# HyPEADVice

## Visão Geral do Projeto

HypeAdvice é uma aplicação Web que combina a praticidade de Conselho.ela. Consumo uma API pública chamada AdviceSlip que gera conselhos aleatórios em inglês. O usuário pode salvar esses conselhos no banco de dados local com uma Tipo (opcional ou Pago), pesquisar por descrição ou buscar um conselho pelo ID direto na API

## Telas Disponíveis

Localhost:8080/advice-crud.xhtml
Localhost:8080/advice-list.xhtml
Localhost:8080/swagger-ui.html

## Estrutura do Projeto

My Opinion

- domain/
  - bean/ → Controla os filtros (nome, tipo), chama o serviço, mostra mensagens
  - controller/ → Endpoints REST (GET /listar, POST /salvar, trata o payload recebe, diz controlar as ROTAS API)
  - entity/ → É uma classe anotada com @Entity, diz ao JPA que a classe representa uma Tabela, Responde a Linha específica com as colunas da Tabela
  - enums/ → Os tipos possíveis. Ex: Conselho ou Notícia. Usado pelo Advice.
  - exception/ → Erros customizados quando algo (Inesperado) i um erro de lógica. Usados pelo Service para a Interface, o Bean mostrar a mensagem ou aviso de problema. Fora da regra de negócios)
  - repository/ → Acesso ao banco de dados (Ou que falando é onde realizamos as persistências com o JPA, Ex: que é onde, removal/inserir, pesquisa, no banco e Conexão, Transação)
  - service/ → Regras de negócio (Lógica. Ou que o sistema faz)
  - utils/ → Utilitários, auxiliares
  - vo/ → objetos de resposta da API Externa, basicamente o espelho da API que CONSUMIMOS, posteriormente Vem. RETURN

- webapp/
  - advice-crud.xhtml
  - advice-list.xhtml

## Fluxo de Cadastros

1. Usuário preenche o formulário
   - Campo nome
   - Campo descrição (Required="true")
   - Campo Tipo (Select: GRATUITO ou PAGO, Required="true")

2. Advice-crud.xhtml (Tela do JSF)

```
<p:commandButton actionListener="#{adviceBean.salvar}"
                 update="form-advice-list:messagensGlobais"
                 value="Salvar" />
```

⚠ Antes de chegar no Bean, o JSF executa as validações dos campos marcados com Required (Descrição e Tipo)

↓ Se passar nas validações

3. Advice.salvar() Controller da apresentação / Controller JSF

```
Validação Adicional Manual:
    If (advice.getTipo() == null) {
        addFacesMessage(SEVERITY_ERROR, "Erro", "...");
        Return;
    }

↳ Se passou:
    Advice Service.save(Advice); → Chama a Camada de Serviço
    Advice.add(Advice); → Atualiza a lista em memória
    ResetAdvice(); → Limpa o formulário
    addFacesMessage(SEVERITY_INFO, "Sucesso", "...");
```

4. Advice Service.save(Advice) Camada de regra de negócio

```
@Transactional(rollbackFor = Exception.class)
Public Advice save(Advice advice) {
    Return AdviceRepository.saveAndFlush(advice);
}
```

A Transactional garante: se der erro, faz ROLLBACK (desfaz tudo)

5. Advice Repository.saveAndFlush(advice) Camada de persistência

```
Public interface AdviceRepository extends JpaRepository<Advice, Long>
```

⚠ Não tem código! Quem implementa é o Spring Data JPA + Hibernate.
SaveAndFlush = Salvar e Forçar o envio imediato pro banco (diferente do save() comum, que pode agrupar).

6. Hibernate gera o SQL mágica JPA

```
INSERT INTO advice (nome, descricao, tipo) VALUES (? ? ?)
```

7. PostgreSQL banco de dados

√ Linha inserida na Tabela Advice
√ Gera o ID auto-incrementado

8. Advice Bean

√ Mensagem Global: "Conselho cadastrado com Sucesso."
√ Tabela "Conselhos Cadastrados" atualizada com o novo Registro
√ Formulário Limpo, Pronto pra Novo Cadastro

## Fluxo de Busca por ID

Esse fluxo NÃO usa o Banco PostgreSQL!
Ele consulta Uma API Externa (adviceslip.com)

1. Usuário acessa Advice-List.xhtml
   - VÊ Dois Formulários: Busca por descrição e Busca por ID
   - Digita o ID no Campo "Pesquisa por ID" (Input Type="NUMBER", Required)

↳ Ao clicar no Botão "Buscar por ID"

2. Advice-List.xhtml (Tela do JSF)

```
<p:commandButton actionListener="#{adviceListBean.buscarPorId}"
                 look="primary"
                 update="form-advice-list:messagensGlobais"
                 value="Buscar por ID" />
```

⚠ O JSF valida: Required="true" → ID não pode estar Vazio
O input Type="NUMBER" → Só aceita números

3. AdviceListBean.buscarPorId() Controller JSF

```
Public Void buscarPorId() {
    Try {
        This.adviceListVO = adviceService.buscarById(idBusca);
        addFacesMessage(SEVERITY_INFO, "Sucesso", "Conselho Localizado com Sucesso...");
    } Catch (Exception e) {
        addFacesMessage(Severity_Error, "Erro", e.getMessage());
    }
}
```



Bloco Try/Catch – Qualquer erro vira mensagem na Tela

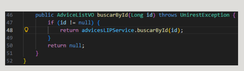
4. AdviceService.buscarById(id) Camada de regra.

```
Public AdviceListVO buscarById(Long id) Throws UnirestException {
    If (id != null) { → Valida se ID não é nulo
        Retorna AdvicesLIPService.buscarById(id); → Delega pro AdvicesLIPService (que é que fala com a API)
    }
    Return null
}
```

5. AdvicesLIPService.buscarById(id) Camada de integração externa

```
HttpResponse<String> response = Unirest.get(API_Base_URL + "/" + id)
    .Header("Accept-Language", "pt")
    .Header("Content-Type", "Aplication/Json")
    .AsString();
```

Usa Biblioteca Unirest para fazer GET HTTP
URL final: https://api.adviceslip.com/advice/{id}



6. API Externa – adviceslip.com

GET https://api.adviceslip.com/advice/15

Retorno JSON Exemplo:
{"slip":{"id":5,"advice":"Don't be afraid..."}}

↳ Resposta Vacta

7. AdviceLIPService Trata a resposta

```
Verifica o status HTTP:
|- Se status != 200 → Lança RuntimeException
|
|— Se status == 200:
|   |— Checa se body contém "advice slip not found" ou "type:notice"
|   |   Lança RecursoNaoEncontradoException
|   |
|   |— Se encontrou:
|       1. addUnknownData(body) → Ignora data atual no JSON
|          (porque a API não devolve data, mas a Tela precisa exibir)
|       2. Utils.jsonToObject() → Converte JSON em AdviceVO (Gson.com)
|       3. Empacota em AdviceListVO (que tem List<Slip>)
|       4. Retorna pro Service
```

8. Resposta sobra a Pilha
   AdviceLIPService → AdviceService → AdviceListBean

9. AdviceListBean Vale na Tela.

√ Caso Sucesso:
- AdviceListVO popula
- Tabela "Conselhos Localizados" MOSTRA: ID, Conselho, Data
- Mensagem Global: "Conselho Localizado com Sucesso."

X Caso Erro (Catch):
- Mensagem Global de Erro com a mensagem da Exception
- Ex: Conselho com ID 9999 não Encontrado.

## Confirmação da Minha Opinião Técnica

Beans?

Repository

Utils

Controller ?

Entity

Service

VO

Exception

## @ Annotations

O que é?

Anotação é um "Rótulo" que você gruda em uma Classe, Método, Atributo ou Parâmetro pra dar informação Extra sobre ele

Pense Assim: A anotação não faz nada sozinha. Ela é só um ADESIVO de metadado. Quem lê o Adesivo e age sobre ele é uma Framework (Spring, JPA, Hibernate, JSF).

Analogia:

Imagina caixas em um Depósito
Uma caixa com Adesivo "Frágil" → O Adesivo em si não protege nada
Mas o funcionário do Depósito lê o Adesivo e Trata Aquela Caixa com Cuidado

| Anotação | O que faz |
|---|---|
| @SpringBootApplication | Marca o ponto de Entrada da Aplicação |
| @Service | Registra a Classe como Serviço Gerenciado Pelo Spring |
| @Repository | Registra a Class/Interface como Camada de Persistência |
| @AutoWired | Injeta Dependência Automaticamente |
| @Transactional | Envolve o Método em uma Transação |
| @Named | Registra o Bean no JSF com um Nome |
| @ViewScoped | O Bean vive Enquanto a Página estiver |
| @Component | Anotação Genérica do Spring |

Linguagem Simples
É Onde o Programa começa
O Spring Cuida desse objeto pra Mim
Essa Camada Fala com o Banco
O Spring Cuida e Entrega o Objeto pra Mim
Se Falhar no Meio, Desfaz Tudo
O JSF Recebeu Esse Bean Pelo Nome
Os Dados não Somem, Ao Interagir com a Tela
Registra Qualquer Classe no Spring

## Resumo Fluxo ID

Estes são os destaques que vão te diferenciar:

**1. 🔄 Flow: Form/VO vira banco de dados**
- Diferente do fluxo de salvar, não tem PostgreSQL aqui
- Consulta uma API REST externa (adviceslip.com)
- Por que isso importa? Mostra que o projeto integra com sistemas externos, não é só CRUD com banco.

**2. Separação de responsabilidades clara**
- `AdviceService` → orquestrador, decide se delega pro banco ou pra API
- `AdvicesLIPService` → especialista com falar com a API externa
- Por que separar? Se um dia a API mudar URL, formato, só mexe em um lugar.

**3. Biblioteca Unirest para HTTP**
- Chrome: HTTP simples e fluente
- Faz GET com headers customizados (`Accept-Language`, `Content-Type`)

**4. Tratamento de "API inexistente"**
- A API do adviceslip.com **retorna status 200** mesmo quando o conselho não existe! 😱
- Por isso o código checa o **conteúdo do body** procurando `"Advice slip not found"` ou `"type":"notice"`
- Por que isso importa? Mostra que você não confia cegamente no status HTTTC valida o corpo da resposta.

**5. Exceção customizada** `RecursoNaoEncontradoException`
- Em vez de retornar `null` ou um genérico, lança uma exceção semântica
- Extends `RuntimeException` (unchecked)
- Permite tratamento específico no Beans e mensagem amigável pro usuário

**6. Truque de** `adviceVoDate()`
- A API externa não retorna data no JSON
- Mas a tela tem uma coluna "Data" pra mostrar
- Solução: inserimos a DATA como se tivesse vindo (injetando a data atual)
- Comentário pra defesa: "É uma solução pragmática — poderia ser feito após a conversão também, mas o autor escolheu fazer no JSON Utils."

**7. Padrão VO (Value Object)**
- `AdviceListVO`, `AdviceVO`, `Slip` → objetos que representam dados da API
- **Não são Entities** (não tem `@Entity`), não vão pro banco
- Por que separar de Entity? A API externa tem um formato próprio que não é igual à tabela do banco

**8.** `@JsonProperty("total_results")` **no VO**
- A API externa `total_results` (snake_case)
- O Java usa `totalResults` (camelCase)
- A anotação faz a tradução automática na desserialização

### 📐 Comparação dos 2 fluxos do projeto

Vale destacar isso na defesa — mostra que você entendeu a arquitetura inteira:

| Aspecto | Salvar (Cadastro) | Buscar por ID |
|---|---|---|
| Tela | advice-crud.xhtml | advice-list.xhtml |
| Bean | AdviceBean | AdviceListBean |
| Service | AdviceService | AdviceService + AdvicesLIPService |
| Persistência | AdviceRepository (JPA) | API externa (Unirest) |
| Destino final | PostgreSQL | api.AdvicesIlp.com |
| Objeto retornado | Advice (Entity) | AdviceListVO (VO) |
| Transação | `@Transactional` | Não tem (não há banco) |
| Tratamento de erro | Validação manual + JSF | try/catch + exceção customizada |