

SISTEMA DE RECOMENDAÇÃO DE --- CARTAS



O que é um Design Pattern?

Um design pattern é uma solução geral reutilizável para problemas recorrentes no desenvolvimento de software.

- **Benefícios:**

- Facilita a manutenção do código.
- Aumenta a reutilização de lógica e componentes.
- Torna o código mais compreensível e escalável.



Design Patterns usados no Projeto

- **Strategy:** Lida com a recomendação de cartas baseada em tipos específicos.
- **Singleton:** Gerencia a conexão com o banco de dados, garantindo uma única instância para todo o sistema.



Padrão Strategy – Teoria

- **Objetivo:** Permitir que uma família de algoritmos seja definida, encapsulada e selecionada em tempo de execução.
- **Vantagem:** Substituir a lógica de seleção (neste caso, das cartas) de maneira flexível, sem alterar a estrutura de código existente.
- **Aplicação:** Com o padrão Strategy, cada tipo de recomendação (mágico, corpo a corpo, distância) é implementado como uma estratégia separada. Isso facilita a adição de novos critérios de recomendação, mantendo o código modular e de fácil manutenção.

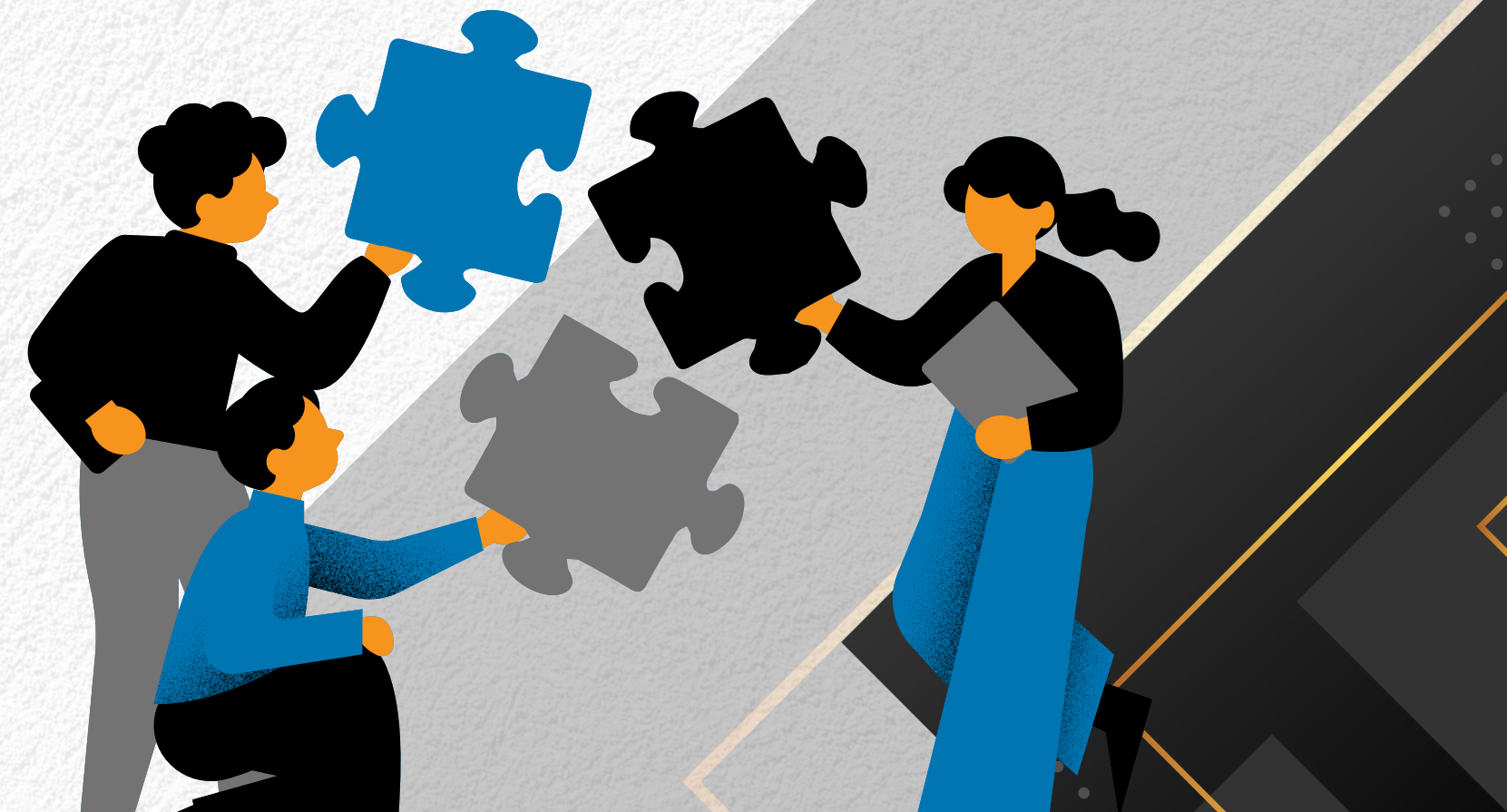


Padrão Strategy no Projeto

- **Estrutura:**

- **Strategies:** MagicStrategy, MeleeStrategy, e RangeStrategy, que são classes de estratégia para diferentes recomendações de cartas.
- **Interface de Chamadas:** Na aplicação, selecionamos uma estratégia específica conforme o tipo de carta solicitado pelo usuário.

```
if (type === "distancia") {  
    result = await RangeStrategy.build(Card);  
} else if (type === "corpo") {  
    result = await MeleeStrategy.build(Card);  
} else if (type === "magico") {  
    result = await MagicStrategy.build(Card);  
} else {  
    return res.status(400).json({ message: "Tipo de carta inválido" });  
}
```



Padrão Singleton – Teoria

- **Objetivo:** Garantir que uma classe tenha apenas uma instância, com ponto de acesso global.
- **Vantagem:** Reduz o uso desnecessário de recursos e facilita o gerenciamento do estado.
- **Aplicação:** No nosso projeto, usamos o Singleton para a conexão do banco de dados. Isso evita a criação de múltiplas conexões e reduz a complexidade de código ao garantir uma única instância compartilhada em todo o sistema.



Padrão Singleton no Projeto

- O arquivo db.js cria e exporta uma instância única do banco de dados usando Sequelize, garantindo uma conexão centralizada.

```
const { Sequelize } = require("sequelize");

// Criação da instância do Sequelize
const sequelize = new Sequelize({
  dialect: "postgres",
  host: "localhost",
  port: 5432,
  username: "my_user",
  password: "my_password",
  database: "my_database",
});
```



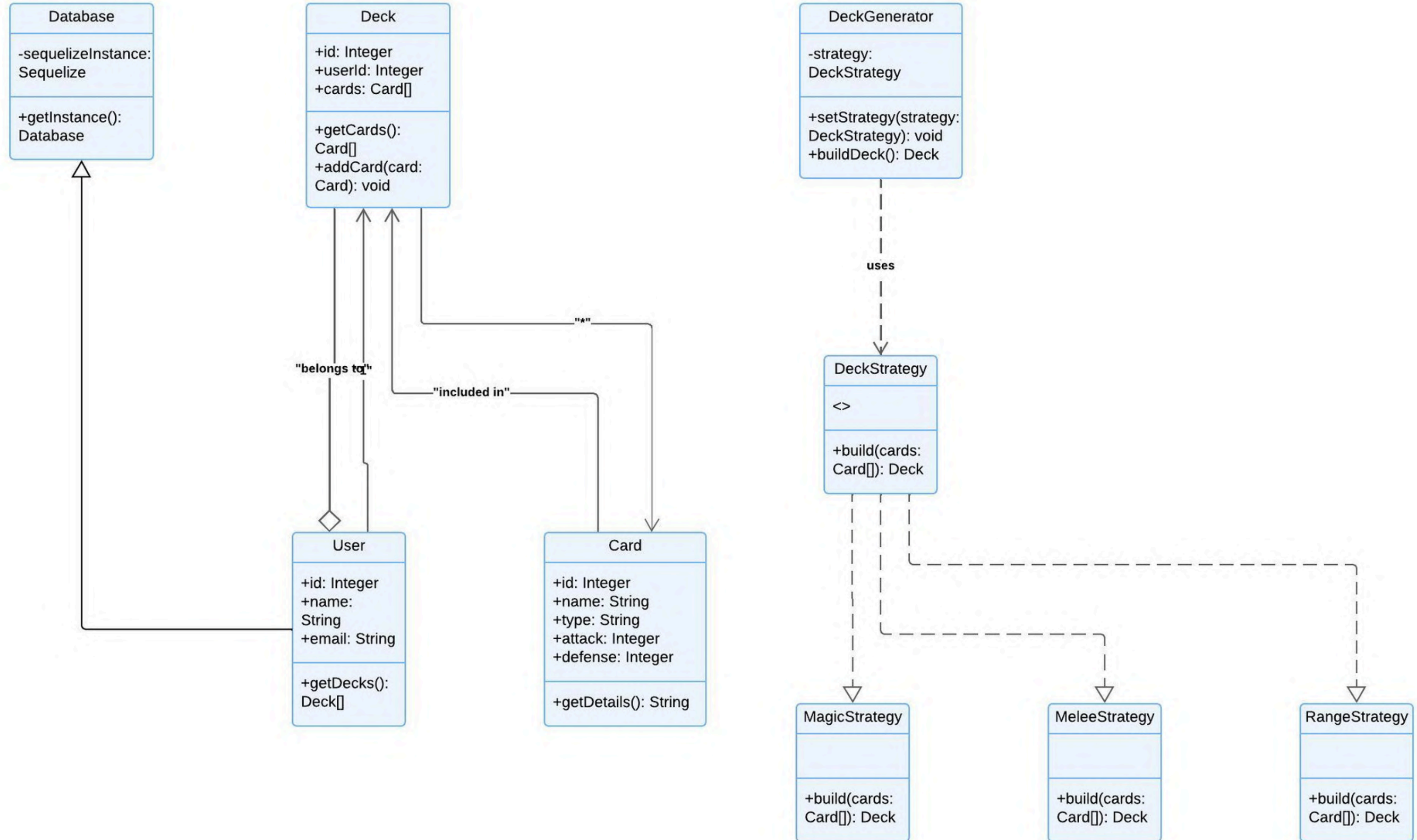
Benefícios dos Design Patterns no Projeto



- **Manutenção Simples:** A estrutura modular com Strategy permite adicionar novos tipos de recomendação sem modificar o código existente.
- **Eficiência e Consistência:** O padrão Singleton evita a criação de várias conexões com o banco, otimizando recursos.
- **Facilidade para Escalar:** Com o uso desses padrões, nosso projeto pode ser expandido facilmente para oferecer novas funcionalidades.



Diagrama de Classes



THANK YOU

