

Atividade 1

Tarefa: Proponha a modelagem em MongoDB que melhor se adequa ao cenário e justifique usando os conceitos das Aulas 1 e 2

A melhor opção é utilizar o padrão **Embedding**, incorporando o *array* de manutenções dentro do documento principal do carro.

- **Justificativa:** O desempenho é a prioridade. O Embedding permite recuperar o carro e **todo** o seu histórico de manutenção em **uma única consulta** (sem a necessidade de *joins*), o que é ideal para **análises rápidas**. Embora o número de manutenções seja grande, o volume de dados de cada manutenção (data, tipo, mecânico) é pequeno o suficiente para caber dentro do limite de 16MB do documento MongoDB.

QUESTÃO 2 — Inserção de Dados (CRUD / Aula 3) Insira 3 registros de telemetria na coleção leituras contendo: carro, sensor, valor e data/hora.

```
>_MONGOSH
> db.leituras.insertMany([
  {
    "carro": "GT-R",
    "sensor": "temperatura_motor",
    "valor": 95.2,
    "data_hora": ISODate("2025-11-17T18:00:00Z")
  },
  {
    "carro": "F40",
    "sensor": "pressao_oleo",
    "valor": 7.5,
    "data_hora": ISODate("2025-11-17T18:01:00Z")
  },
  {
    "carro": "GT-R",
    "sensor": "velocidade",
    "valor": 280.5,
    "data_hora": ISODate("2025-11-17T18:02:00Z")
  }
]);
```

```

    acknowledged: true,
    insertedIds: [
      '0': ObjectId('691bab0f8f9c5efab969a680'),
      '1': ObjectId('691bab0f8f9c5efab969a681'),
      '2': ObjectId('691bab0f8f9c5efab969a682')
    ]
}

```

QUESTÃO 3 — Consultas com Operadores Lógicos (Aula 4) Liste todas as leituras onde:

- o sensor seja “temperatura_motor” OU “pressao_oleo”,
- E o valor seja maior que 90.

```

> db.leituras.find({
  $or: [
    { "sensor": "temperatura_motor" },
    { "sensor": "pressao_oleo" }
  ],
  "valor": { $gt: 90 }
});
< [
  {
    _id: ObjectId('691bab0f8f9c5efab969a680'),
    carro: 'GT-R',
    sensor: 'temperatura_motor',
    valor: 95.2,
    data_hora: 2025-11-17T18:00:00.000Z
  }
]

```

QUESTÃO 4 — Atualização Avançada (Aula 4) O carro “GT-R” teve uma falha no sensor de temperatura. Atualize todas as leituras desse carro adicionando o campo: "status_sensor": "verificar" E remova o campo codigo_defeito, caso exista.

```
> db.leituras.updateMany(  
  { "carro": "GT-R" }, // Filtro  
  {  
    $set: { "status_sensor": "verificar" },  
    $unset: { "codigo_defeito": "" }  
  }  
);  
< {  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 2,  
  modifiedCount: 2,  
  upsertedCount: 0  
}
```

QUESTÃO 5 — Paginação (Aula 4)

Liste as 5 leituras mais recentes do sensor

“velocidade”, ignorando as primeiras 10.

```
> db.leituras.updateMany(  
  { "carro": "GT-R" }, // Filtro  
  {  
    $set: { "status_sensor": "verificar" },  
    $unset: { "codigo_defeito": "" }  
  }  
);  
< {  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 2,  
  modifiedCount: 2,  
  upsertedCount: 0  
}  
> db.leituras.find({  
  "sensor": "velocidade"  
})  
  .sort({ "data_hora": -1 })  
  .skip(10)  
  .limit(5);
```

QUESTÃO 6 — Agregação (Aula 6)

Calcule a média de temperatura do motor por carro,

ordenando os maiores valores primeiro.

```

    }
    > db.leituras.find({
      "sensor": "velocidade"
    })
    .sort({ "data_hora": -1 })
    .skip(10)
    .limit(5);
<
> db.leituras.aggregate([
  {

    $match: { "sensor": "temperatura_motor" }

  },
  {

    $group: {
      "_id": "$carro",
      "media_temperatura": { $avg: "$valor" }
    }
  },
  {

    $sort: { "media_temperatura": -1 }

  }
]);
< {
  _id: 'GT-R',
  media_temperatura: 95.2
}

```

QUESTÃO 7 — API Node.js (Aula 7)

A equipe precisa registrar uma nova leitura via API

REST.

Crie a rota POST /leituras usando Express + Mongoose

que:

1. Valida se os campos obrigatórios foram

enviados

2. Insere a leitura

3. Retorna código HTTP 201

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under 'SISTEMA-TELEMETRIA-RACE'. It includes 'models' (Carro.js, Clima.js, Leitura.js), 'routes' (leituras.js), and 'utils' (backupScheduler.js, climalporter.js).
- Code Editor:** The file 'leituras.js' is open. The code is as follows:

```

routes > JS leituras.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const Leitura = require('../models/Leitura');
4
5  router.post('/leituras', async (req, res) => {
6    try {
7      const novaLeitura = new Leitura(req.body);
8
9      // 1. A validação ocorre implicitamente quando .save() é chamado.
10     const leituraSalva = await novaLeitura.save(); // 2. Insere a leitura
11
12     // 3. Retorna código HTTP 201 (Created)
13     res.status(201).json(leituraSalva);
14
15   } catch (error) {
16     if (error.name === 'ValidationError') {
17       // Se a validação do Mongoose falhar (campo obrigatório faltando)
18       // return res.status(400).json({ erro: "Dados de leitura incompletos ou inválidos." });
19     }
20     res.status(500).json({ erro: "Erro interno do servidor." });
21   }
22 });
23 module.exports = router;

```

QUESTÃO 8 — Consumo de API Externa (Aula 7)

A equipe deseja importar automaticamente a temperatura ambiente antes das corridas.

Crie uma função Node.js usando axios que:

- consome a API externa,
 - extrai temp,
 - salva na coleção clima.

The screenshot shows a browser-based debugger interface. At the top, there's a navigation bar with icons for search, file, settings, and help. Below the bar, the code editor displays a file named `climaImporter.js` with the following content:

```
utils > JS climaImporter.js [⑥] API KEY
1 // utils/climaImporter.js
2 const axios = require('axios');
3 const Clima = require('../models/Clima');
4
5 const API_KEY = '74d94efa090fe8563410f94c81d3ba'; // << SUBSTITUIR
6 const API_URL = 'https://api.openweathermap.org/data/2.5/weather';
7 const CITY = 'Jacareí,br';
8
9 async function importarTemperaturaAmbiente() {
10   try {
11     // 1. Consome a API externa usando os parâmetros
12     const response = await axios.get(API_URL, {
13       params: {
14         q: CITY,
15         appid: API_KEY,
16         units: 'metric' // Para extrair em Celsius
17       }
18     });
19   }
20 }
```

Below the code editor, there's a toolbar with buttons for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is selected, showing the message "Debugger attached." followed by the command `> sistema-telemetria-race@1.0.0 start` and the file `> node server.js`.

At the bottom, there's a status bar with the message "Agendamento de backup diário ativado (0, 10).". It also shows a green checkmark next to "Conectado ao MongoDB!" and another green checkmark next to "Temperatura salva: 21.4°C".

server.js

```
9  const DB_URI = 'mongodb://localhost:27017/telemetria_race';
10
11 // Conexão com o MongoDB
12 mongoose.connect(DB_URI)
13   .then(() => {
14     console.log('✅ Conectado ao MongoDB!');
15
16     // 🚨 CORREÇÃO: Chamar a Q. 8 aqui para que ela execute automaticamente
17     importarTemperaturaAmbiente();
18   })
19   .catch(err => console.error('❌ Erro de conexão:', err));
20
21 // Rota de Teste para Q. 8 (Pode manter, mas não é a solução do problema)
22 app.get('/importar-clima', async (req, res) => {
23   try {
24     await importarTemperaturaAmbiente();
25     res.status(200).send("Importação de clima iniciada e concluída. Verifique o console/DB.");
26   } catch (e) {
27     res.status(500).send("Erro na importação de clima.");
28   }
29 });
30 // Middlewares (Mantenha antes das rotas, mas depois das chamadas de API)
31 app.use(express.json()); // Habilita o Express a processar JSON
32
33 // Rotas
34 app.use(leituraRoutes); // Usa a rota POST /leituras (Q. 7)
35
36
37 // Automação (Q. 10)
38 backupScheduler.start();
39
40 // Inicia o servidor
41 app.listen(PORT, () => {
  });

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS


```
> sistema-telemetria-race@1.0.0 start
> node server.js

Debugger attached.
Agendamento de backup diário ativado (Q. 10).
Servidor Node.js rodando na porta 3000
✅ Conectado ao MongoDB!
✅ Temperatura salva: 21.4°C.
```

```
>_MONGOSH
> use telemetria_race
< switched to db telemetria_race
> db["clima"].find()
< [
  {
    _id: ObjectId('691bb75211b3e5888751d13b'),
    temperatura: 21.4,
    descricao: 'broken clouds',
    data_importacao: 2025-11-18T00:01:22.875Z,
    __v: 0
  }
]
telemetria_race> |
```

QUESTÃO 9 — Segurança no MongoDB (Aula 8)

Crie um usuário chamado engenheiroCorrida com acesso apenas de leitura ao banco telemetria_race.

```
> use admin

db.createUser({
  user: "engenheiroCorrida",
  pwd: "SENHA_SEGURA_DO_ENGENHEIRO",
  roles: [
    { role: "read", db: "telemetria_race" }
  ]
});
< switched to db admin
admin>
```

QUESTÃO 10 — Backup e Restauração (Aula 8)

A equipe quer fazer backup diário dos dados de telemetria.

Crie o comando de backup e um exemplo de automação com node-cron.

baixar a **Ferramenta:** Use mongodump.

Rodar

```
mongodump --db="telemetria_race" --out="/caminho/para/backups/diarios/telemetria_$(date +%Y%m%d)"
```

```

JS climImporter.js JS backupScheduler.js JS server.js

utils > JS backupScheduler.js ...
1 // utils/backupScheduler.js
2 const cron = require('node-cron');
3 const { exec } = require('child_process');
4
5 // 1. Comando de Backup (do Passo 1)
6 const BACKUP_COMMAND = 'mongodump --db=telemetria_race --out=/caminho/para/backups/diarios/telemetria_${date +%Y%m%d}';
7
8 function startBackupScheduler() {
9   console.log('Agendamento de backup diário ativado (Q. 10).');
10
11 // 2. Agendar a execução para 01:00 AM
12 cron.schedule('0 1 * * *', () => {
13   console.log(`Executando backup diário da telemetria...`);
14
15   // 3. Executar o comando no Shell
16   exec(BACKUP_COMMAND, (error) => {
17     if (error) {
18       console.error(`X Erro no backup: ${error.message}`);
19       return;
20     }
21     console.log(`✓ Backup concluído com sucesso.`);
22   });
23 }, {
24   timezone: "America/Sao_Paulo" // Garante a hora correta
25 });
26
27 module.exports = { start: startBackupScheduler };

```

```

EXPLORER
SISTEMA-TELEMETRIA-RACE
  models
    JS Carro.js
    JS Clima.js
    JS Leitura.js
  node_modules
  routes
    JS leituras.js
  utils
    JS backupScheduler.js
    JS climImporter.js
    package-lock.json
    package.json
    README.txt
  server.js

JS climImporter.js JS backupScheduler.js JS server.js

JS server.js > ...
20
21 // Rota de Teste para Q. 8 (Pode manter, mas não é a solução do problema)
22 app.get('/importar-clima', async (req, res) => {
23   try {
24     await importarTemperaturaAmbiente();
25     res.status(200).send("Importação de clima iniciada e concluída. Verifique o console/DB.");
26   } catch (e) {
27     res.status(500).send("Erro na importação de clima.");
28   }
29 });
30 // Middlewares (Mantenha antes das rotas, mas depois das chamadas de API)
31 app.use(express.json()); // Habilita o Express a processar JSON
32
33 // Rotas
34 app.use(leituraRoutes); // Usa a rota POST /leituras (Q. 7)
35
36
37 // Automação (Q. 10)
38 backupScheduler.start();
39
40 // Inicia o servidor
41 app.listen(PORT, () => {
42   console.log(`Servidor Node.js rodando na porta ${PORT}`);
43 });

```

Ao iniciar o servidor com `npm run dev`, a mensagem **"Agendamento de backup diário ativado (Q. 10)."** aparecerá no console, confirmando que a rotina de automação está pronta para ser executada à 01:00 AM todos os dias.