Padrões de Projeto no Sistema de Reserva de Laboratórios e Salas

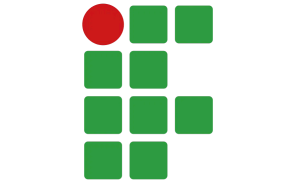
Aplicação prática dos conceitos de Engenharia de Software II

Nome do Aluno: Jhonatas Gomes Ribeiro

Disciplina: Engenharia de Software II

Instituição: IFPI, Campus Corrente

Data: 28 de Julho



Padrões de Projeto Utilizados

Para o desenvolvimento do Sistema de Reserva de Laboratórios e Salas, foram implementados três padrões de projeto de categorias diferentes, cada um com propósitos específicos e complementares.

Singleton

Padrão Criacional

Garante que uma classe tenha apenas uma instância e forneça um ponto de acesso global a ela.

Facade

Padrão Estrutural

Fornece uma interface unificada para um conjunto de interfaces em um subsistema, definindo uma interface de nível mais alto que torna o subsistema mais fácil de usar.

Observer

Padrão Comportamental

Define uma dependência um-para-muitos entre objetos, de modo que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente.

Angular (Frontend)

NestJS (Backend)

INSTITUTO FEDERAL
Piauí

Singleton (Criacional)

Definição

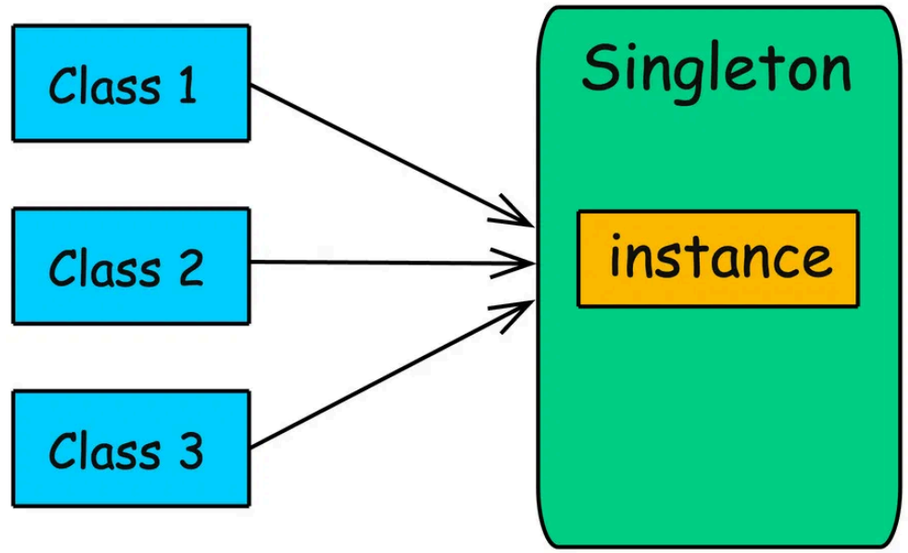
Garante que uma classe tenha apenas uma instância e forneça um ponto de acesso global a ela.

Onde é utilizado

- **Frontend (Angular):** Todos os serviços (AuthService, UserService, ReservasService, etc.) são implementados como Singletons através do decorador `@Injectable({ providedIn: 'root' })`.
- **Backend (NestJS):** Todos os serviços são Singletons por padrão dentro do escopo do módulo ou da aplicação, devido ao sistema de Injeção de Dependência do NestJS.

Justificativa

Para serviços que gerenciam um estado global (como o usuário autenticado, o estado de alertas) ou que interagem com recursos únicos (como a conexão com o banco de dados via PrismaService), ter uma única instância evita inconsistências e otimiza o uso de recursos.




```
// Exemplo Frontend: src/app/core/services/alerta.service.ts
@Injectable({
  providedIn: 'root' // Garante que há apenas uma instância global
})
export class AlertaServiceTs {
  // ...
}

// Exemplo Backend: src/prisma/prisma.service.ts
@Injectable() // Serviços NestJS são Singletons por padrão
export class PrismaService {
  // ...
}
```

INSTITUTO FEDERAL
Piauí

Facade (Estrutural)

Definição

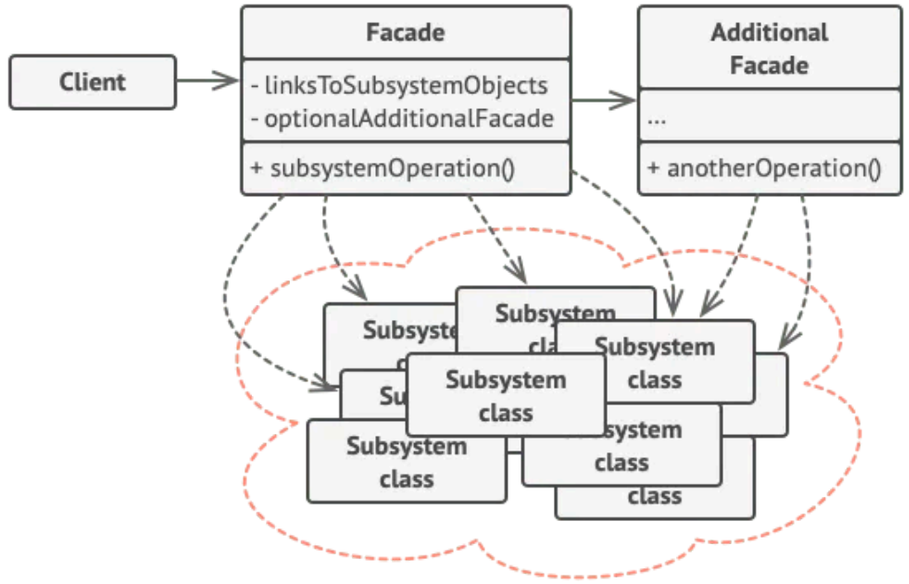
Fornece uma interface unificada para um conjunto de interfaces em um subsistema. Define uma interface de nível mais alto que torna o subsistema mais fácil de usar.

Onde é utilizado

- **Frontend (Angular Services):** Serviços como UserService, ReservasService e SalaService atuam como fachadas para as complexidades das requisições HTTP e do tratamento de erros.
- **Backend (NestJS Services):** Serviços como UsuariosService, ReservasService e SalasService atuam como fachadas para as operações complexas do Prisma ORM e para a lógica de banco de dados.

Justificativa

Reduz a complexidade e o acoplamento entre as camadas da aplicação, tornando o sistema mais fácil de usar, entender e manter. Um componente não precisa conhecer os detalhes de implementação, apenas utiliza a interface simplificada.




```
// Exemplo Frontend: src/app/core/services/user.service.ts
export class UserService {
  constructor(private http: HttpClient,
              private alertaService: AlertaServiceTs) {}

  register(dto: CreateUsuarioDto): Observable {
    return this.http.post(this.apiUrl, dto).pipe(
      tap(() => this.alertaService.success('Usuário registrado!')),
      catchError(this.handleError.bind(this))
    );
  }
  // ... outros métodos ...
}
```

INSTITUTO FEDERAL
Piauí

Observer (Comportamental)

Definição

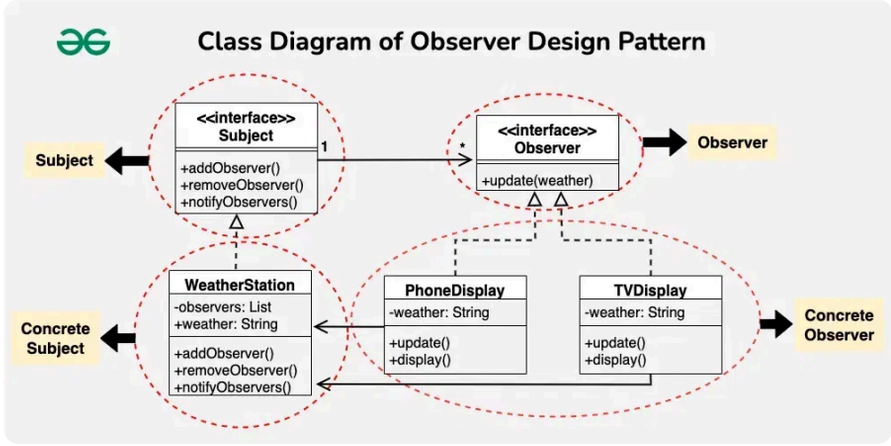
Define uma dependência um-para-muitos entre objetos, de modo que quando um objeto (o "observável") muda de estado, todos os seus dependentes (os "observadores") são notificados e atualizados automaticamente.

Onde é utilizado

- **Frontend (Angular com RxJS):** Amplamente utilizado em toda a aplicação Angular.
- **AuthService:** O BehaviorSubject papelUsuario exposto como userRole$ é um observável.
- **AlertaServiceTs:** O Subject toastSubject exposto como toast$ é um observável.

Justificativa

Promove um forte desacoplamento entre a fonte dos dados e os consumidores dos dados, facilitando a programação reativa e assíncrona.




```
// Exemplo Frontend (Observável): src/app/auth/services/auth.service.ts
import { BehaviorSubject } from 'rxjs';
export class Auth {
  private papelUsuario = new BehaviorSubject(null);
  userRole$ = this.papelUsuario.asObservable(); // Observável

  public login(...) {
    // ... após login bem-sucedido ...
    this.papelUsuario.next(decodificado.tipo); // Notifica observadores
  }
}

// Exemplo Frontend (Observador): src/app/shared/components/menu.component.ts
export class Menu implements OnInit {
```

INSTITUTO FEDERAL
Piauí

Outros Conceitos e Padrões Relevantes

Factory Method (Criacional)

Embora não haja uma classe "Factory" explícita, os métodos create em seus serviços (ex: UsuariosService.create, ReservasService.create) atuam como "métodos de fábrica" simplificados, encapsulando a lógica de criação de objetos.

Injeção de Dependência (DI)

É o princípio arquitetural subjacente que permite a aplicação de Singletons, Facades e a testabilidade. Angular e NestJS são fortemente baseados em DI.

Data Transfer Object (DTO)

Utilizado extensivamente para definir a estrutura dos dados transferidos entre as camadas (frontend e backend), garantindo tipagem e validação.

Validação em Camadas

A validação de dados é aplicada em múltiplas camadas: Frontend (Angular Reactive Forms), Backend (class-validator) e Banco de Dados (UNIQUE CONSTRAINT e Triggers MySQL).

Cascade Delete

Configurado no schema.prisma (onDelete: Cascade) para a relação Reserva -> Sala, garantindo que reservas sejam automaticamente excluídas quando uma sala é removida.

Stored Procedures e Functions

Utilizados para encapsular a lógica de CRUD para SalasService e filtrar dados no ReservasService. Isso delega a lógica de consulta/modificação para o banco de dados.

Conclusão

Benefícios da Aplicação dos Padrões

A implementação dos padrões de projeto no Sistema de Reserva de Laboratórios e Salas trouxe diversos benefícios:

- Maior modularidade e desacoplamento entre os componentes do sistema
- Facilidade de manutenção e extensão do código
- Melhor organização da arquitetura do software
- Reutilização de código e redução de duplicação
- Melhor gerenciamento de estado e comunicação entre componentes

Considerações Finais

Os padrões de projeto são ferramentas poderosas que, quando aplicados corretamente, melhoram significativamente a qualidade do software. Neste projeto, a combinação de padrões criacionais, estruturais e comportamentais resultou em uma arquitetura robusta e flexível.
A experiência adquirida com a aplicação prática desses conceitos de Engenharia de Software II demonstra a importância do conhecimento teórico aliado à implementação em cenários reais.