Documentação de Boas Práticas de Uso de Padrões de Projeto – Mind System - Grupo 3

Objetivo

Padronizar a arquitetura e garantir organização, extensibilidade e manutenção simples do sistema. Os padrões aqui definidos seguem boas práticas de desenvolvimento orientado a objetos e arquiteturas modernas (Domain-Driven Design, Clean Architecture e microservices quando aplicável).

---

1. Factory Method

Onde usar

Criação de objetos complexos que variam conforme o tipo, especialmente:

- **Questionários** (vocacional, emocional, personalizado)
- **Laudos** (resumo, completo, clínico)
- **Pesquisas de feedback** (satisfação, comportamento, engajamento)

Justificativa

Permite criar objetos sem expor a lógica complexa de construção e facilita adicionar novos tipos sem modificar código existente.

Exemplo de Implementação

```
abstract class QuestionarioFactory {
    abstract criar(): Questionario;
}

class QuestionarioVocacionalFactory extends QuestionarioFactory {
    criar() {
        return new QuestionarioVocacional();
    }
}

class QuestionarioEmocionalFactory extends QuestionarioFactory {
    criar() {
        return new QuestionarioEmocional();
    }
}

// Uso
const factory = new QuestionarioVocacionalFactory();
const questionario = factory.criar();
```

2. Publisher/Subscriber (Observer / Event Bus)

Onde usar

- Notificações de:
    - **Novos laudos gerados** (para o aluno)
    - **Alteração/confirmação de consultas**
    - **Disponibilização de novos questionários/pesquisas**
    - **Eventos de autenticação** (ex.: avisar administrador sobre logins suspeitos)

Justificativa

Minimiza acoplamento e permite troca futura de tecnologias como Mail, Firebase, RabbitMQ, Kafka.

Exemplo

```
class EventBus {
    private subscribers = {};

    subscribe(event, handler) {
        this.subscribers[event] = this.subscribers[event] || [];
        this.subscribers[event].push(handler);
    }

    publish(event, data) {
        (this.subscribers[event] || []).forEach(h => h(data));
    }
}

// Publicação
eventBus.publish("LAUDO_GERADO", laudo);

// Assinatura
eventBus.subscribe("LAUDO_GERADO", (laudo) => enviarEmail(laudo));
```

3. Repository Pattern

Onde usar

Em todas as entidades persistentes:

- Usuario, Psicologo, Administrador
- Questionario, Pergunta, Resposta
- Consulta
- Laudo

- PesquisaFeedback

Justificativa

Abstrai o acesso ao banco, facilitando:

- troca de database
- testes
- evitar SQL disperso pelo código
- manter domínio desacoplado

Exemplo

```
interface UsuarioRepository {
    salvar(usuario: Usuario);
    buscarPorEmail(email: string): Usuario;
}

class UsuarioRepositorySQL implements UsuarioRepository {
    salvar(usuario) {
        db.query("INSERT INTO usuarios ...");
    }
}
```

---

4. Singleton Pattern

Onde usar

Componentes que devem ter **instância única** na aplicação:

- Gerenciador de autenticação OAuth (Login Social – RF1)
- Gerenciador de PDF (Gerar Laudo PDF – RF5)
- EventBus (caso não use Kafka/RabbitMQ diretamente)
- Logger central
- Conexão de banco

Exemplo

```
class GeradorPDF {
    private static instancia;

    private constructor() {}

    static getInstancia() {
        if (!GeradorPDF.instancia) {
            GeradorPDF.instancia = new GeradorPDF();
        }
```

```
        return GeradorPDF.instancia;
    }
}
```

5. Adapter Pattern

Onde usar

Casos de uso que integram com serviços externos:

- **Login Social com Google** (API Google OAuth)
- **Consulta Virtual** (Google Meet)
- **Envio de e-mails e notificações**

Justificativa

Padroniza a interface interna do sistema, isolando APIs externas que podem mudar ao longo do tempo.

Exemplo

```
// Interface interna
interface ServicoAutenticacaoSocial {
    autenticar(token: string): Usuario;
}

// Adaptação do Google
class GoogleAuthAdapter implements ServicoAutenticacaoSocial {
    autenticar(token) {
        const googleData = GoogleAPI.verify(token);
        return new Usuario(googleData.email);
    }
}
```

6. DTO (Data Transfer Object)

Onde aplicar

Toda interação entre camadas ou APIs:

- Requests/responses
- Transporte de dados entre UI ↔ Backend
- Exportação/importação de dados

Justificativa

Evita expor modelos internos e melhora segurança (RNF5 – Autenticação segura).

Exemplo

```
class UsuarioDTO {
    id: number;
    nome: string;
    email: string;
}
```

Resumo dos Padrões

Padrão	Motivação	Onde Aplicar
Factory Method	Criar objetos complexos	Questionário, Laudo, Pesquisa
Pub/Sub	Eventos assíncronos	notificações, consultas, laudos
Repository	Consistência e desacoplamento do BD	todas as entidades
Singleton	Instância única	PDF, Auth, DB, EventBus
Adapter	Integração externa	Google OAuth, Google Meet
DTO	Segurança e padronização de APIs	Casos de uso com interação entre camadas ou APIs