

QUESTÃO 1 - Modelagem (Embedding x Referencing)

1. Modelagem por Referência, como o número de manutenções por carro pode chegar a centenas, usar a modelagem **Embedding (incorporação)** faria o documento do carro crescer excessivamente

QUESTÃO 2 - Inserção de Dados (CRUD / Aula 3)

use monitoramento-veicular

```
db.leituras.insertMany([
  {
    carro: "GT-R",
    sensor: "temperatura_motor",
    valor: 95.2,
    dataHora: new Date()
  },
  {
    carro: "F40",
    sensor: "pressao_oleo",
    valor: 4.8,
    dataHora: new Date()
  },
  {
    carro: "GT-R",
    sensor: "velocidade",
    valor: 285.5,
    dataHora: new Date()
  }
])
```

```
> MONGOISH
> db.mon1
< test.mon1
< use monitoramento-veicular
< switched to db monitoramento-veicular
> db.leituras.insertMany([
  {
    carro: "GT-R",
    sensor: "temperatura_motor",
    valor: 95.2,
    dataHora: new Date()
  },
  {
    carro: "F40",
    sensor: "pressao_oleo",
    valor: 4.8,
    dataHora: new Date()
  },
  {
    carro: "GT-R",
    sensor: "velocidade",
    valor: 285.5,
    dataHora: new Date()
  }
])
< [
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('692dbd365eda1f5c921209b0'),
    '1': ObjectId('692dbd365eda1f5c921209b1'),
    '2': ObjectId('692dbd365eda1f5c921209b2')
  }
]
monitoramento-veicular >
```

QUESTÃO 3 - Consultas com Operadores Lógicos (Aula 4)

```
db.leituras.find({
  $or: [
    { sensor: "temperatura_motor" },
    { sensor: "pressao_oleo" }
  ],
  valor: { $gt: 90 }
})
```

```
> MONGOSH
> db.leituras.find({
  $or: [
    { sensor: "temperatura_motor" },
    { sensor: "pressao_oleo" }
  ],
  valor: { $gt: 90 }
})
< {
  _id: ObjectId("692dbd365eda1f5c921209be"),
  carro: 'GT-R',
  sensor: 'temperatura_motor',
  valor: 95.2,
  dataHora: 2025-12-01T16:07:18.874Z
}
monitoramento-veicular> |
```

QUESTÃO 4 - Atualização Avançada (Aula 4)

```
db.leituras.updateMany(
  { carro: "GT-R" }, // Filtro: todas as leituras do carro "GT-R"
  {
    $set: { status_sensor: "verificar" }, // Adiciona/Atualiza o campo
    $unset: { codigo_defeito: "" } // Remove o campo
}
```

```
)
```

```
> MONGOSH
> db.leituras.updateMany(
  { carro: "GT-R" }, // Filtro: todas as leituras do carro "GT-R"
  {
    $set: { status_sensor: "verificar" }, // Adiciona/Atualiza o campo
    $unset: { codigo_defeito: "" } // Remove o campo
  }
)
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
monitoramento-veicular>
```

QUESTÃO 5 - Paginação (Aula 4)

```
db.leituras.find(
  { sensor: "velocidade" } // 1. Filtra por sensor
)
.sort(
  { dataHora: -1 } // 2. Ordena pela mais recente (-1 para decrescente)
)
.skip(10) // 3. Ignora (pula) as 10 primeiras leituras
```

.limit(5) // 4. Limita (pega) as 5 próximas

```
> MONGOSH
> db.leituras.find(
  { sensor: "velocidade" }
)
{
  .sort(
    { datahora: -1 }
  )
  .skip(10)
  .limit(5)
<
monitoramento-veicular>|
```

Como, não temos registro desse sensor, o script não vai retornar nada.

QUESTÃO 6 - Agregação (Aula 6)

```
db.leituras.aggregate([
  {
    $match: { sensor: "temperatura_motor" }
  },
  {
    $group: {
      _id: "$carro",
      mediaTemperatura: { $avg: "$valor" }
    }
  },
  {
    $sort: { mediaTemperatura: -1 }
  }
])
```

```
> MONGOSH
> db.leituras.aggregate([
  {
    $match: { sensor: "temperatura_motor" }
  },
  {
    $group: {
      _id: "$carro",
      mediaTemperatura: { $avg: "$valor" }
    }
  },
  {
    $sort: { mediaTemperatura: -1 }
  }
])
< [
  {
    _id: "GT-R",
    mediaTemperatura: 95.2
  }
]
monitoramento-veicular> |
```

QUESTÃO 7 — API Node.js (Aula 7)

```
const express = require('express');

const router = express.Router();

const Leitura = require('../models/Leitura');

// POST /leituras
router.post('/', async (req, res) => {

  try {
    const { carro, sensor, valor } = req.body;

    // 1. Validação explícita (Mongoose faz uma implícita, mas esta é mais rápida e clara)

    if (!carro || !sensor || valor === undefined) {
      return res.status(400).json({ mensagem: 'Erro: Campos obrigatórios (carro, sensor, valor) devem ser fornecidos.' });
    }

  }
```

```

// 2. Insere a leitura

const novaLeitura = new Leitura({ carro, sensor, valor });

await novaLeitura.save();

// 3. Retorna código HTTP 201 (Created)

res.status(201).json(novaLeitura);

}

} catch (error) {

    // Trata erros de validação do Mongoose

    if (error.name === 'ValidationError') {

        return res.status(400).json({ mensagem: 'Erro de validação nos dados.', detalhes: error.message });

    }

    console.error("Erro ao registrar leitura:", error);

    res.status(500).json({ mensagem: 'Erro interno do servidor.' });

}

});

module.exports = router;

```

QUESTÃO 8 — Consumo de API Externa (Aula 7)

```

require('dotenv').config();

const axios = require('axios');

const mongoose = require('mongoose');

const Clima = require('./models/Clima'); // Modelo criado no passo anterior

// --- Configurações da API Externa ---

```

```
const OPEN_WEATHER_API_KEY = process.env.WEATHER_API_KEY || 'API_KEY';

const CIDADE = 'Jacarei, BR';

const API_URL =
`http://api.openweathermap.org/data/2.5/weather?q=${CIDADE}&appid=${OPEN_WEATHER_API_KEY}&units=metric`;

// Conexão com o MongoDB (garante que a função pode ser executada standalone)
mongoose.connect(process.env.MONGODB_URI)

.then(() => console.log('Conexão com MongoDB estabelecida para importação.'))
.catch(err => console.error('Erro de conexão com MongoDB:', err));

/** 
 * Consome a API de clima, extrai a temperatura e salva no MongoDB.
 */

async function importarTemperaturaAmbiente() {
  console.log(`Buscando dados de clima para ${CIDADE}...`);

  try {
    // 1. Consome a API externa
    const response = await axios.get(API_URL);

    // 2. Extrai 'temp'
    const temperatura = response.data.main.temp;
    const localNome = response.data.name;

    console.log(`Temperatura atual em ${localNome}: ${temperatura}°C`);

    // 3. Salva na coleção 'clima'
    const registroClima = new Clima({
```

```

        temperatura: temperatura,
        local: localNome,
        dataHora: new Date()

    });

    await registroClima.save();

    console.log('Registro de clima salvo com sucesso!');

}

} catch (error) {
    console.error('Erro ao importar ou salvar dados de clima:', error.message);
}

} finally{
    // Fecha a conexão após a operação
    mongoose.connection.close();
}

}

// Executa a função
importarTemperaturaAmbiente();

```

QUESTÃO 9 — Segurança no MongoDB (Aula 8)

```

db.createUser(
{
    user: "engenheiroCorrida",
    pwd: "SenhaFortissima123#", // 🚨 Defina uma senha forte!
    roles: [
        { role: "read", db: "telemetria_race" } // Permissão de apenas leitura
    ]
}
)

```

```
> MONGOSH
> db.createUser(
  {
    user: "engenheiroCorrida",
    pwd: "SenhaFortissima123#", // 🔑 Defina uma senha forte!
    roles: [
      { role: "read", db: "telemetria_race" } // Permissão de apenas leitura
    ]
  }
)
< { ok: 1 }
monitoramento-veicular>
```

QUESTÃO 10 — Backup e Restauração (Aula 8)

```
// backup_scheduler.js (Script para automação)
const cron = require('node-cron');
const shell = require('shelljs');

// O caminho de destino fornecido na imagem
const BACKUP_DIR =
'C:\Users\painh\OneDrive\Desktop\revisao_p3\backups\';

// -----
// Agendamento: Roda todos os dias à 01:00 da manhã [cite: 64]
// -----
cron.schedule('0 1 * * *', () => {
  // Gera a string de data (ex: 2025-12-01) para o nome da pasta
  const dateStr = new Date().toISOString().slice(0, 10);
  const outputDir = `${BACKUP_DIR}${dateStr}`;

  console.log(`[${new Date().toLocaleString()}] Iniciando backup diário do
banco 'telemetria_race'...`);

  // Comando mongodump, usando aspas duplas para o caminho do Windows
  const backupCommand = `mongodump --db telemetria_race --out
"${outputDir}"`;

  // Executa o comando
  if (shell.exec(backupCommand).code !== 0) {
    console.error('✖ Erro: Falha na execução do mongodump.');
  }
});
```

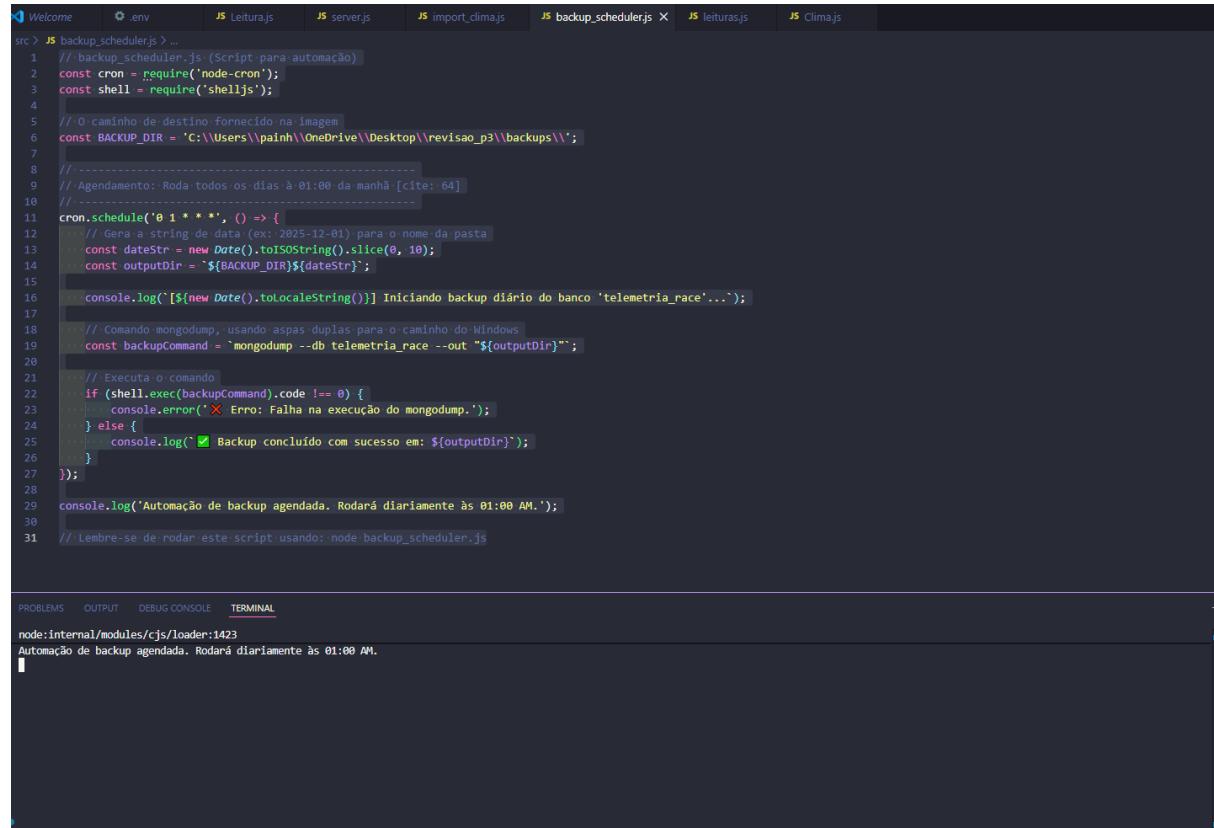
```

    } else {
        console.log(`✅ Backup concluído com sucesso em: ${outputDir}`);
    }
});

console.log('Automação de backup agendada. Rodará diariamente às 01:00 AM.');

// Lembre-se de rodar este script usando: node backup_scheduler.js

```



```

src > JS backup_scheduler.js > ...
1 // backup_scheduler.js (Script para automação)
2 const cron = require('node-cron');
3 const shell = require('shelljs');
4
5 // O caminho de destino fornecido na imagem
6 const BACKUP_DIR = 'C:\\Users\\painh\\OneDrive\\Desktop\\revisao_p3\\backups\\';
7
8 // -----
9 // Agendamento: Roda todos os dias às 01:00 da manhã [cite: 64]
10 // -----
11 cron.schedule('0 1 * * *', () => {
12     // Gera a string de data (ex: 2025-12-01) para o nome da pasta
13     const dateStr = new Date().toISOString().slice(0, 10);
14     const outputDir = `${BACKUP_DIR}${dateStr}`;
15
16     console.log(`[${new Date().toLocaleString()}] Iniciando backup diário do banco 'telemetria_race'...`);
17
18     // Comando mongodump, usando aspas duplas para o caminho do Windows
19     const backupCommand = `mongodump --db telemetria_race --out "${outputDir}"`;
20
21     // Executa o comando
22     if (shell.exec(backupCommand).code !== 0) {
23         console.error(`✖ Erro: Falha na execução do mongodump.`);
24     } else {
25         console.log(`✅ Backup concluído com sucesso em: ${outputDir}`);
26     }
27 });
28
29 console.log('Automação de backup agendada. Rodará diariamente às 01:00 AM.');
30
31 // Lembre-se de rodar este script usando: node backup_scheduler.js

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

node:internal/modules/cjs/loader:1423
Automação de backup agendada. Rodará diariamente às 01:00 AM.