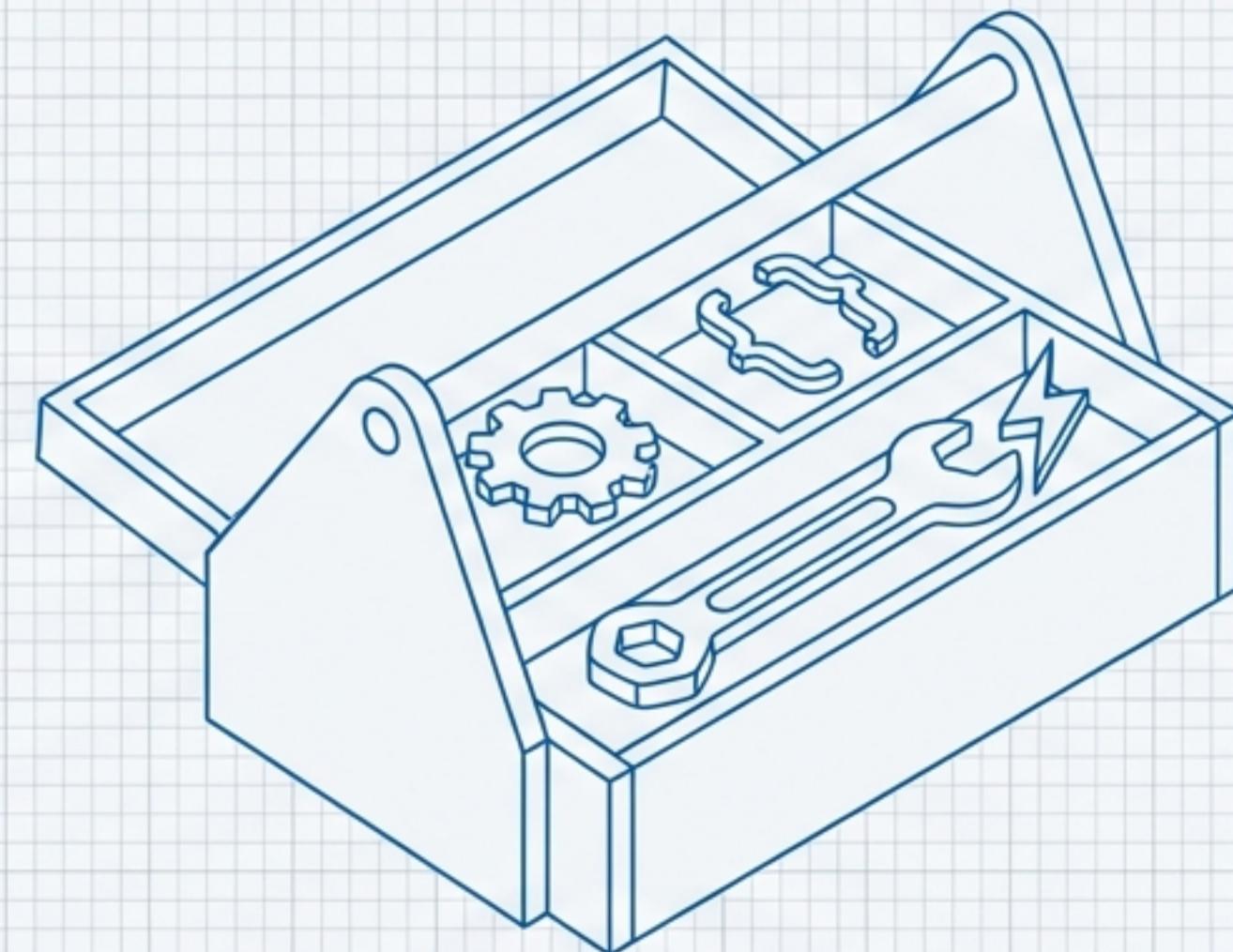


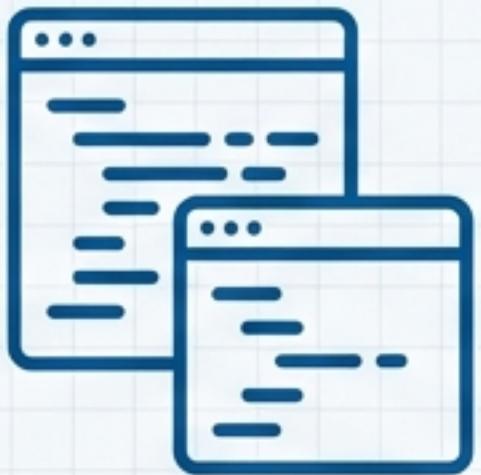
Desvendando `cds.utils`

Sua Caixa de Ferramentas Essencial para o Desenvolvimento CAP Node.js



Vá além do básico e descubra como o módulo **`cds.utils`** pode transformar seu fluxo de trabalho, tornando seu código mais limpo, seguro e expressivo.

Por que `cds.utils`? Simplifique Seu Código.



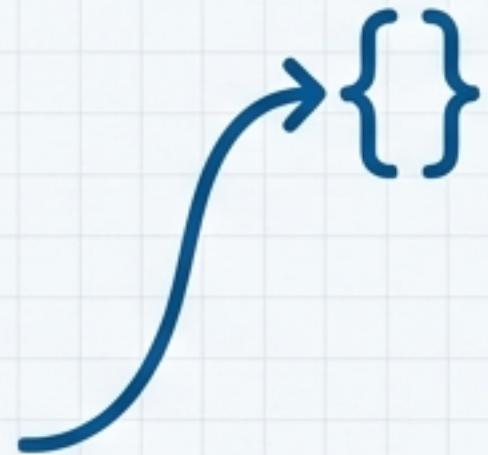
Menos Boilerplate

Chega de `try-catch` para URIs ou `JSON.parse/stringify` manuais. `cds.utils` oferece variantes seguras e automação inteligente.



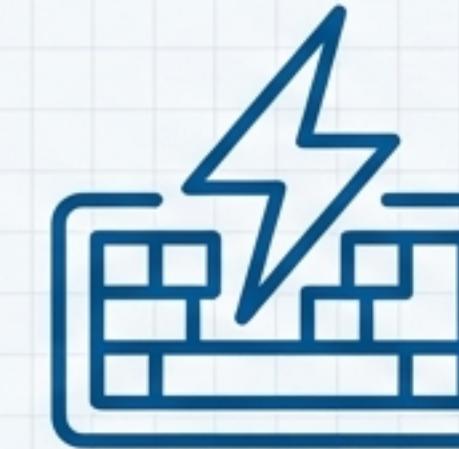
Consciência de Contexto CAP

Funções que entendem a estrutura do seu projeto. Operações de arquivo são resolvidas automaticamente a partir do `cds.root`, eliminando a complexidade do gerenciamento de paths.



API Fluente e Expressiva

Escreva código que é mais fácil de ler e manter com APIs fluentes como `write(...).to(...)` e `copy(...).to(...)`.



Atalhos Inteligentes

Acesso rápido aos módulos essenciais do Node.js (`fs`, `path`, `inspect`) sem a necessidade de `require` explícitos.

Categoria 1: Fundamentos e Identidade

Gerando IDs e Lidando com Dados Externos de Forma Segura

Geração de UUIDs

Identificadores Únicos e Imediatos

```
const { uuid } = cds.utils  
  
let id = uuid() // Gera um novo UUID
```

Decodificação Segura

O Fim do `try-catch` para URLs

****Decodificação Segura**:** As variantes do CAP para `decodeURI` e `decodeURIComponent` são 'seguras'. Em caso de erro, elas retornam a string de entrada original em vez de lançar um `URIError`, simplificando o código.

```
// Abordagem Padrão  
let input = "%E0%A4%A"  
let uri  
try {  
  uri = decodeURI(input)  
} catch {  
  uri = input  
}
```

```
// Com cds.utils  
const { decodeURI } = cds.utils  
  
let uri = decodeURI("%E0%A4%A") // Retorna a entrada se falhar
```



Categoria 2: Navegação e Contexto do Projeto

Garantindo que Seus Caminhos de Arquivo Funcionem em Qualquer Lugar

‘local()’

Caminhos Clicáveis no Terminal: Use `local()` para imprimir nomes de arquivos no console. O VS Code e outros terminais modernos transformarão esses caminhos em links clicáveis, independentemente de onde você executou o comando `cds run`.

The screenshot shows a terminal window with a dark header bar labeled 'VS Code'. The window is divided into two sections by a vertical line. The left section is titled 'Executado da Pasta Pai (`/parent_folder`)' and shows the command '[samples] cds run bookshop' followed by '[cds] - loaded model from 5 file(s):' and two underlined file paths: 'bookshop/srv/cat-service.cds' and 'bookshop/db/schema.cds'. An ellipsis '...' is at the bottom. The right section is titled 'Executado de Dentro do Projeto (`/parent_folder/bookshop`)' and shows the command '[bookshop] cds run' followed by '[cds] - loaded model from 5 file(s):' and two underlined file paths: 'srv/cat-service.cds' and 'db/schema.cds'. Another ellipsis '...' is at the bottom.

`local(filename)` retorna uma representação relativa do arquivo para o diretório de trabalho original do processo (`process.cwd()`), garantindo consistência nos logs.

Categoria 3: Verificando o Sistema de Arquivos

Sondando o Terreno Antes de Agir

****Relativo ao `cds.root**:** Todas as checagens são resolvidas em relação à raiz do seu projeto CAP (`cds.root`), então você não precisa se preocupar com `path.join` ou `__dirname`.

`exists(file)`

Verifica se um arquivo ou pasta existe.

```
const { exists } = cds.utils

if (exists('server.js')) {
  // ... fazer algo
}
```

`isdir(file)`

Verifica se é um diretório e retorna o caminho absoluto resolvido.

```
const { isdir, fs } = cds.utils

let dir = isdir('app')
if (dir) {
  let entries = fs.readdirSync(dir)
  // ...
}
```

`.isfile(file)`

Verifica se é um arquivo e retorna o caminho absoluto resolvido.

```
const { isfile } = cds.utils

let file = isfile('package.json')
if (file) {
  // ...
}
```

Categoria 4: Leitura e Escrita Inteligente de Arquivos

Manipulando Arquivos com Superpoderes

async read(file)

****`JSON.parse()` Automático**:** Se o arquivo lido tiver a extensão `json`, o conteúdo é automaticamente convertido para um objeto JavaScript. A codificação padrão é `utf8`.

```
const { read } = cds.utils
// Retorna um objeto JS, não uma string!
let pkg = await read('package.json')
console.log(pkg.name) // -> 'my-cap-project'
```

async write(data).to(...file)

****`JSON.stringify()` Automático e API Fluente**:** Se o dado fornecido for um objeto, ele é automaticamente serializado para uma string JSON. A API fluente `.to()` torna a escrita mais legível.

```
const { write } = cds.utils
const data = { version: "1.0.0" }
// Ambas as formas são válidas
await write(data).to('some', 'file.json')
await write(data).to('some/file.json')
```

Categoria 5: Construção e Organização de Projetos

Estruturando Diretórios e Copiando Artefatos com Facilidade

****Caminhos como Argumentos**:** Em vez de concatenar strings, passe segmentos do caminho como argumentos separados para maior clareza e segurança.

Criação Recursiva de Diretórios

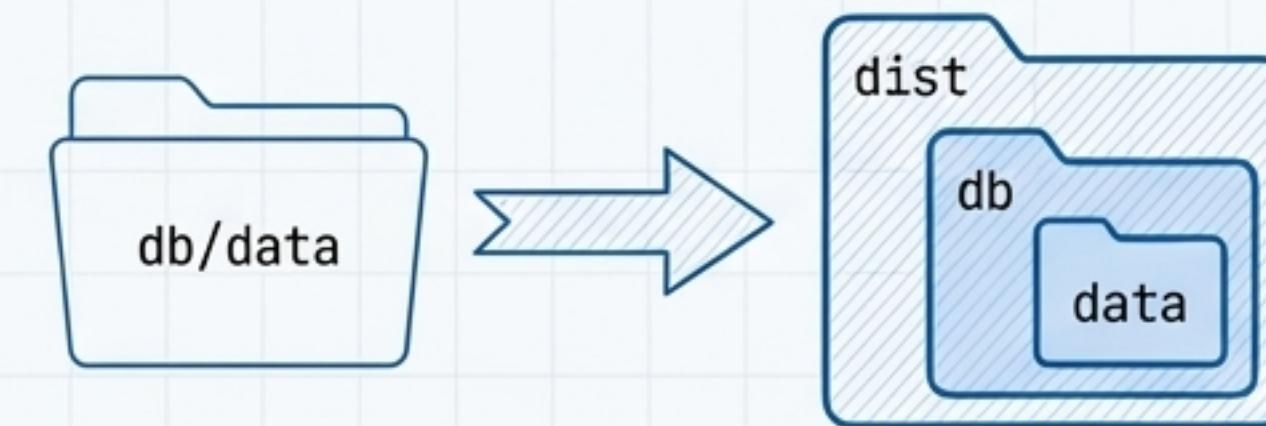
```
async mkdirp(...path)  
  
const { mkdirp } = cds.utils  
  
// Cria a estrutura 'dist/db/data' se não existir  
await mkdirp('dist', 'db', 'data')
```



Nota Técnica: Equivalente a `fs.promises.mkdir(..., { recursive: true })`.

Cópia Inteligente de Arquivos

```
async copy(src).to(...dst)  
  
const { copy } = cds.utils  
  
// Copia o conteúdo de 'db/data' para 'dist/db/data'  
await copy('db/data').to('dist', 'db', 'data')
```



Nota Técnica: Utiliza `fs.promises.cp()` internamente.

Categoria 6: Gerenciamento e Limpeza

Escolhendo a Ferramenta Certa para Remover Arquivos e Diretórios

| Função | O que faz? | Comportamento Principal | Use quando... |
|------------------------------------|--|--|--|
| <code>async rm(...path)</code> | Remove um arquivo . | Utiliza <code>'fs.promises.rm()'</code> . | ...você precisa deletar um arquivo específico. |
| <code>async rmdir(...path)</code> | Remove um diretório recursivamente. | Lança um erro se o diretório não existir. | ...o diretório deve existir e a falha precisa ser reportada. |
| <code>async rimraf(...path)</code> | Remove um diretório recursivamente. | Não faz nada se o diretório não existir. | ...você quer garantir que um diretório não exista (ex: <code>'dist'</code>), sem causar erro em uma execução limpa. |

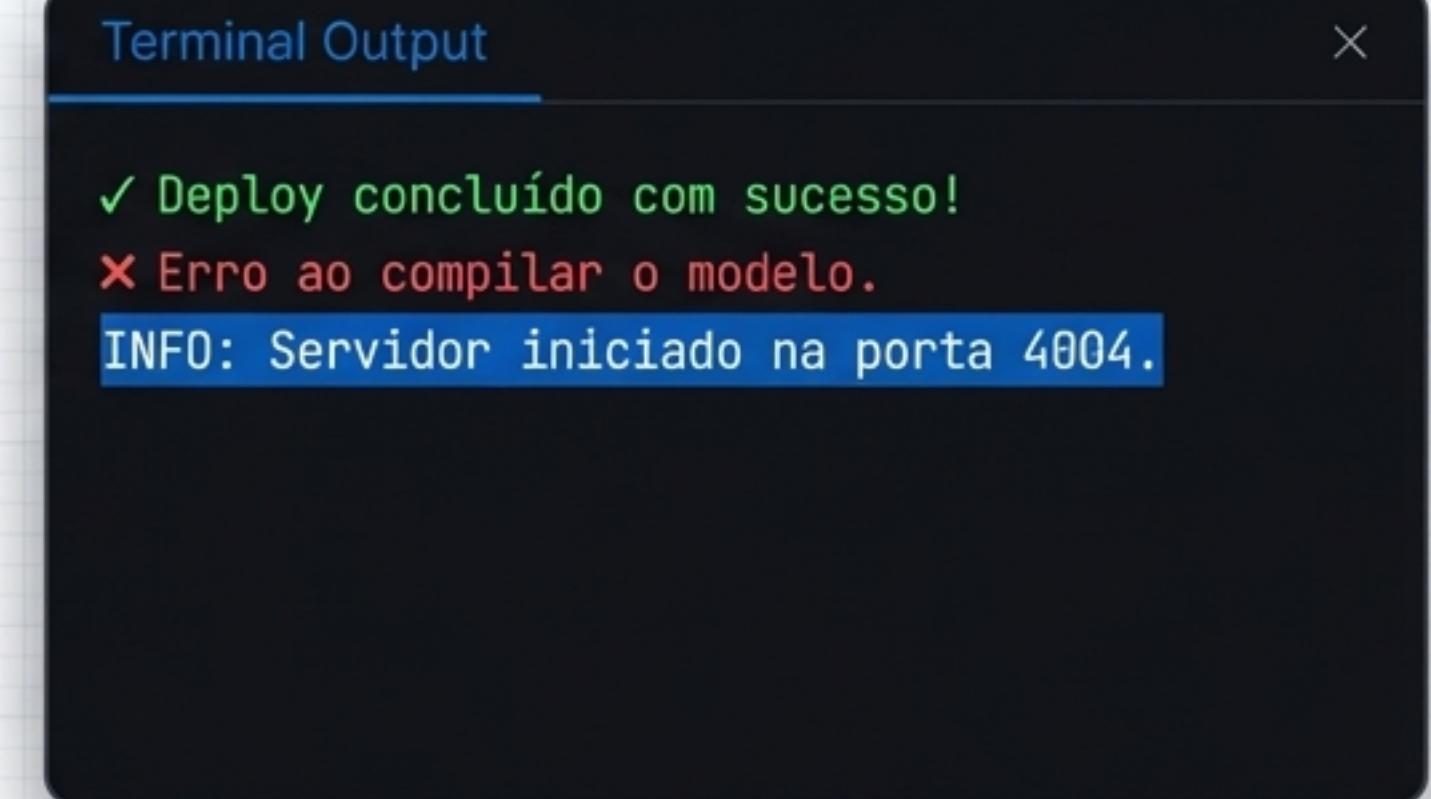
Nota Técnica: `rimraf` utiliza `fs.promises.rm` com as opções { `recursive: true, force: true` }.

Categoria 7: Dando Vida ao Seu Terminal

Melhorando a Experiência do Desenvolvedor com Saídas Coloridas

****Ativação Inteligente**:** As cores são ativadas ou desativadas automaticamente com base no suporte do terminal. Esse comportamento pode ser forçado com as variáveis de ambiente `NO_COLOR` ou `FORCE_COLOR`.

```
const { BRIGHT, RED, GREEN, BLUE, RESET, bg } =  
  cds.utils.colors  
  
console.log(BRIGHT, GREEN, '✓ Deploy concluído com  
sucesso!', RESET)  
console.log(BRIGHT, RED, '✗ Erro ao compilar o  
modelo.', RESET)  
console.log(bg.BLUE, 'INFO: Servidor iniciado na  
porta 4004.', RESET)
```



Categoria 8: Atalhos para os Clássicos

Acesso Direto às Ferramentas Padrão do Node.js

Para máxima conveniência, `cds.utils` fornece atalhos para os módulos e funções mais utilizados do Node.js. Menos `require`, mais produtividade.

| Atalho em `cds.utils` | Equivalente em Node.js |
|-----------------------|-------------------------|
| cds.utils.fs | require('fs') |
| cds.utils.path | require('path') |
| cds.utils.inspect | require('util').inspect |

Exemplo de Uso

```
const { read, fs, path } = cds.utils

// Usando uma função de 'utils'
let pkg = await read('package.json')

// Usando um atalho para um módulo nativo
let absolutePath = path.join(cds.root, 'package.json')
```

Resumo: Sua Caixa de Ferramentas Completa



Fundamentos

uuid, decodeURI



Contexto

local



Verificação

exists, isdir, isfile



Arquivos I/O

read, write



Estrutura

mkdirp, copy



Limpeza

rm, rmdir, rimraf



DX no Terminal

colors



Atalhos Node.js

fs, path, inspect

Ao dominar o kit de ferramentas `cds.utils`, você escreve um código que é:

- ✓ **Mais Limpo:** Menos boilerplate e mais declarativo.
- ✓ **Mais Seguro:** Com funções que lidam com erros de forma inteligente.
- ✓ **Consciente do Contexto:** Sempre operando a partir da raiz do seu projeto.
- ✓ **Mais Produtivo:** Com automação e atalhos para tarefas comuns.

Explore e Construa

A melhor maneira de aprender é fazendo.

Abra seu projeto CAP e refatore uma tarefa comum usando `cds.utils` hoje mesmo! Tente substituir um `fs.readFile` de um JSON pela função `read()` ou simplifique um script de limpeza com `rimraf`.

Recursos

Para consulta detalhada e mais exemplos, acesse a documentação oficial.

cap.cloud.sap/docs/node.js/cds-utils

