchidos
- Validação igual à criação
- Confirmação de alterações

DELETE:

- Confirmação com SweetAlert2
- Animação de remoção
- Atualização automática da lista

9.5.Gráficos Interativos

Todos os Gráficos estão no MongoDB neste link - https://charts.mongodb.com/charts-xml-czajzfn/public/dashboards/69514d9b-3646-46d5-819f-a8696c02ee7a , os outros gráficos encontram se no website do projeto.

Abaixo, tem um resumo dos gráficos mais importantes.

9.5.1. GRÁFICO 1: RESERVAS POR UNIDADE (Bar Chart Vertical)

Dados atuais do sistema (8 reservas totais):

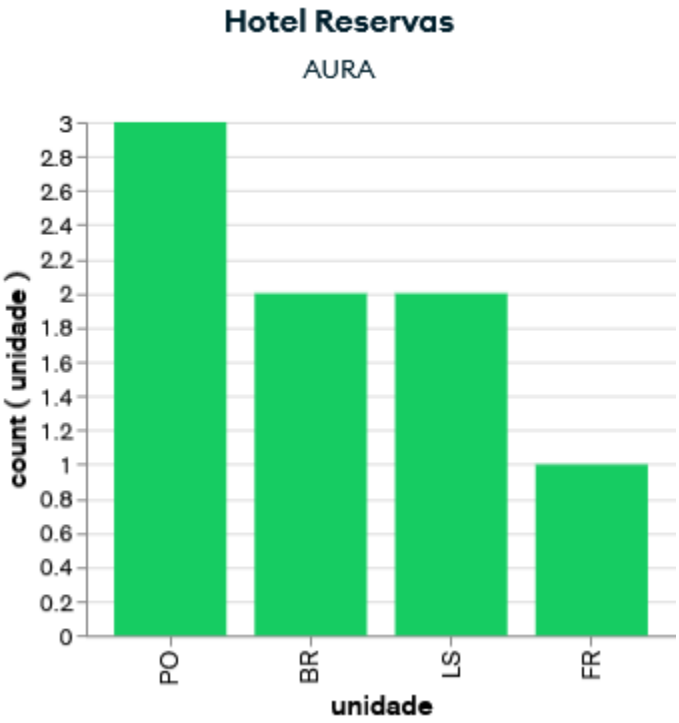


Figura 7-Reservas por Unidade

9.5.2. GRÁFICO 2: SERVIÇOS POR TIPO (Horizontal Bar Chart)

Dados atuais do sistema (30 serviços vendidos):

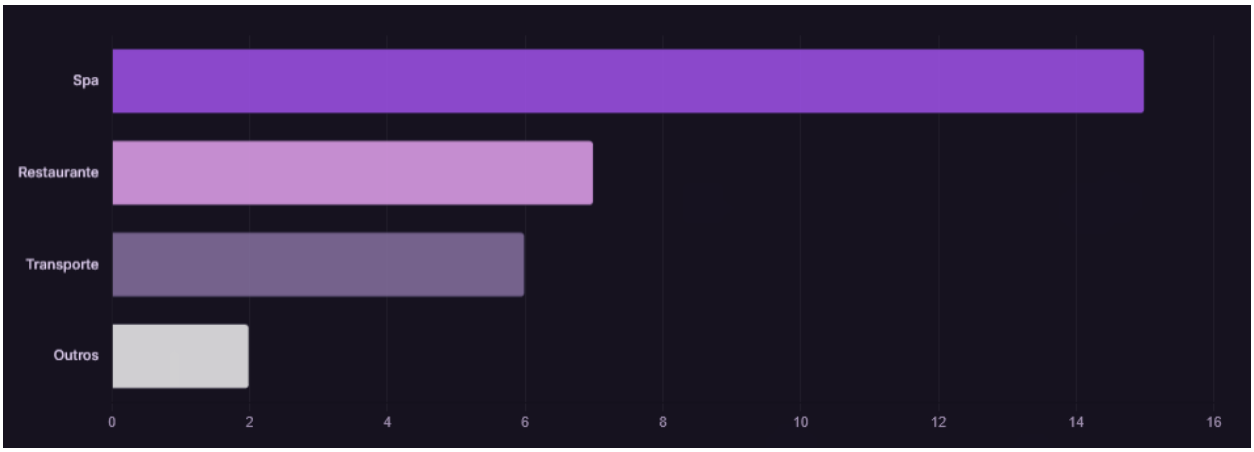


Figura 8-Serviços por Tipo

Total de Serviços Vendidos: 30 unidades

9.5.3. ESTATÍSTICAS DO DASHBOARD

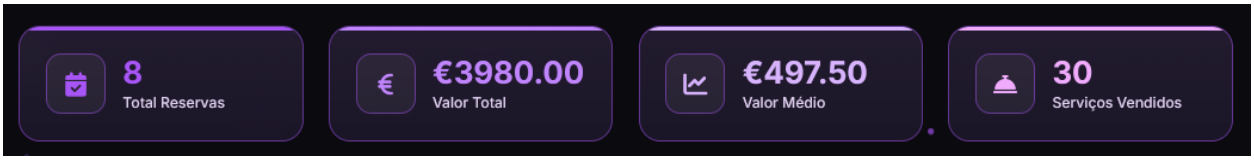


Figura 9-Estastisticas do Dashboard

9.6.CÓDIGO JAVASCRIPT DOS GRÁFICOS

9.6.1. Gráfico de Reservas por Unidade

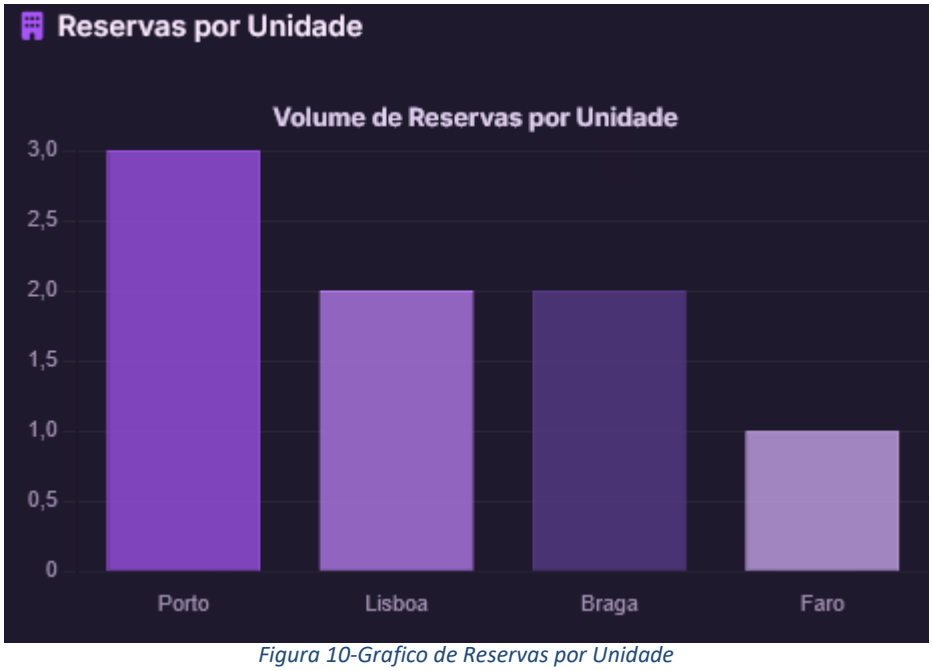


Figura 10-Grafico de Reservas por Unidade

9.6.2. Gráfico de Serviços por Tipo

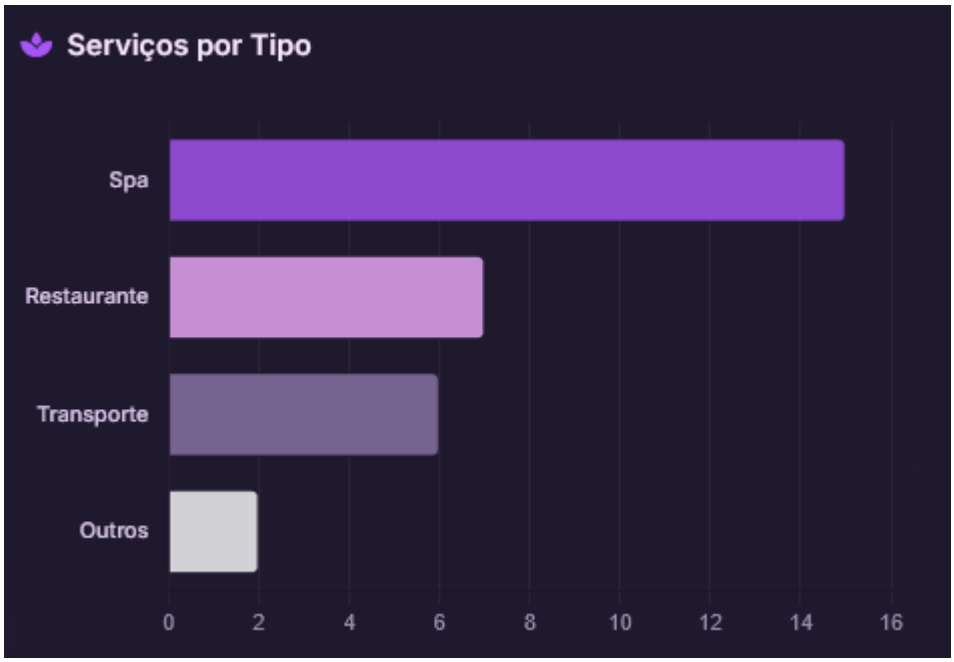


Figura 11-Serviços por Tipo

9.7. Efeitos Visuais e Animações

- Partículas de fundo (Canvas SVG)
- Efeito neve animado (CSS animations)
- Glass morphism nos cards
- Hover effects nos botões
- Transições suaves (0.3s ease)

9.8. Design Responsivo

- Validação em tempo real
- Feedback visual (bordas verdes/vermelhas)
- Mensagens de erro descritivas
- Prevenção de submissão inválida

10. DOCUMENTAÇÃO DA API (Swagger/OpenAPI)

Interface interativa disponível em: https://xml-eight.vercel.app/swagger.html

Características:

- Especificação OpenAPI 3.0
- Descrição de todos os endpoints
- Schemas de request/response
- Exemplos de utilização
- Testes interativos na própria página

11. DEPLOYMENT E INFRAESTRUTURA

Componente	URL / Serviço
Frontend	https://xml-eight.vercel.app
Backend API	https://xml-a96l.onrender.com
Swagger Docs	https://xml-eight.vercel.app/swagger.html
MongoDB Charts	https://charts.mongodb.com/charts-xml-czajzfn/public/dashboards/69514d9b-3646-46d5-819f-a8696c02ee7a
MongoDB Atlas	cluster0.xxxxx.mongodb.net
GitHub (Local)	https://github.com/DiogoCosta5667/XML
GitHub (Vercel)	https://github.com/ArpaoCeleste/xml
Uptime Robot	https://uptimerobot.com/

11.1. Variáveis de Ambiente (.env):

MONGODB_URI=mongodb+srv://user:pass@cluster.mongodb.net/hotel

PORT=3000

12. TESTES E VALIDAÇÃO

Testes Realizados:

- Validação XSD do ficheiro reservas.xml
- Execução de 3 queries XQuery no BaseX
- Testes de todos os 11 endpoints via Postman
- Verificação de CRUD completo
- Testes de responsividade (Chrome DevTools)
- Validação de gráficos com dados reais
- Testes de conflito de reservas

13. DIFICULDADES E SOLUÇÕES

13.1. Dificuldades

- Configuração MongoDB Atlas
- CORS entre Vercel e Render
- Sintaxe Aggregation vs XQuery
- Responsividade Mobile Charts
- Render spin-down (free tier)
- Deploy de múltiplos repôs

13.2. Solução:

- Configuração correta de Network Access
- Middleware cors() com configuração adequada
- Documentação MongoDB e exemplos práticos
- CSS media queries + Chart.js responsive
- UptimeRobot para manter serviço ativo
- Configuração de remotes Git separados

14. Conclusão

O projeto demonstrou com sucesso a integração de múltiplas tecnologias para criar uma solução completa de gestão de reservas hoteleiras.

Objetivos Alcançados

- Esquema XSD com 7 tipos personalizados e validações
- 3 consultas XQuery + módulo RESTXQ completo
- 5 consultas MongoDB Aggregation funcionais
- API REST com 11 endpoints operacionais
- Interface web responsiva com gráficos
- Deploy em ambiente cloud (Vercel + Render)
- Documentação Swagger completa

Métricas do Projeto

- Linhas de código: ~6500+ (HTML + CSS + JS + Node.js)
- Endpoints API: 11
- Queries de agregação: 5
- Consultas XQuery: 3
- Tipos XSD personalizados: 7

15. Referências

15.1. Documentação Oficial

- Node.js - https://nodejs.org/en/docs/
- Express.js - https://expressjs.com/en/guide/routing.html
- MongoDB Manual - https://www.mongodb.com/docs/manual/
- W3C XML Schema - https://www.w3.org/TR/xmlschema-1/
- XQuery 3.1 - https://www.w3.org/TR/xquery-31/
- BaseX Documentation - https://docs.basex.org/

15.2. Bibliotecas Frontend

- Chart.js - https://www.chartjs.org/docs/
- SweetAlert2 - https://sweetalert2.github.io/
- Font Awesome - https://fontawesome.com/docs

15.3. Hosting e Deploy

- Vercel - https://vercel.com/docs
- Render - https://render.com/docs
- MongoDB Atlas - https://www.mongodb.com/docs/atlas/

15.4. Ferramentas

- VS Code - Editor de código
- Postman - Testes de API
- Chrome DevTools - Debug e responsividade
- Git/GitHub - Controlo de versões
- Google AntiGravity - Editor de código