

```
entity Books : cuid {  
    title : String;  
    author : Association to Authors;  
}  
  
service CatalogService {  
    entity Books as projection on my.Books;  
}
```

```
annotate CatalogService.Books with @UI : {  
    LineItem : [  
        {Value : title, Label : 'Title'}  
    ]  
};
```

# Dominando o SAP CAP: Da Modelagem à Produção

Um guia avançado para construir aplicações enterprise-ready com as melhores práticas do Cloud Application Programming Model.

```
entity Orders : cuid {  
    customer : Association to Customers;  
    items : Composition of many OrderItems;  
}
```

# A Filosofia CAP: Capture a Intenção, Não a Implementação

O SAP CAP é um framework opinativo projetado para acelerar o desenvolvimento de aplicações de alta qualidade. Em vez de se perder em código imperativo, o CAP nos permite focar no modelo de domínio e na intenção de negócios.

- **O Quê, não Como**

Declaramos *o que* queremos (ex: um campo é traduzível, um dado é gerenciado), e o framework fornece implementações genéricas e otimizadas.

- **Provedores Genéricos**

Nossos modelos CDS alimentam diretamente o banco de dados, os serviços e as UIs, garantindo consistência e minimizando código boilerplate.

## A Jornada do Construtor

Seguiremos o ciclo de vida lógico da construção de uma aplicação, desde a planta baixa até a fortaleza pronta para produção. Cada etapa revela como o CAP oferece as ferramentas certas no momento certo.



Modelagem

Serviços

UI

Produção  
(Segurança & Performance)

# Etapa 1: A Planta Baixa - Modelagem de Domínio Sólida

## Conceito Chave

Um modelo de domínio limpo e compreensível é a base de tudo. O CAP, com seus aspectos reutilizáveis, nos ajuda a começar com o pé direito.

- **cuid**: Para chaves primárias canônicas e universalmente únicas. Garante que cada registro seja globalmente identificável, essencial em cenários distribuídos.
- **managed**: Para dados de auditoria essenciais. Adiciona automaticamente campos para rastrear quem criou/modificou e quando, preenchidos pelo runtime.

## Exemplo Prático

### Com `cuid`

```
// Com cuid
using { cuid } from '@sap/cds/common';
entity Books : cuid { /*...*/ }
```

```
// Equivalente a:
entity Books {
    key ID : UUID;
    /*...*/
}
```

### Com `managed`

```
// Com managed
using { managed } from '@sap/cds/common';
entity Foo : managed { /*...*/ }
```

```
// Equivalente a:
entity Foo {
    createdAt : Timestamp @cds.on.insert: $now;
    createdBy : User      @cds.on.insert: $user;
    modifiedAt : Timestamp @cds.on.insert: $now
                           @cds.on.update: $now;
    modifiedBy : User      @cds.on.insert: $user
                           @cds.on.update: $user;
}
```

**Com apenas duas palavras, `cuid` e `managed`, já implementamos um padrão robusto para identidade e auditoria de dados.**

# Melhores Práticas para uma Modelagem Clara e Eficaz

## Princípio 1

### Mantenha a Simplicidade (KISS): Prefira Modelos Planos.

Evite estruturas aninhadas profundas.  
Estruturas planas são mais fáceis de consumir e entender.

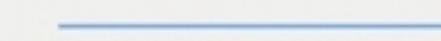
#### Ruim

```
// Ruim: aninhado e complexo
entity Contacts {
    isCompany : Boolean;
    companyData : CompanyDetails;
    personData : PersonDetails;
}
```



#### Bom

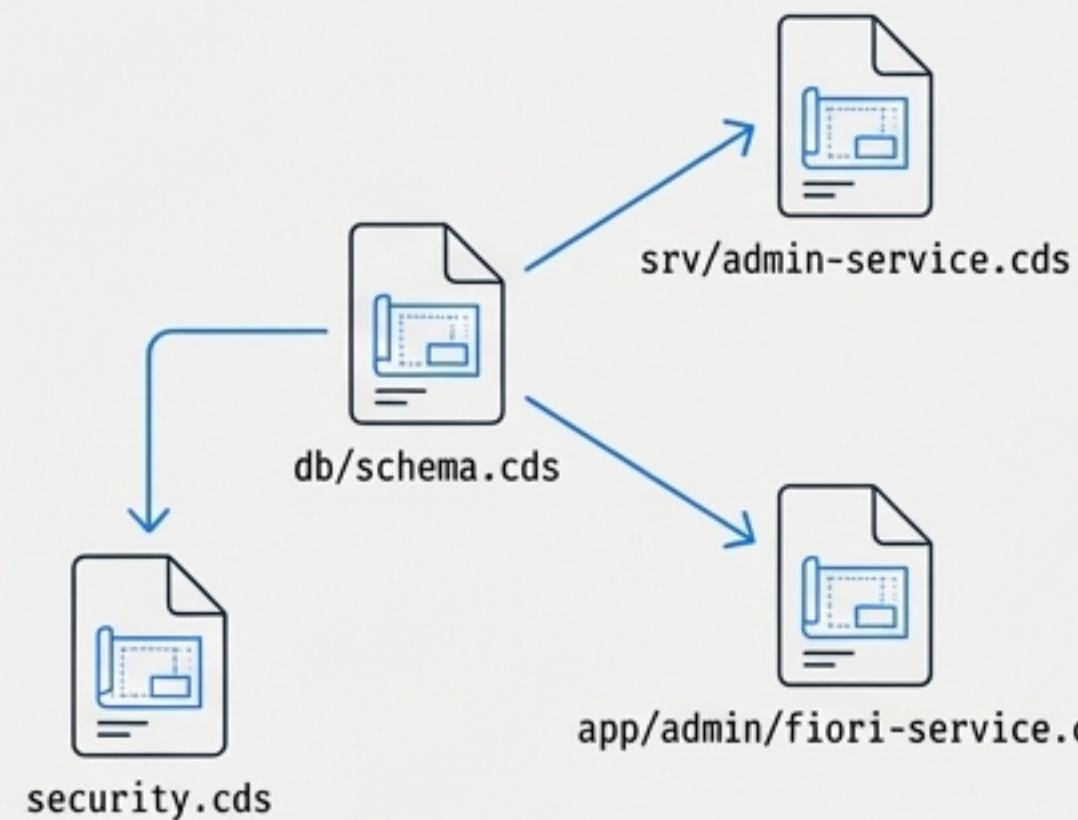
```
// Bom: plano e direto
entity Contacts {
    isCompany : Boolean;
    company : String;
    title : String;
    firstname : String;
    lastname : String;
}
```



## Princípio 2

### Separação de Responsabilidades

Use Aspectos para separar preocupações.  
Mantenha o modelo de domínio principal limpo, adicionando anotações de UI, segurança e outras em arquivos separados.



## Princípio 3

### Convenções de Nomenclatura

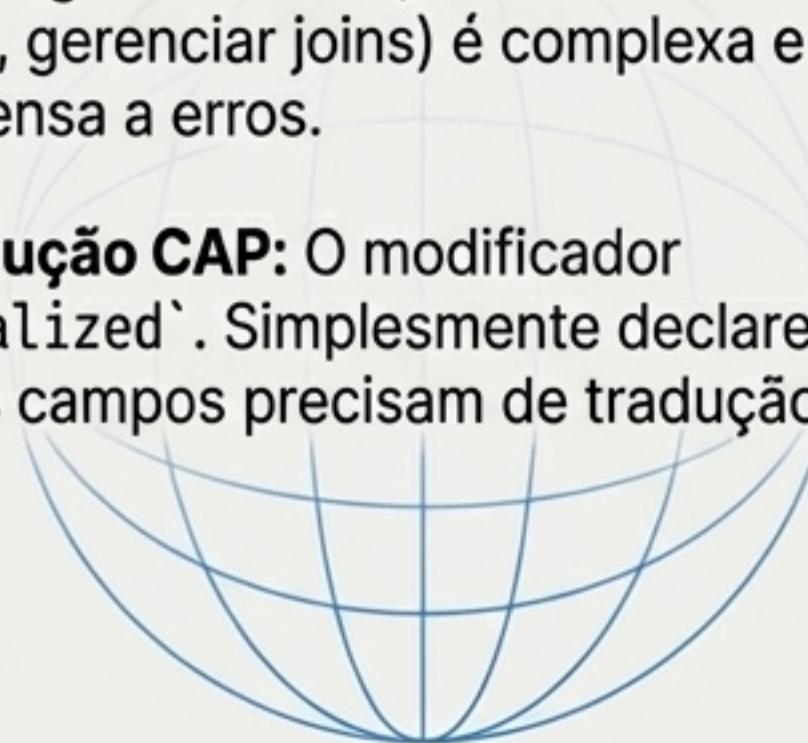
- Nomes de Entidades/Tipos:** Capitalized, Plural para entidades (Books), Singular para tipos (Genre).
- Nomes de Elementos:** lowerCase.
- Seja Conciso:** Prefira Authors.name em vez de Authors.authorName.

# Etapa 2: Construindo para o Mundo - Dados Localizados

## O Desafio

**O Desafio:** Aplicações empresariais precisam operar globalmente, exibindo dados no idioma do usuário. A abordagem manual (criar tabelas de texto, gerenciar joins) é complexa e propensa a erros.

**A Solução CAP:** O modificador `localized`. Simplesmente declare quais campos precisam de tradução.



## Exemplo de Código

```
entity Books {  
    key ID : UUID;  
    title : localized String; // <--- Simples assim!  
    descr : localized String;  
}
```

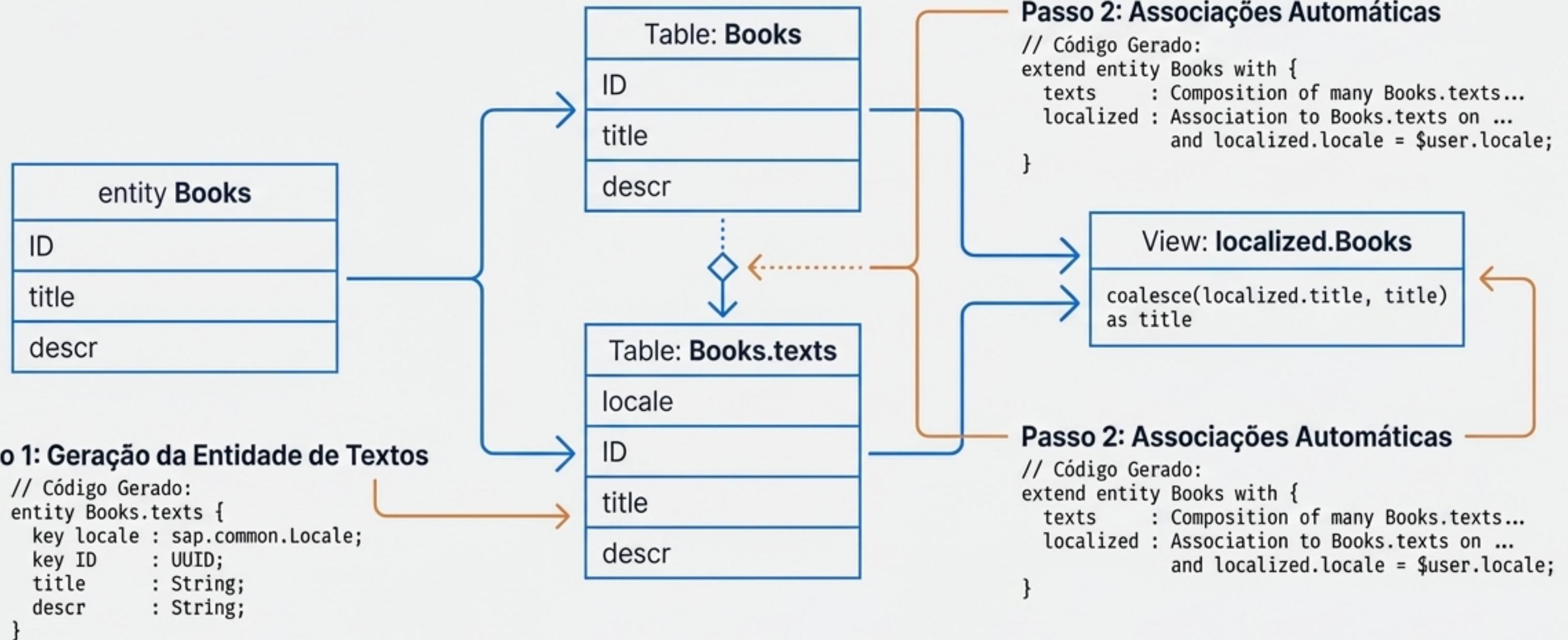
**Como Funciona (Visão Geral):** O CAP automaticamente gera a complexidade por trás dos panos.

1. Cria uma entidade `Books.texts` associada (ex: `Books.texts`).
2. Usa a pseudo-variável `'\$user.locale'` para buscar a tradução correta.
3. Gera views SQL com lógica de `coalesce` para fallback para o idioma padrão se uma tradução não existir.



**O CAP transforma um problema complexo de i18n em uma única palavra-chave, permitindo que o desenvolvedor se concentre na lógica de negócios.**

# Dados Localizados: A Engenharia por Trás da Simplicidade



## Adicionando Dados Iniciais

Requer dois arquivos CSV: `Books.csv` para o idioma padrão e `Books\_texts.csv` para as traduções.

# Etapa 3: Conectando Ecossistemas - Consumo de Serviços

**O Cenário:** Aplicações modernas são microserviços que se comunicam. Precisamos consumir dados de sistemas externos (ex: SAP S/4HANA) de forma segura e eficiente.

## Passo 1 (à esquerda)

### Importar a Definição da API

O CAP consome definições de serviço (EDMX) para entender a API remota.

**Comando Chave:** `cds import <arquivo_edmx> --as cds`

Isso gera um arquivo `\*.cds` na pasta `srv/external`.

```
[cds] - serving CatalogService at /browse
[cds] - serving CatalogService at /admin
[cds] - serving AdminService at /admin
[cds] - mocking API_BUSINESS_PARTNER { at: '/api-business-partner' }
[cds] - server listening on { url: 'http://localhost:4004' }
[cds] - launched in: 1.104s
[ terminate with ^C ]
```

**Mensagem Principal:** Desenvolva sua lógica de integração de ponta a ponta localmente, de forma rápida e independente de sistemas externos.

## Passo 2 (à direita)

### Mocking Local para Desenvolvimento

A maior vantagem: não é preciso ter uma conexão real com o sistema remoto para desenvolver.

**Como funciona:** Simplesmente execute `cds watch`. O CAP detecta o serviço externo e o simula automaticamente.

### Adicionando Dados Mock:

Crie um arquivo CSV na pasta `srv/external/data` para popular o serviço mockado.

# Integrando Dados Remotos com Projeções e Mashups

## Projeções (O Padrão Recomendado)

Crie uma 'interface' local para o serviço remoto usando projeções. Isso desacopla sua aplicação da API externa e permite selecionar apenas os campos necessários.

### Exemplo de Projeção:

```
using { API_BUSINESS_PARTNER as bupa } from '../srv/external/  
API_BUSINESS_PARTNER.cds';  
  
entity Suppliers as projection on bupa.A_BusinessPartner {  
    key BusinessPartner as ID,  
    BusinessPartnerFullName as fullName,  
    BusinessPartnerIsBlocked as isBlocked  
}
```

## Mashups (Combinando Mundos)

Exponha a projeção em seu próprio serviço e delegue as requisições para o serviço remoto através de um handler customizado.

### Exemplo de Handler (Node.js):

```
const bupa = await cds.connect.to('API_BUSINESS_PARTNER');  
this.on('READ', 'Suppliers', req => {  
    return bupa.run(req.query);  
});
```

O CAP 'faz a mágica' de mapear os filtros e seleções da sua projeção (`Suppliers`) para a entidade remota (`A\_BusinessPartner`) e vice-versa.

**Conexão em Produção:** No `package.json`, configure as 'credentials' para usar um 'destination' do BTP no perfil de produção.

```
"[production]": {  
    "credentials": {  
        "destination": "S4HANA",  
        "path": "/sap/opu/odata/sap/API_BUSINESS_PARTNER"  
    }  
}
```

# Etapa 4: A Fachada Inteligente - Servindo UIs Fiori

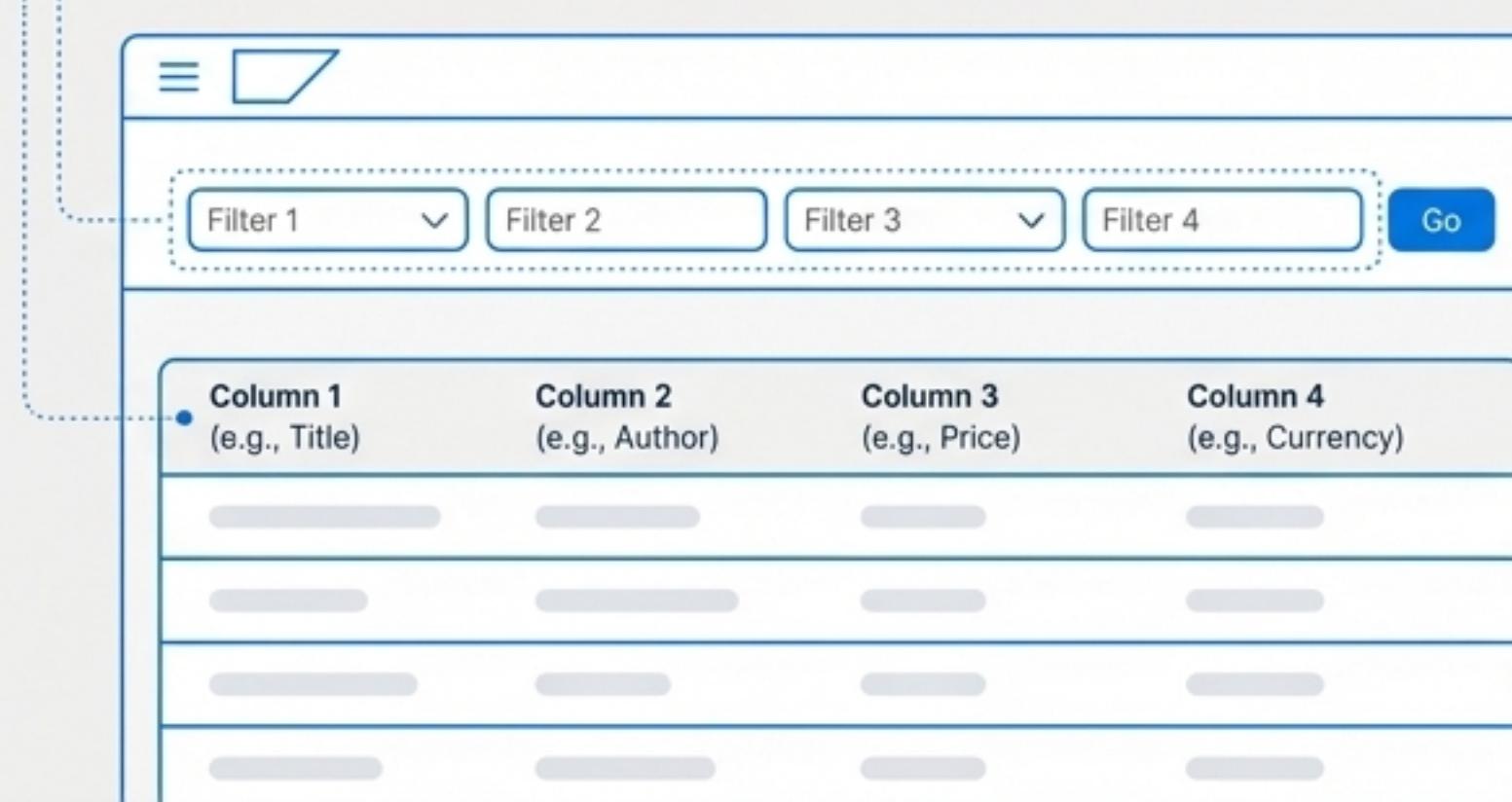
## O Conceito

SAP Fiori Elements são UIs genéricas que se constroem dinamicamente com base nos metadados e anotações do serviço OData. "Metadata-driven UI".

## O Papel das Anotações

Elas adicionam semântica ao seu serviço, dizendo à UI como apresentar os dados.

- `@UI.LineItem`: Define as colunas de uma tabela.
- `@UI.SelectionFields`: Define os campos de filtro.
- `@UI.HeaderInfo`: Define o título de uma página de objeto.



## Exemplo de Anotação

```
annotate CatalogService.Books with @(
    UI: {
        LineItem: [
            { Value: title },
            { Value: author.name, Label: '{i18n>Author}' },
            { Value: price },
            { Value: currency.symbol, Label: ' ' }
        ]
    }
);
```

## Melhor Prática: Separação de Responsabilidades

Mantenha as anotações de UI em arquivos `cds` separados, dentro da pasta `app/`, para não poluir o modelo de domínio.

## Fiori Preview

Durante o desenvolvimento, o CAP oferece um link de 'Fiori Preview' na página inicial, permitindo visualizar o efeito das anotações instantaneamente.

# Criando Experiências de Usuário Superiores com Drafts e Value Helps

## Seção 1: Suporte a Rascunhos (Draft Support)

**Problema:** Permitir que usuários editem dados complexos sem travar o registro ativo e com a possibilidade de cancelar ou continuar depois.

**Solução CAP:** A anotação @odata.draft.enabled.

```
annotate AdminService.Books with @odata.draft.enabled;
```

**Como funciona:** O CAP gerencia todo o ciclo de vida do rascunho (criação, edição, ativação/SAVE), armazenando as alterações em tabelas de rascunho separadas até que sejam ativadas. Isso é transparente para a UI e para a maior parte do código do desenvolvedor.

The screenshot shows a Fiori-style edit screen for a book entry. At the top, it displays the title "Wuthering Heights" and the author "Emily Brontë". Below the title, there are two input fields: "Stock:" and "Title:". At the bottom of the screen are two buttons: "Save" and "Cancel". Above the input fields, there is a small orange circle with a white dot, indicating that the data is currently a draft.

## Seção 2: Ajuda de Entrada de Valores (Value Helps)

**Problema:** Fornecer listas de seleção (F4 Help) para campos associados (ex: escolher uma moeda de uma lista).

**Solução CAP:** A anotação de conveniência @cds.odata.valuelist.

```
// No @sap/cds/common, a entidade CodeList já possui esta anotação  
annotate sap.common.CodeList with @cds.odata.valuelist;  
  
// Portanto, ao usar...  
using { Currency } from '@sap/cds/common';  
entity Books { currency : Currency; } // ...o Value Help é gerado automaticamente
```

The diagram illustrates the integration of a value help feature. On the left, a simple input field labeled "Currency" contains the text "GBP". An arrow points from this field to a larger, detailed view on the right. This view is titled "Select Currency" and contains a search bar. Below the search bar is a table with four rows, each representing a currency code and its description: "USD" (United States Dollar), "EUR" (Euro), "BRL" (Brazilian Real), and "GBP" (British Pound Sterling). The row for "GBP" is highlighted with a blue background.

Currency	Description
USD	United States Dollar
EUR	Euro
BRL	Brazilian Real
GBP	British Pound Sterling

# Etapa 5: A Fortaleza - Segurança por Design



## Seguro por Padrão (Secure by Default)

O CAP parte do princípio de que tudo é protegido. O desenvolvedor deve explicitamente liberar o acesso, não o contrário. Uma aplicação CAP vinculada a uma instância **XSUAA** autentica **todos** os endpoints por padrão.

### Autenticação (Quem é você?)

O CAP se integra nativamente com o serviço de identidade da plataforma (XSUAA).

A configuração é simplificada pelo comando `cds add xsuaa`.

Para UIs, o **Application Router** gerencia a sessão e o token OAuth2, mas a autenticação no backend do CAP ainda é mandatória.



### Autorização (O que você pode fazer?)

As regras de autorização são declaradas diretamente no modelo CDS usando anotações.

```
entity Books @restrict: [
  { grant: 'READ', to: 'authenticated-user' },
  { grant: 'CREATE', to: 'content-maintainer' },
  { grant: 'UPDATE', to: 'content-maintainer' },
  { grant: 'DELETE', to: 'admin' },
] { ... }
```

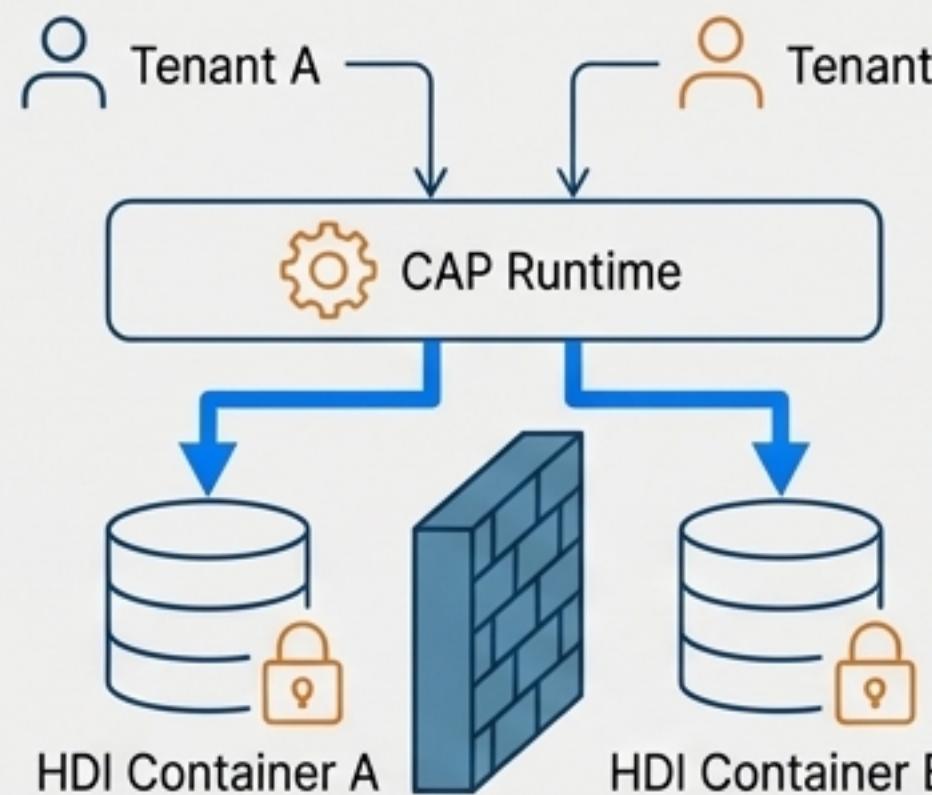
Isso mantém as regras de negócio e de segurança juntas, de forma clara e legível.

# Segurança Avançada: Multi-Tenancy e Proteção contra Ameaças

## Isolamento de Dados em Multi-Tenancy

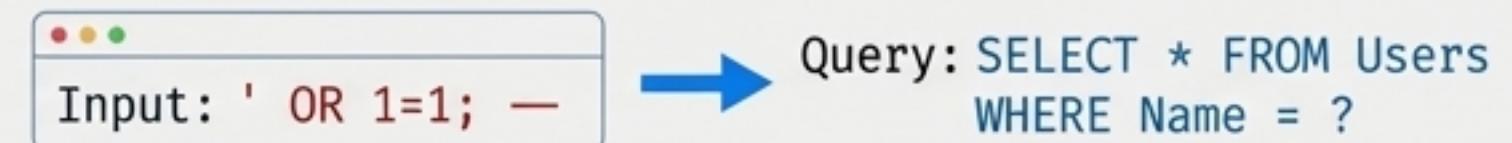
O CAP foi projetado desde o início para suportar multi-tenancy.

- **Dados Persistentes:** Para cada tenant, o CAP provisiona um **container HDI isolado** no SAP HANA Cloud. Isso significa schemas de banco de dados e usuários técnicos separados, garantindo que um tenant nunca possa acessar os dados de outro.
- **Dados Transientes:** O runtime do CAP utiliza `cds.context` para garantir que os dados em memória de uma requisição não vazem para outra.



## Proteção Contra Entradas Maliciosas (Untrusted Input)

- **SQL Injection:** O motor de query do CAP (CQN) é imune a SQL Injection por padrão, pois transforma as queries em *prepared statements*.



⚠️ **Atenção:** A proteção se aplica aos valores dos parâmetros. Se a estrutura da query (nomes de entidades ou colunas) for construída dinamicamente, a validação manual ainda é necessária.

- **Cross-Site Request Forgery (CSRF):** O Application Router gerencia a proteção de token CSRF (`x-csrf-token`) por padrão. Os serviços CAP, sendo stateless, não precisam se preocupar com isso.

# Etapa 6: O Motor Otimizado - Modelagem de Performance

Uma modelagem inadequada pode levar a gargalos de performance severos no banco de dados, mesmo com poucas linhas de código.

## Abuso de `UNION`

UNION em views tem uma penalidade de performance e complexifica o modelo. Frequentemente usado para emular polimorfismo.



### Exemplo 'Ruim':

```
// Evitar! Não performático.
view FruitsByVendor as
    select from Apples UNION
    select from Bananas UNION
    select from Cherries
    { ID, description, vendor }
    where vendor.description = 'TopFruitCompany';
```

cds



## Campos Calculados na Leitura (On Read)

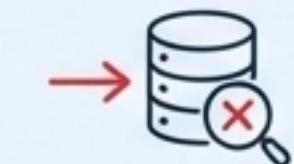
Operações de banco de dados em campos calculados (`CASE`, concatenação) **não podem usar índices**, resultando em *full table scans*.



### Exemplo 'Ruim':

```
// Evitar! Causa full table scan.
entity OrdersItemsView as projection on OrdersItems {
    *,
    case
        when quantity > 500 then 'Large'
        when quantity > 100 then 'Medium'
        else 'Small'
    end as category : String
};
```

cds



**Cuidado ao portar modelos legados!** Padrões que eram necessários em sistemas antigos são anti-padrões de performance em arquiteturas modernas.

# Padrões de Performance em CAP: A Abordagem Inteligente

## Solução para `UNION`: Polimorfismo com Associações

Em vez de `UNION`, modele uma entidade "interface" e use associações para os tipos específicos. O acesso aos dados detalhados é feito sob demanda.

### Exemplo

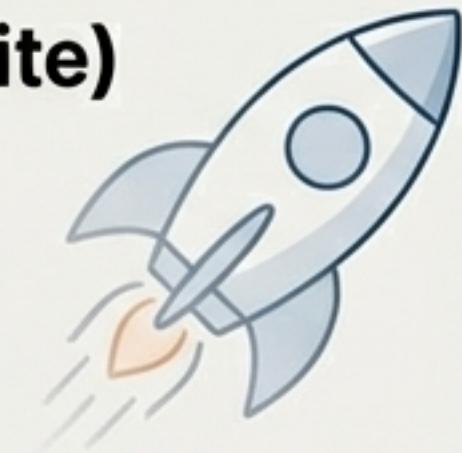
```
// Modelagem polimórfica correta  
entity Fruit : cuid, managed {  
    type : String enum {  
        apple; banana; ...  
    };  
    description : String;  
    // ... associações para detalhes específicos  
}
```



cds

## Solução para Campos Calculados: Pré-calcular na Escrita (On Write)

A melhor otimização é calcular o valor uma vez (na escrita) em vez de muitas vezes (em cada leitura).



### Exemplo

```
// Ótimo! O campo é armazenado e pode ser indexado.  
extend my.OrdersItems with {  
    category: String = case  
        when quantity > 500 then 'Large'  
        when quantity > 100 then 'Medium'  
        else 'Small'  
    end stored;  
}
```

**Dica para Filtros e Ordenação:** Sempre filtre/ordene na tabela 'menor' (itens) antes de fazer o JOIN com a tabela 'maior' (cabeçalho). Use associações para permitir que o otimizador de query faça isso.

# Etapa 7: Garantia de Qualidade - Testes Automatizados

O CAP fornece uma biblioteca de teste poderosa, `@cap-js/cds-test`, para simplificar a escrita de testes de integração.

## Configuração Simples

- Instale com `npm add -D @cap-js/cds-test`.
- Em seu arquivo de teste, inicie o servidor CAP com uma única linha:

```
const cds = require('@sap/cds')
const test = cds.test(__dirname+'/.') // Inicia o servidor
```

### \* Testando APIs de Serviço (Programaticamente)\*

Acesse os serviços diretamente via `cds.connect.to` e use a API de query.

```
it('Testa a API de serviço', async () => {
  const AdminService = await cds.connect.to('AdminService');
  const { Authors } = AdminService.entities;
  const authors = await AdminService.read(Authors);
  expect(authors).to.have.lengthOf(3); // Exemplo com Chai
});
```



### \*Testando APIs HTTP\*

Use os helpers `GET`, `POST`, etc., que já conhecem a URL do servidor de teste.

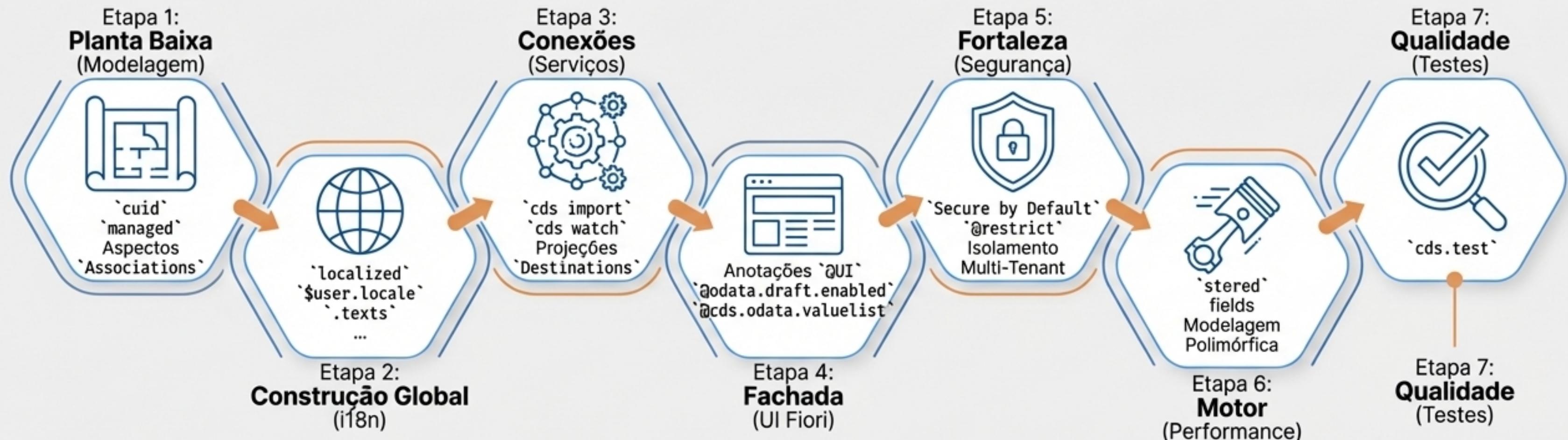
```
it('Testa a API HTTP com usuário mock', async () => {
  const { data } = await test.GET('/browse/Books');
  expect(data.value).to.not.be.empty;

  // Testando endpoints autenticados
  const { status } = await test.POST('/admin/Authors',
    { name: 'Novo Autor' },
    { auth: { username: 'alice', password: '' } }
  );
  expect(status).to.equal(201);
});
```



O `cds.test` abstrai a complexidade de iniciar/parar o servidor e fazer requisições, permitindo que você foque na lógica do teste.

# A Jornada Completa: O Caminho CAP para a Excelência



Seguindo a 'Jornada do Construtor', o SAP CAP não é apenas um conjunto de ferramentas, mas um guia opinativo que nos leva a construir aplicações empresariais que são, por design:

- Robustas e Manuteníveis
- Globais e Acessíveis
- Conectadas e Extensíveis
- Intuitivas e Produtivas
- Seguras e Escaláveis
- Performáticas e Eficientes

**Construa da maneira CAP. Construa para o futuro.**