Projeto de Banco de Dados NoSQL MongoDB

Sistema de Autenticação Odontoprev

1. Documento de Projeto

1.1 Descrição do Projeto

O projeto consiste em um sistema de autenticação e autorização para a plataforma Odontoprev, implementado utilizando ASP.NET Core Web API como backend e MongoDB como banco de dados NoSQL. O sistema gerencia usuários com diferentes níveis de acesso (Admin e User), implementando autenticação JWT (JSON Web Tokens) para segurança das operações.

1.2 Justificativa para Escolha do MongoDB

A escolha do MongoDB como banco de dados NoSQL para este projeto baseia-se em várias vantagens técnicas e estratégicas:

Flexibilidade de Schema:

- Permite evolução natural dos modelos de dados sem necessidade de migrations complexas
- Facilita a adição de novos campos aos documentos de usuários sem impactar registros existentes
- Suporte nativo a estruturas de dados hierárquicas e aninhadas

Performance:

- Consultas rápidas através de índices otimizados
- Operações de leitura e escrita eficientes para dados de autenticação
- Cache interno que melhora a performance de consultas frequentes

Escalabilidade:

- Sharding horizontal nativo para crescimento de dados
- Replica sets para alta disponibilidade
- Suporte a clusters distribuídos

Integração com .NET:

- Driver oficial MongoDB.Driver para C#
- Serialização automática de objetos .NET para documentos BSON
- Suporte nativo a operações assíncronas

Adequação ao Domínio:

- Dados de usuários são naturalmente documentais

- Diferentes tipos de usuários podem ter atributos específicos
- Histórico de login e metadados podem ser facilmente expandidos

2. Modelo de Dados e Justificativa

2.1 Estrutura do Documento User

```
{
  "_id": ObjectId,
  "Username": String,
  "Email": String,
  "CPF": String,
  "PasswordHash": Binary,
  "PasswordSalt": Binary,
  "Phone": String,
  "Role": String,
  "CreatedAt": ISODate,
  "LastLogin": ISODate,
  "Active": Boolean
}
```

2.2 Justificativa do Modelo

Campos:

- _id: Identificador único automático do MongoDB
- Username: Nome de exibição do usuário
- Email: Identificador único para login
- CPF: Documento brasileiro obrigatório
- PasswordHash/PasswordSalt: Segurança de senhas com hashing + salt
- Role: Controle de acesso baseado em papéis (RBAC)
- Active: Controle de status
- CreatedAt: Auditoria de criação
- LastLogin: Controle de atividade
- Phone: Contato adicional

2.3 Vantagens do Modelo NoSQL

1. **Flexibilidade**: Novos campos podem ser adicionados sem afetar documentos existentes
2. **Performance**: Todos os dados do usuário em um documento, evitando JOINs
3. **Desnormalização Controlada**: Dados relacionados ao usuário ficam próximos
4. **Evolução**: Schema pode evoluir naturalmente com os requisitos

3. Construção de Dados e Operações

3.1 Documentos JSON/BSON Criados

Os documentos de usuários criados estão presentes anexados nesta pasta: “OdontoprevAuth.Users.json”

Cada documento contém 10 atributos (11 se considerar o _id).

3.2 Operações CRUD Implementadas

Create (Registro):

```
public async Task<User> CreateAsync(User user)
{
   await _users.InsertOneAsync(user);
   return user;
}
```

Read (Consultas):

```
public async Task<List<User>> GetAllAsync() =>
   await _users.Find(_ => true).ToListAsync();

public async Task<User?> GetByIdAsync(string id) =>
   await _users.Find<User>(user => user.Id == id).FirstOrDefaultAsync();

public async Task<User?> GetByEmailAsync(string email) =>
   await _users.Find<User>(user => user.Email == email).FirstOrDefaultAsync();
```

Update:

```
public async Task UpdateAsync(string id, User userIn) =>
   await _users.ReplaceOneAsync(user => user.Id == id, userIn);
```

Delete:

```
public async Task RemoveAsync(string id) =>
   await _users.DeleteOneAsync(user => user.Id == id);
```

4. Interface de Consulta de Dados

4.1 Endpoints da API

Autenticação:

- POST /api/auth/register - Registro de novos usuários
- POST /api/auth/login - Login com JWT

Usuários:

- GET /api/users/me - Perfil do usuário atual
- GET /api/users - Listar todos (Admin)
- GET /api/users/{id} - Usuário específico (Admin)
- PUT /api/users/{id} - Atualizar usuário (Admin)
- DELETE /api/users/{id} - Remover usuário (Admin)

4.2 Funcionalidades Implementadas

1. **Autenticação JWT**: Tokens seguros com expiração configurável

2. **Autorização RBAC**: Controle baseado em papéis (Admin/User)
3. **Validação de Dados**: DTOs para entrada e saída de dados
4. **Hashing de Senhas**: Implementação segura com salt
5. **CORS**: Configurado para integração frontend
6. **Swagger**: Documentação automática da API

4.3 Observações

Você pode obter mais informações sobre a API acessando o repositório no Github, *link anexado nesse arquivo*.

5. Análise de Performance e Escalabilidade

5.1 Performance Atual

Operações de Leitura:

- Consultas por ID: O(1) com índice automático em _id
- Consultas por email: O(log n) com índice recomendado
- Listagem de usuários: O(n) - adequado para volumes moderados

Operações de Escrita:

- Inserção: O(1) para documentos individuais
- Atualização: O(1) com índice por ID
- Exclusão: O(1) com índice por ID

5.2 Otimizações Recomendadas

Índices Sugeridos:

```
// Índice único para email (login)
db.Users.createIndex({ "Email": 1 }, { unique: true })

// Índice para consultas por CPF
db.Users.createIndex({ "CPF": 1 })

// Índice composto para consultas filtradas
db.Users.createIndex({ "Active": 1, "Role": 1 })

// Índice para auditoria
db.Users.createIndex({ "CreatedAt": -1 })
```

5.3 Estratégias de Escalabilidade

Escala Vertical:

- Aumento de CPU e RAM para melhor performance
- Storage SSD para operações I/O mais rápidas

Escala Horizontal:

- **Replica Sets**: Para alta disponibilidade e distribuição de leitura
- **Sharding**: Distribuição de dados por múltiplos servidores
- **Sharding Key Sugerida**: Hash do email ou ID para distribuição uniforme

Configuração de Sharding:

```
// Habilitar sharding na coleção
sh.enableSharding("OdontoprevAuth")
sh.shardCollection("OdontoprevAuth.Users", { "_id": "hashed" })
```

5.4 Métricas de Performance Esperadas

- **Latência**: < 10ms para consultas com índice
- **Throughput**: 1000+ operações/segundo em hardware padrão
- **Concorrência**: Suporte a 100+ conexões simultâneas

6. Segurança e Integração

6.1 Medidas de Segurança Implementadas

Autenticação:

- Hashing de senhas com salt usando algoritmos seguros
- JWT com assinatura HMAC256
- Tokens com expiração configurável (600 minutos padrão)

Autorização:

- Role-Based Access Control (RBAC)
- Proteção de endpoints por decorators [Authorize]
- Validação de claims no token JWT

Configuração JWT Segura:

```
TokenValidationParameters = new TokenValidationParameters
{
   ValidateIssuerSigningKey = true,
   IssuerSigningKey = new SymmetricSecurityKey(key),
   ValidateIssuer = true,
   ValidIssuer = "Auth.API",
   ValidateAudience = true,
   ValidAudience = "OdontoprevClients",
   ValidateLifetime = true,
   ClockSkew = TimeSpan.Zero
};
```

6.2 Segurança do MongoDB

Recomendações de Segurança:

1. **Autenticação do Banco:**

```
// Criar usuário administrativo
use admin
db.createUser({
  user: "odontoprevAdmin",
  pwd: "senhaSegura123!",
  roles: ["userAdminAnyDatabase", "readWriteAnyDatabase"]
})
```

2. **Autorização por Banco:**

```
// Criar usuário específico da aplicação
use OdontoprevAuth
db.createUser({
  user: "apiUser",
  pwd: "senhaApp456!",
  roles: ["readWrite"]
})
```

3. **Configurações de Segurança:**

- Habilitar autenticação: --auth
- SSL/TLS para conexões: --sslMode requireSSL
- Bind IP específico: --bind_ip 127.0.0.1,<IP_INTERNO>
- Auditoria: --auditDestination file

4. **Connection String Segura:**

```
mongodb://apiUser:senhaApp456!@localhost:27017/OdontoprevAuth?authSource=OdontoprevAuth&ssl=tru
e
```

6.3 Integração com Outras Aplicações

APIs RESTful:

- Endpoints padronizados seguindo convenções REST
- Documentação automática via Swagger/OpenAPI
- Suporte a CORS para aplicações web

Microserviços:

```
// Configuração para integração entre serviços
services.AddHttpClient<IOdontoprevService>(client =>
{
   client.BaseAddress = new Uri("https://api.odontoprev.internal/");
   client.DefaultRequestHeaders.Add("Authorization", $"Bearer {token}");
});
```

Frontend Integration:

```
// Exemplo de integração JavaScript
const apiClient = {
   baseURL: 'https://auth-api.odontoprev.com.br/api',
```

```
  async login(credentials) {
    const response = await fetch(`${this.baseURL}/auth/login`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(credentials)
    });
    return await response.json();
  },

  async getProfile(token) {
    const response = await fetch(`${this.baseURL}/users/me`, {
      headers: {
        'Authorization': `Bearer ${token}`
      }
    });
    return await response.json();
  }
};
```

Integração com Gateway API:

```
# Configuração Kong/Ocelot
routes:
 - name: auth-service
   service: odontoprev-auth
   match:
    path: /api/auth/*
   plugins:
    - rate-limiting
    - cors
    - jwt-auth
```

6.4 Monitoramento e Logs

MongoDB Monitoring:

- MongoDB Compass para interface gráfica
- Profiler para análise de queries lentas
- Métricas de performance via MongoDB Atlas

Application Monitoring:

```
// Exemplo de logging estruturado
services.AddLogging(builder =>
{
  builder.AddConsole()
       .AddFile("logs/odontoprev-auth-{Date}.txt")
       .AddStructuredLogging();
});
```

7. Considerações Finais

O projeto demonstra uma implementação robusta de sistema de autenticação usando MongoDB como banco NoSQL. A arquitetura escolhida oferece:

- **Flexibilidade** para evolução dos requisitos
- **Performance** adequada para cargas de trabalho esperadas
- **Segurança** seguindo melhores práticas da indústria
- **Escalabilidade** para crescimento futuro
- **Integração** facilitada com outras aplicações

O uso do MongoDB se justifica pela natureza documental dos dados de usuários, facilidade de implementação com .NET Core e capacidades de escalabilidade horizontal que atendem às necessidades de crescimento da plataforma Odontoprev.

8. Próximos Passos

1. **Implementar cache Redis** para sessões ativas
2. **Adicionar auditoria completa** de operações
3. **Implementar backup automatizado**
4. **Configurar monitoramento proativo**
5. **Adicionar testes de carga** e performance
6. **Implementar CI/CD** para deployments automatizados

9. Scripts Referentes ao MongoDB

9.1 CREATE(Insert)

```
OdontoprevAuth> db.Users.insertOne({
                "Username": "João Silva",
                "Email": "joao.silva@example.com",
                "CPF": "123.456.789-00",
                "PasswordHash": BinData(0, "7TYdnYSLaj3mD95TGnEQ7k/TNxCql80lLkzs0z0Ke3kIjsl7oQm6Uhw8Z+3x3N8AWTIIosyPGv+Eq57ar8o0eg=="),
                "PasswordSalt": BinData(0, "POy+eIpIuhhBvuJ++LRzD2Py6vyhK5NExtTibTylDJ2t3n5hCqjKAtNBC6m0XOpiohGphpsNJ2KAVHJqEJ4N8QWT7unyKLYBOZU7fWxg0EQV3nYezsRd6fbjqkOw
                "Phone": "(11) 98765-4321",
                "Role": "Admin",
                "CreatedAt": new Date(),
                "LastLogin": null,
                "Active": true
            });
```

9.2 READ(Find)

```
OdontoprevAuth> db.Users.findOne({"_id": ObjectId("682e634cbe596b12180b9834")});
OdontoprevAuth> db.Users.findOne({"Email": "joao.silva@example.com"});
OdontoprevAuth> db.Users.findOne({"CPF": "123.456.789-00"});
OdontoprevAuth> db.Users.find({"Role": "Admin"});
```

9.3 UPDATE(Update)

```
OdontoprevAuth> db.users.updateOne(
                  {"_id": ObjectId("682e634cbe596b12180b9834")},
                  {"$set": {"LastLogin": new Date()}}
                );
OdontoprevAuth> db.users.updateOne(
                  {"Email": "joao.silva@example.com"},
                  {"$set": {"Phone": "(11) 99999-9999"}}
                );
OdontoprevAuth> db.users.updateOne(
                  {"CPF": "123.456.789-00"},
                  {"$set": {"Active": false}}
                );
```

9.4 DELETE(Delete)

```
OdontoprevAuth> db.users.deleteOne({"_id": ObjectId("682e634cbe596b12180b9834")});
OdontoprevAuth> db.users.deleteOne({"Email": "joao.silva@example.com");
```

9.5 Índices Únicos

```
OdontoprevAuth> db.users.createIndex({ "Email": 1 }, { unique: true, name: "email_unique_idx"  });
OdontoprevAuth> db.users.createIndex({ "CPF": 1 }, { unique: true, name: "cpf_unique_idx"  });
```

10. Github

Repositório da API de .NET com conexão com o banco:

https://github.com/VitorOnofreRamos/OdontoprevSolutionNET

11. Integrantes

RM553241 – Vitor Onofre Ramos

RM552600 – Beatriz Silva

RM553801 – Pedro Henrique Soares Araujo