


API Livraria com Express.js — Parte 3 (Camada Repository)

Evoluindo o projeto: separando persistência da lógica de negócio

 Professor: Fabricio Bizotto

 Disciplina: Desenvolvimento Web I

 Curso: Ciência da Computação

 Fase: 4ª fase

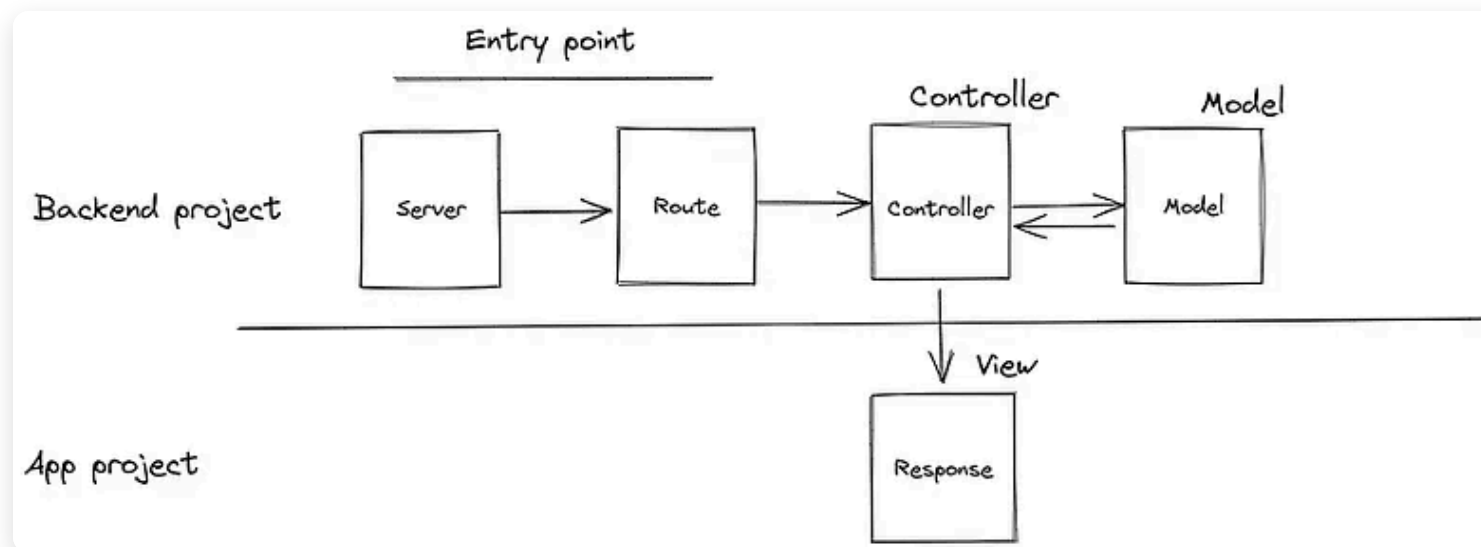
Roteiro

- Introdução à camada **Repository**
- Criar `RepositoryBase` (interface genérica)
- Criar `LivrosRepository` (implementação JSON)
- Refatorar `LivrosController` para usar o repositório
- Testar as rotas e entender a nova arquitetura

Estrutura atualizada do projeto

```
livraria_node_http/  
├── server.js  
├── .env  
├── package.json  
├── src/  
│   ├── app.js  
│   ├── config/  
│   │   └── express.js  
│   ├── data/  
│   │   └── livros.json  
│   ├── repositories/  
│   │   ├── repository.interface.js  
│   │   └── livros.repository.js  
│   ├── controllers/  
│   │   └── livros.controller.js  
│   ├── routes/  
│   │   ├── index.js  
│   │   └── livros.routes.js  
│   └── middlewares/  
│       ├── errorHandler.js  
│       └── validar/  
│           └── livros.validar.js
```

Arquitetura: com camada Repository



A lógica de **acesso a dados** é isolada em uma camada independente.
O **Controller** agora apenas **coordena** a requisição e chama o **Repository**.

src/repositories/repository.interface.js

```
class RepositoryBase {  
  
  constructor() {  
    if (this.constructor === RepositoryBase) {  
      throw new Error("Não é possível instanciar uma classe abstrata");  
    }  
    this.caminhoArquivo = null;  
  }  
  
  async findAll() { throw new Error("Método deve ser implementado"); }  
  async findById(id) { throw new Error("Método deve ser implementado"); }  
  async create(data) { throw new Error("Método deve ser implementado"); }  
  async update(id, data) { throw new Error("Método deve ser implementado"); }  
  async delete(id) { throw new Error("Método deve ser implementado"); }  
  
  async getNextId() {  
    const items = await this.findAll();  
    if (items.length === 0) return 1;  
    return Math.max(...items.map(item => item.id)) + 1;  
  }  
}  
  
module.exports = RepositoryBase;
```

src/repositories/livros.repository.js

```
const fs = require("fs");
const path = require("path");
const RepositoryBase = require("../repository.interface");

class LivrosRepository extends RepositoryBase {
  constructor() {
    super();
    this.caminhoArquivo = path.join(__dirname, "../data/livros.json");
  }

  async findAll() {
    const dados = await this._lerArquivo();
    return JSON.parse(dados);
  }

  async findById(id) {
    const livros = await this.findAll();
    return livros.find(l => l.id === id);
  }

  async create(livroData) {
    const livros = await this.findAll();
    const novoId = await this.getNextId();
    const novoLivro = { id: novoId, ...livroData };
    livros.push(novoLivro);
    await this._saveToFile(livros);
    return novoLivro;
  }
}
```

src/repositories/livros.repository.js (continuação)

```
async update(id, dadosAtualizados) {
  const livros = await this.findAll();
  const indice = livros.findIndex(l => l.id === id);
  if (indice === -1) throw new Error("Livro não encontrado");

  livros[indice] = { ...livros[indice], ...dadosAtualizados };
  await this._saveToFile(livros);
  return livros[indice];
}

async delete(id) {
  const livros = await this.findAll();
  const indice = livros.findIndex(l => l.id === id);
  if (indice === -1) throw new Error("Livro não encontrado");

  const livroRemovido = livros[indice];
  livros.splice(indice, 1);
  await this._saveToFile(livros);
  return livroRemovido;
}

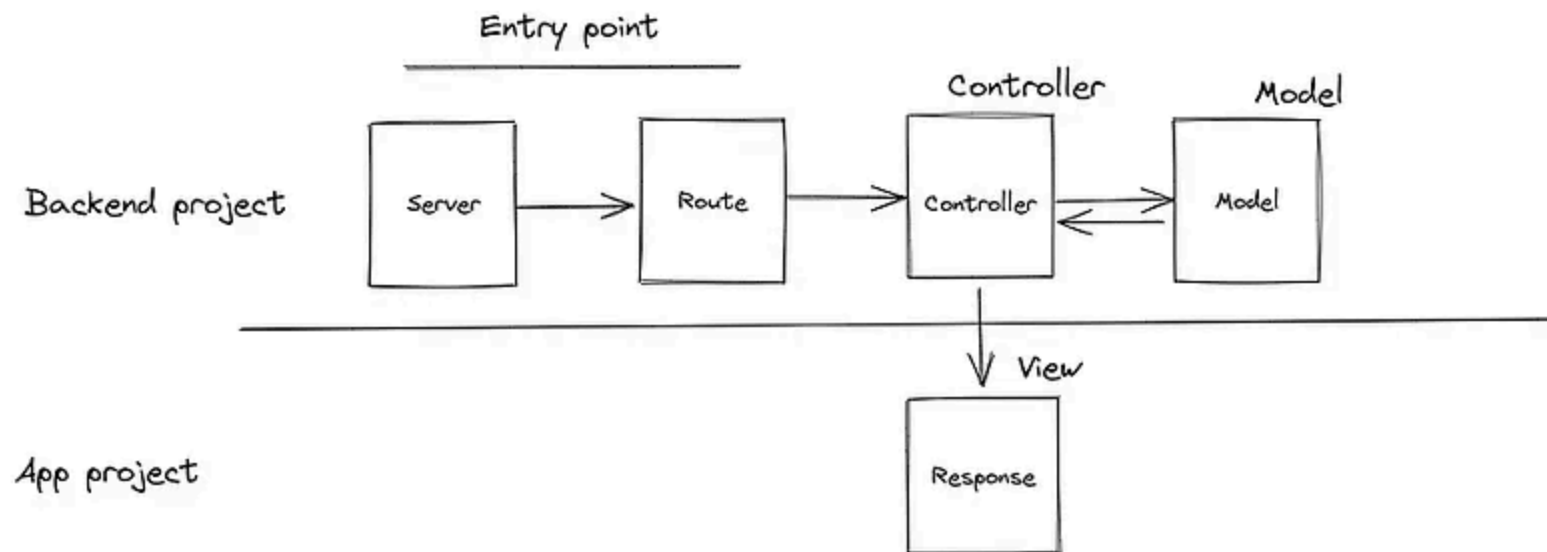
async _saveToFile(data) {
  fs.writeFileSync(this.caminhoArquivo, JSON.stringify(data, null, 2), "utf8");
}

async _lerArquivo() {
  return await fs.promises.readFile(this.caminhoArquivo, "utf8");
}

module.exports = LivrosRepository;
```

Refatorando o Controller

Agora o `LivrosController` não manipula mais diretamente o JSON, mas usa o repositório especializado para isso.



Na camada Model, o `Repository` atua como um intermediário entre o Controller e a fonte de dados (JSON).

src/controllers/livros.controller.js

```
const LivrosRepository = require("../repositories/livros.repository");

class LivrosController {
  constructor() {
    this.repository = new LivrosRepository();
  }

  async listarLivros(req, res, next) {
    const livros = await this.repository.findAll();
    res.status(200).json(livros);
  }

  async buscarLivroPorId(req, res, next) {
    const id = parseInt(req.params.id);
    const livro = await this.repository.findById(id);
    if (!livro) {
      return res.status(404).json({ erro: "Livro não encontrado" });
    }
    res.status(200).json(livro);
  }
}
```

src/controllers/livros.controller.js (continuação)

```
async criarLivro(req, res, next) {
  const { titulo, autor, categoria, ano } = req.body;
  const novoLivro = await this.repository.create({
    titulo,
    autor,
    categoria,
    ano: parseInt(ano)
  });
  res.status(201).json({
    mensagem: "Livro criado com sucesso",
    data: novoLivro
  });
}

async atualizarLivro(req, res, next) {
  const id = parseInt(req.params.id);
  const dados = req.body;
  const livroAtualizado = await this.repository.update(id, dados);
  res.status(200).json({
    mensagem: "Livro atualizado com sucesso",
    data: livroAtualizado
  });
}
// ...
```

src/controllers/livros.controller.js (continuação)

```
// ...
  async removerLivro(req, res, next) {
    const id = parseInt(req.params.id);
    const livroRemovido = await this.repository.delete(id);
    res.status(200).json({
      mensagem: "Livro removido com sucesso",
      data: livroRemovido
    });
  }
}

module.exports = LivrosController;
```

Rotas permanecem iguais

```
const express = require("express");
const router = express.Router();
const LivrosController = require("../controllers/livros.controller");
const livrosController = new LivrosController();
const { validarLivro, validarParamId } = require("../middlewares/validar/livros.validar");

router.get("/", (req, res, next) => livrosController.listarLivros(req, res, next));
router.get("/:id", validarParamId, (req, res, next) => livrosController.buscarLivroPorId(req, res, next));
router.post("/", validarLivro, (req, res, next) => livrosController.criarLivro(req, res, next));
router.put("/:id", validarParamId, validarLivro, (req, res, next) => livrosController.atualizarLivro(req, res, next));
router.delete("/:id", validarParamId, (req, res, next) => livrosController.removerLivro(req, res, next));

module.exports = router;
```

Desafios

- Instalar a extensão da ferramenta Postman no VSCode.
- Testar todas as rotas da API usando o Postman.