Documentação do Microserviço para Monitoramento de Consumo de Energia Elétrica

Marcela Gervasoni Gomes RM94668 / Gabrielly de Paula Freitas RM94645

Introdução

Este projeto consiste na criação de um microserviço em **C# com ASP.NET Core** para monitoramento de consumo de energia elétrica. Ele inclui integração com MongoDB, uso de cache com Redis, implementação de testes unitários com XUnit, e documentação detalhada das rotas e desempenho.

Atividade 1: Código Inicial e API Funcional

Estrutura do Projeto

- **Framework**: ASP.NET Core
- **Estrutura de Pastas**:
- `/Controllers`
- `/Repository`
- `/Main`
- `/Tests`
- `appsettings.json`
- `Program.cs`

Rotas Básicas

1. **GET /health**
    - **Descrição**: Verifica o status do serviço.
    - **Respostas**:
        - `200 OK`: Serviço operacional.
        - `500 Internal Server Error`: Problema interno.
2. **POST /consumo**
    - **Descrição**: Registra dados de consumo.
    - **Body esperado**:
    - ```
      {
        "id": "123",
        "timestamp": "2024-11-22T10:00:00Z",
        "consumo": 120.5
      }
      ```
      ```

- **Respostas**:
  - `201 Created`: Dados salvos com sucesso.
  - `400 Bad Request`: Dados inválidos.

3. **GET /consumo**
   - **Descrição**: Recupera dados de consumo registrados.
   - **Respostas**:
     - `200 OK`: Dados retornados.
     - `404 Not Found`: Nenhum dado encontrado.

Atividade 2: Integração com MongoDB

Configuração do MongoDB

- **Pacote**: `MongoDB.Driver`
- **Configuração**:
- ```
  {
    "MongoSettings": {
      "ConnectionString": "mongodb://localhost:27017",
      "DatabaseName": "ConsumoEnergia"
    }
  }
  ```

Código de Integração

Serviço de Conexão:

```
public class MongoService {
    private readonly IMongoCollection<Consumo> _collection;

    public MongoService(IConfiguration config) {
        var client = new
MongoClient(config["MongoSettings:ConnectionString"]);
        var database =
client.GetDatabase(config["MongoSettings:DatabaseName"]);
        _collection = database.GetCollection<Consumo>("Consumos");
    }

    public async Task InsertAsync(Consumo consumo) => await
_collection.InsertOneAsync(consumo);

    public async Task<List<Consumo>> GetAllAsync() => await
_collection.Find(_ => true).ToListAsync();
}
```

```

Atividade 3: Cache com Redis

Configuração

- **Pacote**: `StackExchange.Redis`
- **Configuração no `appsettings.json`**:
- {
- "RedisSettings": {
- "ConnectionString": "localhost:6379",
- "CacheDurationInSeconds": 300
- }
- }

Implementação de Cache

Serviço de Cache:

```
public class RedisCacheService {
    private readonly IDatabase _cache;

    public RedisCacheService(IConnectionMultiplexer redis) {
        _cache = redis.GetDatabase();
    }

    public async Task<string> GetCachedData(string key) => await
_cache.StringGetAsync(key);

    public async Task SetCachedData(string key, string data, TimeSpan
expiration) {
        await _cache.StringSetAsync(key, data, expiration);
    }
}
```

Adicionando Cache na Rota GET /consumo:

```
[HttpGet("consumo")]
public async Task<IActionResult> GetConsumo([FromServices]
RedisCacheService redisService, [FromServices] MongoService
mongoService) {
    var cacheKey = "consumo_data";
    var cachedData = await redisService.GetCachedData(cacheKey);

    if (!string.IsNullOrEmpty(cachedData)) {
        return
Ok(JsonConvert.DeserializeObject<List<Consumo>>(cachedData));
    }

    var data = await mongoService.GetAllAsync();
    if (data.Count == 0) return NotFound("No data found.");

    await redisService.SetCachedData(cacheKey,
JsonConvert.SerializeObject(data), TimeSpan.FromMinutes(5));
    return Ok(data);
}
```

Atividade 4: Testes Unitários com XUnit

Configuração

- **Pacote**: `XUnit`
- **Estrutura dos Testes**:
    - Inserção no MongoDB.
    - Recuperação de dados do Redis.
    - Respostas de status codes.

Exemplo de Teste:

```
public class ConsumoTests {
    [Fact]
    public async Task InsercaoMongoDB_DeveSalvarDados() {
        var mongoService = new Mock<MongoService>();
        var consumo = new Consumo { Id = "123", Timestamp =
DateTime.Now, Consumo = 120.5 };

        await mongoService.Object.InsertAsync(consumo);

        mongoService.Verify(m => m.InsertAsync(consumo), Times.Once);
    }
}
```

Atividade 5: Documentação e Performance

Performance

- **Ferramenta**: Postman.
- **Resultados**:
  - Sem Cache: Tempo médio ~500ms.
  - Com Cache: Tempo médio ~50ms.

Documentação das Rotas

- **Ferramenta**: Swagger.
- **Exemplo de Rota Documentada (POST /consumo)**:
  - **Request**:
  - ```
    {
      "id": "123",
      "timestamp": "2024-11-22T10:00:00Z",
      "consumo": 120.5
    }
    ```
  - **Response**:
  - ```
    {
      "status": "201 Created",
      "message": "Consumo registrado com sucesso"
    }
    ```
