

Tanjil Khan, Luís Martins, Ricardo Magalhães

Pull request da implementação de POO: <https://github.com/tskxz/stockmaster/pull/18>

Neste commit, implementamos conceitos de poo, refatoramos o código para aplicar encapsulamento: <https://github.com/tskxz/stockmaster/pull/18/commits/e514562bf4866ea55f9e3e75db9515232cfc2dcf>

Fizemos a separação de responsabilidades, criamos as classes de serviços, como por exemplo ProdutoService para nós podermos separar a lógica de negócio (regras de criação, edição, validação, etc.) dos controllers. Seguimos o princípio de Single Responsibility (Responsabilidade Única), onde cada classe ou módulo tem uma única responsabilidade.

Controller: Lida com a interação HTTP (receber requisições e enviar respostas).

Service: Contém a lógica de negócio e interage com os modelos (Base de dados).

Fizemos encapsulamento. Por exemplo, encapsulamos a lógica relacionada a produtos dentro da classe ProdutoService. Os detalhes de implementação (como validações e interações com a base de dados) estão escondidos dentro da classe, expondo apenas uma interface limpa para os controllers.

```
server/services/ProdutoService.js
@@ -0,0 +1,130 @@
+ const ProdutoModel = require("../models/Produto");
+ const ArmazenModel = require("../models/Armazen");
+ const CategoriaModel = require("../models/Categoria");
+
+ class ProdutoService {
+   // Método para criar um produto com validações de estoque
+   async criarProduto(dadosProduto, empresaId) {
+     const { nome, descricao, preco, stock_total, stock_minimo, armazenId, categoriaId } = dadosProduto;
+
+     // Verifica se o armazém pertence à empresa autenticada
+     const armazen = await ArmazenModel.findOne({ _id: armazenId, empresa: empresaId });
+     if (!armazen) {
+       throw new Error("O armazém não pertence à empresa autenticada ou não foi encontrado.");
+     }
+
+     // Verifica se a categoria pertence à empresa
+     let categoria = null;
+     if (categoriaId) {
+       categoria = await CategoriaModel.findOne({ _id: categoriaId, empresa: empresaId });
+       if (!categoria) {
+         return res.status(403).json({
+           status: "error",
+           message: "A categoria não pertence à empresa autenticada ou não foi encontrada.",
+         });
+       }
+     }
+
+     // Verifica se o estoque total excede a capacidade do armazém
+     const produtos = await ProdutoModel.find({ armazen: armazenId });
+     const totalStockAtual = produtos.reduce((total, produto) => total + produto.stock_total, 0);
+     if (totalStockAtual + stock_total > armazen.capacidade) {
+       throw new Error("A capacidade do armazém seria excedida.");
+     }
+
+     // Verifica se o stock mínimo ou total excedem a capacidade do armazém
+     if (stock_total > armazen.capacidade || stock_minimo > armazen.capacidade) {
+       throw new Error("O stock total e o stock mínimo não podem ultrapassar a capacidade do armazém.");
+     }
+
+     // Define o status do produto
+   }
+ }
```

```
server/controllers/produtoController.js
@@ -75,147 +19,44 @@ const criarProduto = async function (req, res) {
+
+ const getProdutosByArmazen = async function (req, res) {
+   try {
+     const { armazenId } = req.params;
+
+     // Verifica se o armazém pertence à empresa autenticada
+     const armazen = await Armazen.findOne({ _id: armazenId, empresa: req.empresa_id });
+     if (!armazen) {
+       return res.status(403).json({
+         status: "error",
+         message: "O armazém não pertence à empresa autenticada ou não foi encontrado.",
+       });
+     }
+
+     // Busca os produtos do armazém
+     const produtos = await Produto.find({ armazen: armazenId }).populate("categoria");
+
+     const produtos = await produtoService.getProdutosByArmazen(req.params.armazenId, req.empresa_id);
+     res.status(200).json({
+       status: "success",
+       results: produtos.length,
+       data: {
+         produtos,
+       },
+     });
+   } catch (error) {
+     res.status(500).json({
+       status: "error",
+       message: error.message,
+     });
+   }
+ }
```

Aplicamos conceitos de POO

Classes e Objetos: Criamos classe para encapsular a lógica de produtos.

Métodos: Cada método da classe tem uma responsabilidade clara (criar, buscar, editar etc...).

Encapsulamento: A lógica de negócio está protegida dentro da classe, expondo apenas o necessário para os controllers.

Abstração: Os controllers não precisam saber como os produtos são criados ou validados, apenas chamam os métodos.

Neste commit, aplicamos a herança e polimorfismo

<https://github.com/tskxz/stockmaster/pull/18/commits/abed18fb79d65c9fa510d806ba6803d80c6eaa8d>

Herança: Herdamos as classes permitindo compartilhar métodos comuns de criação e edição de produtos.

Polimorfismo: ProdutoService sobrescreve (polimorfismo) os métodos criarProduto e editarProduto para incluir validações específicas de armazém, categoria e capacidade, enquanto ainda usa a lógica comum da classe base.