



# MEGA ACADEMY: FIORI FULL STACK

---

**INSTRUTORA: EDISLAINE GODOY SILVA**

## Sumário

1. HISTÓRICO DE REVISÃO.....	4
2. INTRODUÇÃO AO SAP, ECC E S/4HANA.....	5
3. JAVASCRIPT AVANÇADO .....	7
3.1. HISTÓRIA E EVOLUÇÃO DO JAVASCRIPT .....	7
3.2. FUNDAMENTOS DA LINGUAGEM JAVASCRIPT .....	8
3.3. ESCOPO, HOISTING E CLOSURES .....	9
3.4. PROGRAMAÇÃO ASSÍNCRONA: CALLBACKS, PROMISES E ASYNC/AWAIT .....	10
3.5. PADRÕES E BOAS PRÁTICAS EM JAVASCRIPT.....	10
4. SAPUI5: FRAMEWORK DE DESENVOLVIMENTO FIORI .....	10
4.1. O QUE É SAPUI5? .....	10
4.2. ARQUITETURA DO SAPUI5 .....	11
4.3. CONTROLES E COMPONENTES UI.....	11
4.4. DEMOKIT E DOCUMENTAÇÃO OFICIAL.....	12
4.5. MODELOS DE DADOS E DATA BINDING .....	13
4.6. BOAS PRÁTICAS NO DESENVOLVIMENTO COM SAPUI5 .....	13
4.7. INTERNACIONALIZAÇÃO (I18N) .....	14
4.8. ROTEAMENTO E NAVEGAÇÃO ENTRE TELAS.....	14
5. SAP BUSINESS TECHNOLOGY PLATFORM (SAP BTP) .....	15
5.1. VISÃO GERAL DA PLATAFORMA.....	15
5.2. DATABASE & DATA MANAGEMENT.....	16
5.3. ANALYTICS.....	17
5.4. APPLICATION DEVELOPMENT & AUTOMATION .....	17
5.5. INTEGRATION .....	18
5.6. SEGURANÇA, IDENTIDADE E GOVERNANÇA .....	18
5.7. BENEFÍCIOS .....	18
5.8. SAP DISCOVERY CENTER .....	18
6. ODATA V2 E V4: PROTOCOLOS DE INTEGRAÇÃO EM SAP .....	19
6.1. O QUE É ODATA?.....	19
6.2. ODATA V2 VS ODATA V4.....	20
6.3. EXEMPLO DE SERVIÇO ODATA V2 (SAP GATEWAY) .....	20
6.4. EXEMPLO DE SERVIÇO ODATA V4 (CAP/RAP) .....	20

6.5. DEEP ENTITY E NAVEGAÇÃO .....	21
6.6. ANOTAÇÕES E METADADOS PARA FIORI .....	21
7. CDS VIEWS E RESTFUL APPLICATION PROGRAMMING MODEL (RAP).....	21
7.1. O QUE SÃO CDS VIEWS?.....	22
7.2. EXEMPLO BÁSICO DE CDS VIEW .....	22
7.3. O QUE É O SAP RAP? .....	22
7.4. TUTORIAL INTRODUTÓRIO: CRIANDO UMA APLICAÇÃO RAP BÁSICA .....	22
7.5. TIPOS DE CDS VIEWS E BOAS PRÁTICAS .....	23
7.6. MANAGED VS UNMANAGED RAP .....	24
7.7. ASSOCIAÇÕES ENTRE ENTIDADES E VIEWS COMPOSTAS .....	24
7.8. ANOTAÇÕES UI E FIORI ELEMENTS .....	24
7.9. PRINCIPAIS ANOTAÇÕES CDS (ANNOTATIONS).....	25

## 1. HISTÓRICO DE REVISÃO

Revisão	Data	Descrição	Responsável
00	01/06/2025	Criação do documento	Edislaine Godoy Silva

## 2. INTRODUÇÃO AO SAP, ECC E S/4HANA



O SAP (**S**ystems, **A**pplications, and **P**roducts in **D**ata **P**rocessing) é um dos maiores fornecedores mundiais de software empresarial para gestão de processos.

Desde sua fundação na Alemanha em 1972, a SAP se destacou ao oferecer soluções capazes de integrar diferentes áreas de uma organização – de finanças à logística, de RH à manufatura – em uma única plataforma, proporcionando uma visão holística e estratégica do negócio.

Na década de 1990, o SAP R/3 trouxe uma revolução ao introduzir uma arquitetura cliente-servidor em três camadas (apresentação, aplicação e banco de dados), além de permitir uma execução modular das funções de negócios.

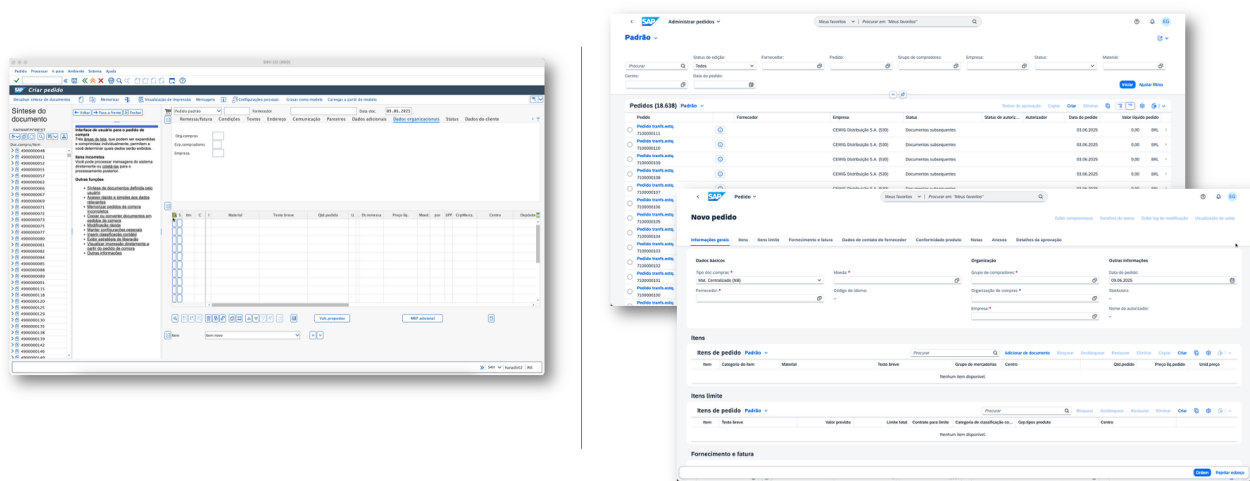
Essa solução foi sucedida pelo SAP **ECC (ERP Central Component)**, uma versão mais robusta e moderna que consolidava os módulos funcionais já existentes com melhorias tecnológicas significativas. O ECC dominou o mercado de ERPs por muitos anos, sendo adotado pelas maiores empresas do mundo.

Contudo, com a evolução tecnológica e o surgimento de novas demandas — como a necessidade de análises em tempo real, mobilidade, cloud computing e interfaces amigáveis — **a SAP lançou o S/4HANA (SAP Business Suite 4 SAP HANA)**, que representa não apenas uma atualização técnica, mas uma reformulação completa da plataforma.



O **S/4HANA** foi desenvolvido para rodar exclusivamente sobre o banco de dados em memória SAP HANA, eliminando redundâncias e otimizando o desempenho das operações e análises simultaneamente. O SAP HANA permite processar grandes volumes de dados transacionais e analíticos em tempo real, reduzindo drasticamente o tempo de resposta das aplicações e simplificando a modelagem dos dados. Com isso, processos que antes exigiam múltiplos steps e tabelas intermediárias foram eliminados. A arquitetura do S/4HANA é mais enxuta, com menos código legado, e oferece performance otimizada para o mundo digital.

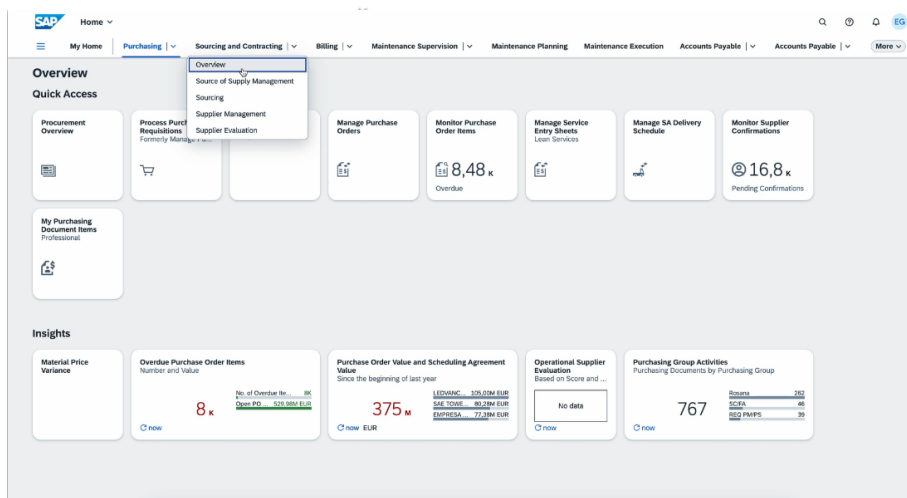
Em paralelo à mudança tecnológica, a SAP também introduziu uma transformação na experiência do usuário (UX) com o SAP Fiori. **O SAP Fiori é a linguagem de design e conjunto de tecnologias voltadas à construção de aplicações com foco no usuário, responsividade, simplicidade e coerência visual.** Em contraste com a interface tradicional SAP GUI, o Fiori apresenta uma interface intuitiva baseada em tiles, cards e navegação moderna, acessível em desktops, tablets e celulares.



SAP GUI vs SAP FIORI

O SAP Fiori utiliza o framework SAPUI5, baseado em HTML5, JavaScript e CSS3, possibilitando o desenvolvimento de aplicações modernas e integradas. Essas aplicações acessam os dados do backend através de serviços OData expostos por meio do SAP Gateway, o que garante uma arquitetura baseada em APIs RESTful. Essa abordagem Full Stack torna o desenvolvedor SAP moderno responsável tanto pelo front-end (UI5) quanto pelo consumo e estruturação de dados (CDS Views, OData, BAPIs, entre outros).

A introdução do SAP Fiori também trouxe consigo o conceito de Fiori Launchpad, uma central de acesso onde os usuários encontram seus aplicativos organizados por funções de negócio. Cada tile representa um aplicativo e pode exibir dados em tempo real, alertas ou indicadores de performance (KPIs). Com isso, o ambiente SAP tornou-se mais proativo e orientado à ação.



Do ponto de vista de arquitetura, o SAP pode ser implantado de diversas formas: totalmente on-premise, em cloud privada ou pública, ou de forma híbrida.

O **SAP BTP (Business Technology Platform)** complementa essa arquitetura ao fornecer serviços de integração, extensão, automação e inteligência artificial para aplicações SAP e não-SAP. A tendência atual é a adoção crescente do SAP S/4HANA Cloud junto com o SAP Fiori e os serviços do BTP, promovendo agilidade, inovação contínua e redução de custos operacionais.

Outro ponto importante é o ciclo de vida dos sistemas SAP. O SAP ECC tem suporte garantido apenas até 2027 (com extensão paga até 2030), e a SAP recomenda fortemente a migração para o S/4HANA. Essa transição, embora desafiadora, representa uma grande oportunidade para modernizar processos, padronizar práticas e preparar a empresa para o futuro digital.

Ao longo desta apostila, você aprenderá não apenas os fundamentos técnicos de JavaScript e SAPUI5, mas também os conceitos de integração via OData, construção de modelos de dados com CDS Views, e desenvolvimento de APIs modernas com o SAP Restful Application Programming Model (RAP). Essa base permitirá que você atue de forma completa em projetos SAP modernos, entregando valor ao negócio e experiências de usuário de excelência.

### 3. JAVASCRIPT AVANÇADO

#### 3.1. HISTÓRIA E EVOLUÇÃO DO JAVASCRIPT



**JavaScript** foi criado em 1995 por Brendan Eich, inicialmente com o nome de Mocha e depois LiveScript, sendo finalmente lançado como JavaScript. Ele foi concebido para rodar no navegador Netscape Navigator com o objetivo de adicionar interatividade às páginas HTML. Com o tempo, evoluiu para uma linguagem de programação poderosa e universal, sendo adotada como uma das principais linguagens da web moderna.

Atualmente, **JavaScript é uma linguagem multiparadigma** (suporta programação funcional, orientada a objetos e imperativa), e com a chegada do Node.js, passou a ser usada também no lado do servidor. Em SAP Fiori, JavaScript é a base para o desenvolvimento com o framework SAPUI5.

### 3.2. FUNDAMENTOS DA LINGUAGEM JAVASCRIPT

JavaScript é uma linguagem interpretada, de tipagem dinâmica e orientada a eventos. Alguns conceitos fundamentais incluem:

- **Tipos de dados primitivos:** string, number, boolean, null, undefined, symbol e bigint
- **Variáveis:** declaradas com var, let e const
- **Operadores:** aritméticos, lógicos, relacionais, ternário
- **Estruturas de controle:** if, switch, for, while, do...while
- **Funções:** funções declaradas, expressões de função e arrow functions
- **Objetos:** estruturas chave-valor dinâmicas e mutáveis
- **Arrays:** listas ordenadas com métodos de alta ordem como map, filter, reduce

#### Declaração de variáveis (var, let, const):

```
var nome = 'João';  
let idade = 30;  
const PI = 3.1415;
```

#### Condicional if/else:

```
if (idade >= 18) {  
    console.log('Maior de idade');  
} else {  
    console.log('Menor de idade');  
}
```

#### Laço for:

```
for (let i = 0; i < 5; i++) {  
  console.log('Número:', i);  
}
```

#### Função tradicional e arrow function:

```
function saudacao(nome) {  
  return 'Olá ' + nome;  
}
```

```
const saudacaoArrow = (nome) => 'Olá ' + nome;
```

#### Objeto e acesso a propriedades:

```
const pessoa = {  
  nome: 'Ana',  
  idade: 25  
};
```

```
console.log(pessoa.nome);
```

#### Array e métodos comuns:

```
const frutas = ['maçã', 'banana', 'laranja'];  
frutas.forEach(fruta => console.log(fruta));
```

#### Exemplo com função callback:

```
function processar(dado, callback) {  
  console.log('Processando:', dado);  
  callback();  
}
```

```
processar('dados', () => console.log('Finalizado!'));
```

### 3.3. ESCOPO, HOISTING E CLOSURES

O escopo define a visibilidade das variáveis. Em JavaScript, temos escopo global, de função e de bloco. Com `var`, o escopo é de função; com `let` e `const`, o escopo é de bloco.

**Hoisting** é o comportamento de mover declarações para o topo do escopo antes da execução do código. Variáveis declaradas com `var` e funções declaradas sofrem hoisting.

**Closures** ocorrem quando uma função interna acessa variáveis da função externa mesmo após esta ter sido executada. É um conceito essencial para programação funcional e encapsulamento de dados.

### 3.4. PROGRAMAÇÃO ASSÍNCRONA: CALLBACKS, PROMISES E ASYNC/AWAIT

Por natureza, JavaScript é single-threaded e orientado a eventos. A programação assíncrona permite executar tarefas sem bloquear o fluxo principal de execução.

- **Callbacks:** funções passadas como argumento para serem executadas após uma tarefa
- **Promises:** objetos que representam o resultado eventual de uma operação assíncrona (pending, fulfilled, rejected)
- **Async/Await:** sintaxe moderna para trabalhar com Promises de forma mais legível e estruturada

### 3.5. PADRÕES E BOAS PRÁTICAS EM JAVASCRIPT

Escrever código limpo e manutenível é fundamental em qualquer projeto profissional. Algumas boas práticas incluem:

- Usar const e let no lugar de var
- Modularizar o código com funções pequenas e reutilizáveis
- Aplicar o princípio DRY (Don't Repeat Yourself)
- Usar arrow functions para preservar o contexto de this
- Evitar manipulações diretas do DOM quando possível
- Utilizar ESLint e formatação automática com Prettier

No contexto SAPUI5, é essencial manter um padrão de estrutura de arquivos, seguir as convenções da arquitetura MVC e aplicar boas práticas de encapsulamento e coesão.

## 4. SAPUI5: FRAMEWORK DE DESENVOLVIMENTO FIORI

### 4.1. O QUE É SAPUI5?

**SAPUI5** é um framework de desenvolvimento baseado em JavaScript, criado pela SAP, voltado para a construção de interfaces de usuário ricas, responsivas e adaptáveis. Ele é a base técnica para o desenvolvimento das aplicações SAP Fiori. O SAPUI5 permite a criação de aplicações web corporativas com alto nível de interatividade e usabilidade, seguindo os padrões de design modernos e adaptáveis a diversos dispositivos (desktop, tablet e smartphone).

Além disso, é um framework open source, com a versão OpenUI5 disponível publicamente. Ele inclui uma vasta biblioteca de controles UI, modelos de dados, gerenciamento de rotas, MVC (Model-View-Controller), internacionalização e suporte a temas customizáveis.

## 4.2. ARQUITETURA DO SAPUI5

A arquitetura do SAPUI5 é baseada no padrão MVC, onde cada componente da aplicação possui sua função bem definida:

- **Model:** camada responsável pelos dados da aplicação (JSON, XML, OData, etc.)
- **View:** camada de interface do usuário (pode ser XML, HTML, JSON ou JS)
- **Controller:** lógica de manipulação de eventos e integração entre view e model

O SAPUI5 também utiliza uma estrutura modular baseada em RequireJS, permitindo o carregamento assíncrono de bibliotecas e componentes. Essa abordagem melhora a performance e a escalabilidade das aplicações.

Exemplo de estrutura de pasta de um projeto SAPUI5:

```
├── webapp/
│   ├── controller/
│   │   └── Main.controller.js
│   ├── view/
│   │   └── Main.view.xml
│   ├── i18n/
│   │   └── i18n.properties
│   ├── model/
│   │   └── models.js
│   └── Component.js
```

## 4.3. CONTROLES E COMPONENTES UI

O SAPUI5 possui uma vasta biblioteca de componentes visuais (UI Controls), como:

- **sap.m.Input:** campo de entrada de texto
- **sap.m.Button:** botão com eventos
- **sap.m.Table:** tabela responsiva
- **sap.m.Dialog:** pop-up para confirmação ou alertas
- **sap.ui.layout.form.SimpleForm:** layout de formulários

Esses componentes são altamente configuráveis e permitem ligação direta com os modelos de dados, usando data binding. Eles também seguem as diretrizes de acessibilidade.

Exemplo de botão com evento no controller:

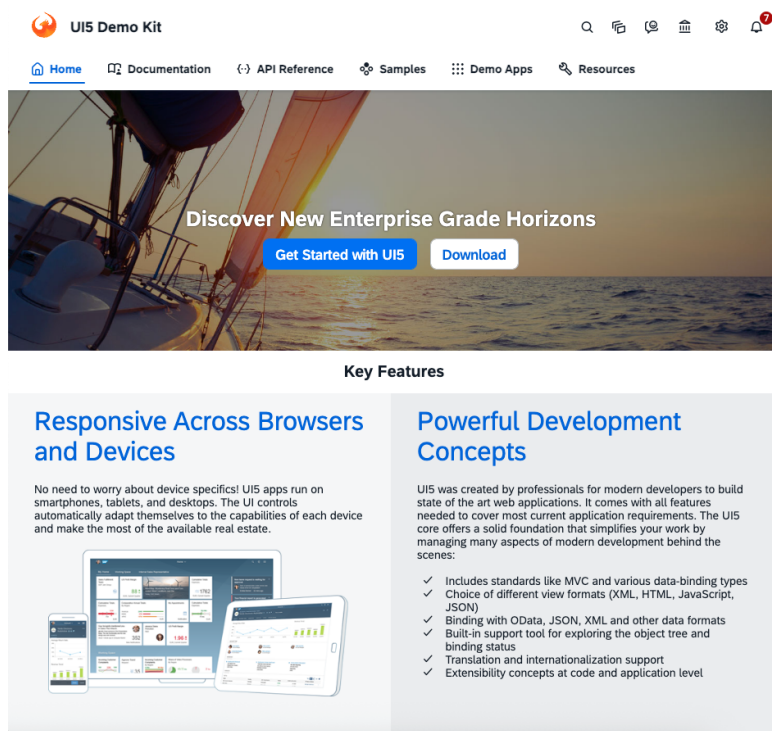
```
<Button text="Clique Aqui" press="onPressBotao"/>
```

```
// Controller.js
```

```
onPressBotao: function() {
    sap.m.MessageToast.show("Você clicou!");
}
```

#### 4.4. DEMOKIT E DOCUMENTAÇÃO OFICIAL

O SAPUI5 DemoKit é o portal oficial de documentação do framework. Ele inclui exemplos interativos, documentação técnica, especificações de API, guias de layout e práticas recomendadas.



O site **DemoKit** é uma ferramenta essencial para desenvolvedores, permitindo:

- Visualizar comportamento de controles
- Consultar propriedades, eventos e métodos
- Acessar exemplos de código real
- Baixar templates de aplicações

**URL:** <https://sdk.openui5.org> ou <https://sapui5.hana.ondemand.com>

Exemplo de consulta a API de um controle (sap.m.Input):

```
https://sapui5.hana.ondemand.com/#/api/sap.m.Input
// Propriedades:
- value (string)
```

- placeholder (string)
- enabled (boolean)

#### 4.5. MODELOS DE DADOS E DATA BINDING

SAPUI5 suporta vários tipos de modelos de dados:

- **JSON Model:** ideal para dados locais e mockados
- **XMLModel:** útil em algumas integrações estruturadas
- **ODataModel:** para consumo de serviços OData v2/v4

O data binding é o processo de vincular dados do modelo com os controles da interface. SAPUI5 oferece binding unidirecional, bidirecional e por expressão.

Exemplo de binding com JSONModel:

```
// No Controller
var oModel = new sap.ui.model.json.JSONModel({ nome: 'Carlos'
});
this.getView().setModel(oModel);

// Na View
<Input value="{nome}" />
```

#### 4.6. BOAS PRÁTICAS NO DESENVOLVIMENTO COM SAPUI5

No desenvolvimento com SAPUI5, seguir boas práticas é essencial para garantir a manutenibilidade, escalabilidade e legibilidade do código. Dentre as recomendações mais relevantes, destacam-se:

- Separar claramente as responsabilidades entre view, controller e model
- Usar nomes significativos para arquivos e funções
- Aplicar modularização e reutilização de componentes (fragments, custom controls)
- Evitar lógica de negócio no controller; manter foco na manipulação de UI
- Documentar funções com comentários explicativos
- Utilizar o console do navegador para depuração (debug, breakpoints)
- Adotar convenções como CamelCase para nomes de variáveis e métodos

Exemplo de fragment reutilizável:

```
// Fragment XML: Message.fragment.xml
<core:FragmentDefinition xmlns="sap.m" xmlns:core="sap.ui.core">
  <Dialog title="Mensagem">
    <Text text="{mensagem}" />
    <beginButton>
```

```
<Button text="Fechar" press=".onFechar"/>
</beginButton>
</Dialog>
</core:FragmentDefinition>
```

#### 4.7. INTERNACIONALIZAÇÃO (i18n)

Internacionalização (i18n) é o processo de adaptar a aplicação para diferentes idiomas e regiões. Em SAPUI5, isso é feito através de arquivos de propriedades armazenados no diretório `i18n`, onde cada chave corresponde a um texto traduzível. O framework carrega automaticamente o idioma com base nas configurações do navegador.

Exemplo de uso do i18n:

```
// i18n.properties
```

```
bemvindo=Bem-vindo
```

```
// XML View
```

```
<Text text="{i18n>bemvindo}" />
```

Carregamento do i18n manualmente:

```
var i18nModel = new sap.ui.model.resource.ResourceModel({
    bundleName: 'nome.do.projeto.i18n.i18n'
});
this.getView().setModel(i18nModel, 'i18n');
```

#### 4.8. ROTEAMENTO E NAVEGAÇÃO ENTRE TELAS

SAPUI5 oferece um mecanismo robusto de roteamento baseado no arquivo `manifest.json`, permitindo a navegação declarativa entre views. O roteador é inicializado automaticamente pelo Component.js e permite navegação com parâmetros e controle de histórico.

Exemplo de definição de rota no manifest.json:

```
"routing": {
  "config": {
    "routerClass": "sap.m.routing.Router",
    "viewType": "XML",
    "controllId": "app",
    "async": true,
    "controlAggregation": "pages"
  },
  "routes": [
    {
```

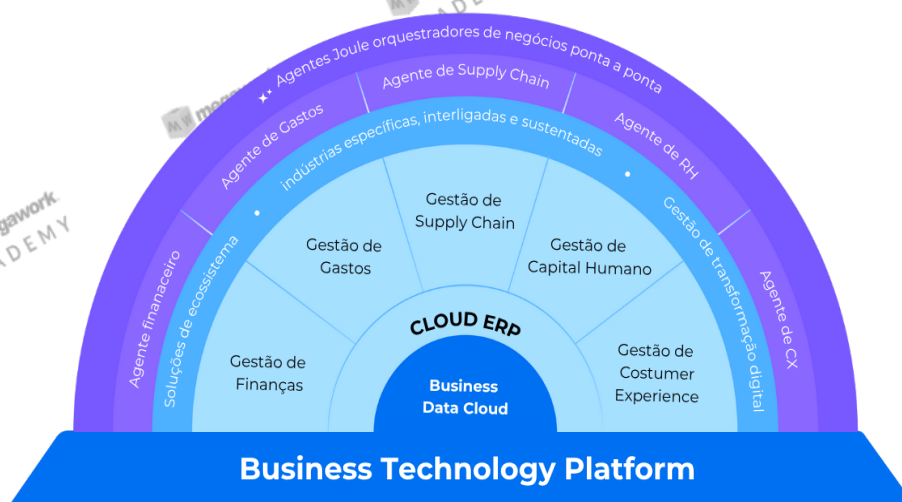
```

    "pattern": "",
    "name": "home",
    "target": "home"
  }
],
"targets": {
  "home": {
    "viewName": "Home"
  }
}
}
}

//Navegação no controller:
this.getOwnerComponent().getRouter().navTo('home');
```

## 5. SAP BUSINESS TECHNOLOGY PLATFORM (SAP BTP)

A SAP Business Technology Platform (SAP BTP) é a plataforma de tecnologia da SAP que unifica banco de dados, gerenciamento de dados, analytics, integração, automação e desenvolvimento de aplicações em uma única fundação baseada em nuvem. Ela serve como base para inovação, extensibilidade e inteligência em soluções SAP e não-SAP, oferecendo serviços e ferramentas modernas que aceleram a transformação digital das empresas.



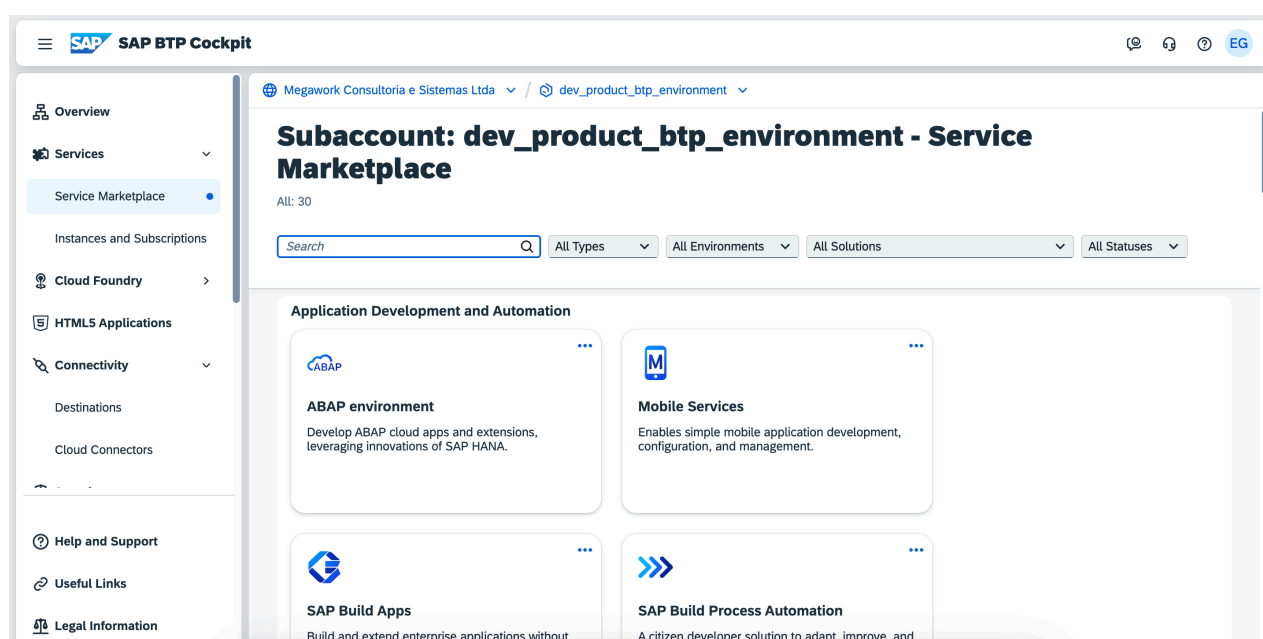
### 5.1. VISÃO GERAL DA PLATAFORMA

**SAP BTP** é o alicerce tecnológico da estratégia de Intelligent Enterprise da SAP. A plataforma conecta processos, dados e experiências em ambientes multicloud, com segurança, escalabilidade e performance. Está disponível nas principais provedoras de nuvem (AWS, Microsoft Azure, Google Cloud Platform e Alibaba Cloud).

Ela está estruturada em quatro grandes pilares funcionais:

- Database & Data Management
- Analytics
- Application Development & Automation
- Integration

Além disso, o SAP BTP oferece capacidades transversais como segurança, identidade, observabilidade e marketplace de serviços (SAP Discovery Center).



## 5.2. DATABASE & DATA MANAGEMENT

Este pilar foca no armazenamento, processamento, governança e virtualização de dados. Permite consolidar dados distribuídos em múltiplas fontes com governança e alta performance.

Principais serviços e soluções:

- **SAP HANA Cloud:** banco de dados in-memory com processamento analítico e transacional (HTAP)
- **SAP Datasphere:** camada semântica de modelagem de dados corporativos
- **SAP Master Data Governance:** gerenciamento e centralização de dados mestres
- **SAP Data Intelligence:** orquestração e preparação de dados, com conectores diversos (on-premise/cloud)
- **SAP Adaptive Server Enterprise (ASE) e SAP IQ:** bancos legados com foco em alta performance e armazenamento analítico

### 5.3. ANALYTICS

A camada de Analytics do SAP BTP é responsável por fornecer insights de dados em tempo real com dashboards, KPIs e integração com fontes SAP e externas. Ela empodera o usuário de negócio a tomar decisões baseadas em dados.

Principais ferramentas e serviços:

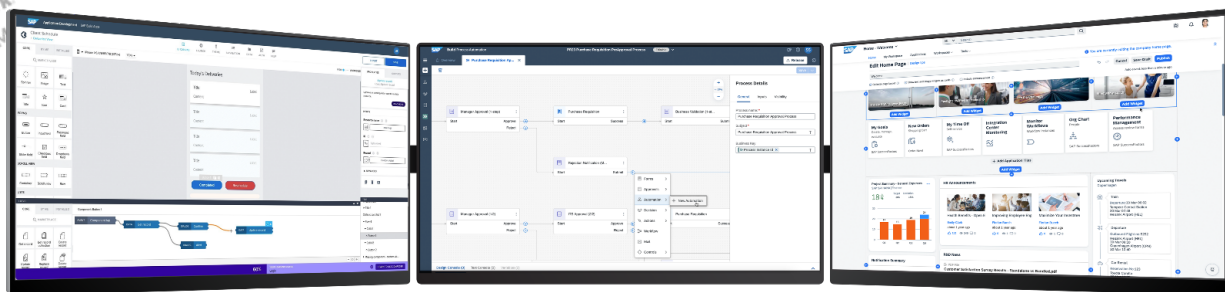
- **SAP Analytics Cloud (SAC):** ferramenta de BI unificada com dashboards, planejamento e previsões
- **SAP BW/4HANA:** data warehouse para grandes volumes com alta performance
- **SAP Data Warehouse Cloud:** solução cloud para modelagem analítica self-service
- **SAP BusinessObjects:** suíte tradicional de relatórios corporativos

### 5.4. APPLICATION DEVELOPMENT & AUTOMATION

Esta camada viabiliza a criação, extensão e automação de aplicações SAP e não-SAP, com ferramentas low-code/no-code, pro-code e serviços de inteligência artificial. Permite que tanto desenvolvedores quanto analistas de negócio criem soluções rapidamente.

Principais serviços:

- **SAP Build Apps:** desenvolvimento visual low-code para web/mobile apps
- **SAP Build Process Automation:** automação de processos com workflows, bots e formulários
- **SAP Build Work Zone:** criação de portais e páginas empresariais
- **SAP Business Application Studio:** IDE moderna baseada em VS Code
- **SAP Cloud Application Programming Model (CAP):** framework para desenvolvimento pro-code com Node.js/Java
- **SAP Mobile Services:** backend para aplicativos móveis integrados ao SAP



## 5.5. INTEGRATION

O SAP BTP provê uma infraestrutura robusta de integração, conectando sistemas SAP e não-SAP de forma segura e padronizada. Suporta APIs REST, OData, conectores prontos, eventos e pipelines.

Principais componentes:

- **SAP Integration Suite:** plataforma unificada para integração (API, eventos, B2B, EDI)
- **SAP API Management:** gerenciamento, monetização e segurança de APIs
- **SAP Event Mesh:** comunicação orientada a eventos entre aplicações
- **Open Connectors:** conectores prontos para Salesforce, Google, Slack, entre outros
- **SAP Graph:** acesso unificado a APIs de dados empresariais

## 5.6. SEGURANÇA, IDENTIDADE E GOVERNANÇA

A segurança é uma camada transversal no SAP BTP, com foco em identidade, autenticação, controle de acesso, criptografia e conformidade com padrões internacionais.

Principais serviços:

- **SAP Identity Authentication Service (IAS):** autenticação centralizada e SSO
- **SAP Identity Provisioning Service (IPS):** provisionamento de usuários entre sistemas
- **XSUAA:** serviço de autorização baseado em roles para apps em Cloud Foundry
- **Audit Logging e Logging Service:** monitoramento e rastreabilidade de ações

## 5.7. BENEFÍCIOS

Entre os principais benefícios da SAP BTP estão:

- Inovação ágil e extensível
- Conectividade nativa com SAP S/4HANA, SuccessFactors, Ariba, etc.
- Multicloud com liberdade de escolha
- Redução de custos operacionais com soluções low-code e automação
- Segurança e compliance empresarial

## 5.8. SAP DISCOVERY CENTER

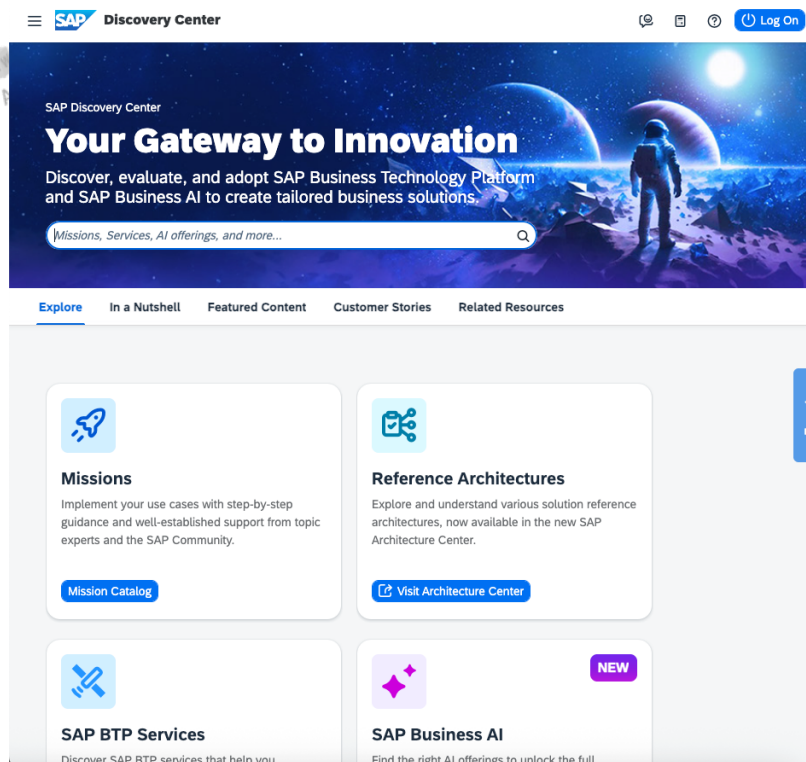
O SAP Discovery Center é um portal que reúne projetos prontos, boosters, tutoriais e serviços da BTP. Ideal para acelerar a adoção e validar protótipos de inovação.

Link: <https://discovery-center.cloud.sap>

Permite:

- Explorar cenários prontos com fluxos guiados
- Avaliar serviços disponíveis em cada datacenter
- Estimar custos por serviço

- Acessar documentação e exemplos reais



## 6. ODATA V2 E V4: PROTOCOLOS DE INTEGRAÇÃO EM SAP

### 6.1. O QUE É ODATA?

OData (Open Data Protocol) é um protocolo baseado em REST que permite a criação e consumo de APIs de forma padronizada. Ele é mantido pela OASIS e suportado pela SAP como principal protocolo de exposição de dados para aplicações Fiori e integrações SAPUI5. OData facilita operações CRUD (Create, Read, Update, Delete) via HTTP e suporta query strings para filtros, ordenação, paginação e expansão de entidades relacionadas.

Um serviço OData é composto por:

- **Entity Types:** definem os objetos e seus atributos (como uma tabela)
- **Entity Sets:** coleções de entidades (como registros de uma tabela)
- **Navigation Properties:** definem os relacionamentos
- **Metadata Document:** arquivo XML que descreve toda a estrutura do serviço

## 6.2. ODATA V2 VS ODATA V4

A SAP adotou amplamente o OData v2 como padrão para aplicações Fiori baseadas no SAP Gateway. Com o tempo, o OData v4 passou a ser suportado nativamente no CAP e nos serviços RAP.

Comparativo resumido:

Recurso	OData v2	OData v4
Formato de metadados	XML	XML e JSON
Suporte SAP Gateway CAP/RAP)	Sim	Limitado (somente
Batch Requests eficiente)	Sim	Sim (mais
Contagem de registros	inlinecount	\$count
Suporte JSON moderno	Limitado	Nativo
Deep Insert	Sim	Sim (com melhorias)
Expansão (\$expand) aninhamento)	Sim	Sim (com

## 6.3. EXEMPLO DE SERVIÇO ODATA V2 (SAP GATEWAY)

Um serviço OData v2 é comumente criado com a transação **\*\*SEGW\*\*** no SAP Gateway. Abaixo um exemplo de como acessar os dados e aplicar filtros:

```
GET /sap/opu/odata/sap/ZORDERS_SRV/OrderSet
GET /sap/opu/odata/sap/ZORDERS_SRV/OrderSet?$filter=Customer eq '1000001'
GET /sap/opu/odata/sap/ZORDERS_SRV/OrderSet?$top=5&$orderby=Date desc
```

As operações CRUD são feitas usando os métodos HTTP:

- **GET** para leitura
- **POST** para criação
- **PUT/MERGE** para atualização
- **DELETE** para exclusão

## 6.4. EXEMPLO DE SERVIÇO ODATA V4 (CAP/RAP)

Em projetos modernos com CAP (Node.js/Java) ou RAP (ABAP RESTful), o padrão OData v4 é adotado. A estrutura é definida com anotações e exposta via cds ou behavior definitions.

```
GET /odata/v4/catalog/Products?$filter=price gt
50&$orderby=createdAt desc&$count=true
POST /odata/v4/catalog/Products
{ "ID": 101, "name": "Notebook", "price": 3200 }
```

O OData v4 permite também:

- Encadeamento de \$expand com \$filter internos
- Retorno parcial com \$select
- Melhor uso de headers HTTP para controle de cache e resposta
- \$apply para agregações com groupby e aggregates (em SAC, por exemplo)

## 6.5. DEEP ENTITY E NAVEGAÇÃO

Deep Insert (ou Deep Entity) permite criar entidades compostas com subentidades em uma única chamada. Isso é comum em ordens com itens, cabeçalho com linhas, etc.

```
POST /OrderSet
{
  "OrderID": "5001",
  "Customer": "20002",
  "Items": [
    { "ProductID": "X1", "Quantity": 2 },
    { "ProductID": "Y1", "Quantity": 5 }
  ]
}
```

## 6.6. ANOTAÇÕES E METADADOS PARA FIORI

As anotações OData ajudam a definir como os dados devem ser apresentados no front-end, especialmente em aplicações Fiori Elements. Elas podem ser embutidas no serviço (inline) ou separadas (external annotations).

```
@UI: {
  lineItem: [
    { position: 10, type: #AS_DATAPOINT, value: 'TotalValue' },
    { position: 20, label: 'Status', value: 'OrderStatus' }
  ]
}
```

## 7. CDS VIEWS E RESTFUL APPLICATION PROGRAMMING MODEL (RAP)

## 7.1. O QUE SÃO CDS VIEWS?

CDS (Core Data Services) Views são artefatos de modelagem de dados usados no SAP S/4HANA para expor dados estruturados de forma semântica. Elas substituem as views clássicas do ABAP Dictionary e são amplamente utilizadas para consumo em SAP Fiori, análises e APIs OData.

Benefícios das CDS Views:

- Modelagem declarativa com sintaxe SQL-like
- Suporte a anotações UI, analíticas e de OData
- Integração com authorizations via @AccessControl
- Performance aprimorada com uso de SAP HANA

## 7.2. EXEMPLO BÁSICO DE CDS VIEW

```
@AbapCatalog.sqlViewName: 'ZV_PEDIDOS'
@AccessControl.authorizationCheck: #NOT_REQUIRED
define view ZI_Pedidos as select from vbak
{
  vbeln as Pedido,
  erdat as DataCriacao,
  vkorg as OrganizacaoVendas,
  netwr as ValorTotal
}
```

## 7.3. O QUE É O SAP RAP?

O SAP RAP (Restful Application Programming Model) é um modelo de desenvolvimento moderno baseado em ABAP que permite criar aplicações de forma eficiente, reutilizando CDS Views, serviços OData v4 e lógica de negócio encapsulada.

Ele é composto por 3 camadas:

- Modelagem de dados: CDS Views e associations
- Definição de comportamento: Behaviour Definitions (BO Behavior)
- Exposição de serviço: Service Definitions e Bindings

## 7.4. TUTORIAL INTRODUTÓRIO: CRIANDO UMA APLICAÇÃO RAP BÁSICA

Este tutorial mostra como criar uma aplicação simples de pedidos com SAP RAP.

### Passo 1: Criar CDS View com dados do pedido

```
define root view entity ZI_Pedido as select from zpedido
{
  key pedido_id,
  cliente,
```

```
    valor_total,  
    status  
}
```

### Passo 2: Definir o Comportamento (Behavior Definition)

```
define behavior for ZI_Pedido alias Pedido  
persistent table zpedido  
lock master  
authorization master ( instance )  
{  
    create;  
    update;  
    delete;  
}
```

### Passo 3: Criar Service Definition

```
define service ZUI_PedidoService {  
    expose ZI_Pedido;  
}
```

### Passo 4: Criar Service Binding para OData v4

No Eclipse, clique com o botão direito sobre o 'Service Definition', escolha 'New Service Binding', selecione OData v4 UI e publique o serviço.

### Passo 5: Testar no Fiori Elements (Preview)

Ao abrir o binding, clique em 'Preview' para visualizar automaticamente a aplicação gerada com Fiori Elements, com funcionalidades de Create, Read, Update e Delete (CRUD).

## 7.5. TIPOS DE CDS VIEWS E BOAS PRÁTICAS

Existem diversos tipos de CDS Views no contexto SAP, cada uma com finalidades específicas:

- **Basic Views** refletem diretamente uma tabela do banco de dados
- **Composite Views:** agregam ou combinam múltiplas basic views
- **Consumption Views:** preparadas para consumo externo (Fiori, SAC, APIs)

Boas práticas incluem reutilizar basic views e separar responsabilidades entre visualização, lógica e consumo.

## 7.6. MANAGED VS UNMANAGED RAP

O SAP RAP suporta dois modos de implementação de comportamento:

- **Managed:** a infraestrutura gerencia a persistência (CRUD automático); você escreve apenas lógica customizada
- **Unmanaged:** você controla toda a lógica de persistência por meio de classes ABAP (sem CRUD automático)

Em projetos modernos com tabelas novas, recomenda-se o uso do modelo **Managed** pela simplicidade e integração com Fiori Elements.

## 7.7. ASSOCIAÇÕES ENTRE ENTIDADES E VIEWS COMPOSTAS

Associações (associations) permitem definir relacionamentos entre views, similares a joins. São utilizadas para navegação, expand de OData, e composição visual no Fiori.

Exemplo: Uma view de pedidos pode se associar a uma view de clientes.

```
define view entity ZI_Pedido with parameters p_ano:
  abap.char(4) as select from zpedido
  association [0..1] to ZI_Cliente as _Cliente on
  $projection.cliente_id = _Cliente.id
{
  key pedido_id,
  cliente_id,
  _Cliente.nome
}
```

## 7.8. ANOTAÇÕES UI E FIORI ELEMENTS

As anotações UI tornam possível a geração automática de interfaces com Fiori Elements. São escritas diretamente nas CDS Views e orientam a exibição de campos, ações, status, filtros, etc.

Exemplo de uso com @UI:

```
@UI: {
  headerInfo: {
    typeName: 'Pedido',
    title: { value: 'pedido_id' },
    description: { value: 'status' }
  },
  lineItem: [
    { position: 10, value: 'cliente_id' },
    { position: 20, value: 'valor_total' }
  ]
}
```

As views anotadas com `@UI.lineItem` e `@UI.selectionField` são interpretadas automaticamente por Fiori Elements, permitindo criar aplicativos empresariais com mínimo esforço de UI manual.

## 7.9. PRINCIPAIS ANOTAÇÕES CDS (ANNOTATIONS)

As anotações (annotations) em CDS Views permitem estender a semântica dos dados, controlando seu comportamento em consumo, exibição, segurança, análise e integração. As principais categorias são:

- `@UI.` → anotações de interface para Fiori Elements
- `@OData.` → exposição e propriedades de serviços OData
- `@Analytics.` → funcionalidades analíticas como cubos e KPIs
- `@AccessControl.` → controle de acesso baseado em roles

### @UI Annotations (Interface)

São utilizadas para gerar automaticamente a interface do Fiori Elements.

```
@UI.lineItem: [  
  { position: 10, value: 'cliente_id', label: 'Cliente' },  
  { position: 20, value: 'valor_total', label: 'Total',  
    criticality: 'status' }  
]
```

- `@UI.lineItem`: define colunas de uma tabela
- `@UI.selectionField`: habilita campo como filtro de pesquisa
- `@UI.identification`: mostra campos em detalhes de objeto
- `@UI.headerInfo`: define cabeçalho da entidade na UI

### @OData Annotations

Usadas para controlar exposição, navegação e nome de entidades no serviço OData.

```
@OData.publish: true  
@OData.entitySet.name: 'Pedidos'
```

- `@OData.publish: true`: publica a view como serviço OData automaticamente
- `@OData.entitySet.name`: define nome do conjunto de entidades na URL

### @Analytics Annotations

Permitem transformar views em cubos analíticos, úteis para SAC e dashboards.

```
@Analytics.query: true  
@Analytics.dataCategory: #CUBE
```

- @Analytics.query: marca a view como fonte de dados analítica
- @Analytics.dataCategory: define se é dimensão, fato ou cubo (#DIMENSION, #FACT, #CUBE)