Tanjil Khan, Luís Martins, Ricardo Magalhães

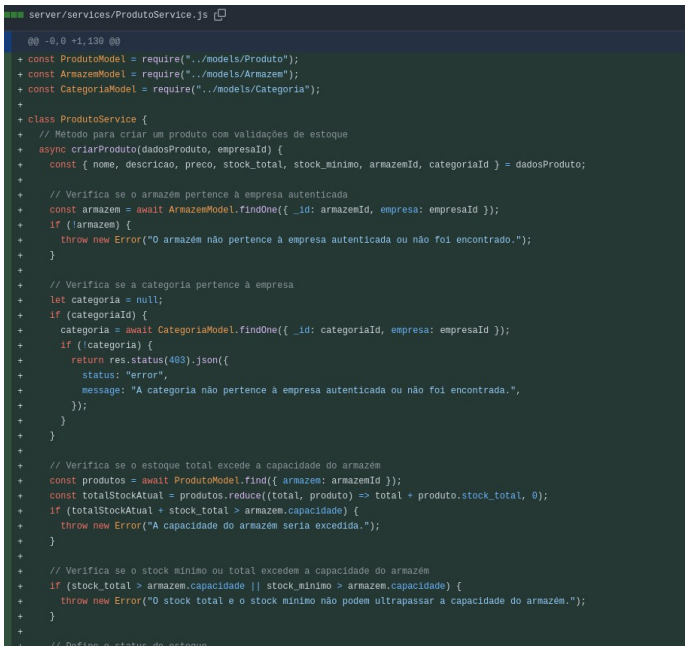
Pull request da implementação de POO: https://github.com/tskxz/stockmaster/pull/18

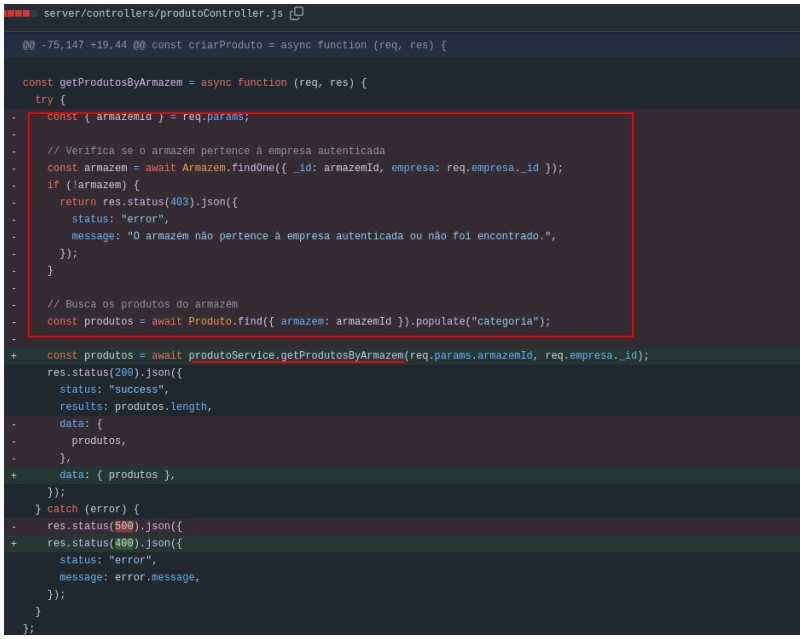
Neste commit, implementamos conceitos de poo, refatoramos o codigo para aplicar encapsulamento: https://github.com/tskxz/stockmaster/pull/18/commits/e514562bf4866ea55f9e3e75db9515232cfc2dcf

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.





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.