# Manual de Boas Práticas

O Cronapp, em seu *Low-Code*, entrega tanto em projetos web quando no mobile, uma arquitetura MVC, RESTful, com suporte a internacionalização e sensível a fusos horários. Apesar de o Cronapp reduzir a complexidade do desenvolvimento, é importante observar as boas práticas no desenvolvimento de aplicações modernas, a fim de deixar seu sistema ainda mais limpo, eficaz, bem programado e rápido. Para isso, criamos esse manual com algumas dicas para que nossos usuários possam seguir.

**Nesta Página**



## Entidades vs Fontes de dados

Aconselhamos sempre o uso de Fontes de dados ao invés de acesso diretamente às entidades. Primeiro, por segurança. Para expor uma entidade, você terá que expor todas (por padrão e simplicidade elas já são expostas). Mesmo que queira expor uma entidade e bloquear outras, você terá um trabalho extra de entidade por entidade, definindo permissões de segurança e isso pode ser impraticável com sistemas grandes. Além disso, a Fonte de dados traz diversos benefícios, como:

- **Restrições de dados**: é possível definir condições à sua consulta a fim de não expor todos os dados no formulário.
  Ex.: `select` com `where`.
- **Uso de blocos como Fonte**: é possível usar blocos para alimentar a Fonte de dados, criando lógicas para entrega os dados. Ex: blocos com `if`s para trazer dados diferentes a partir de uma condição.
- **Uso de eventos**: eventos como antes e depois de inserir, atualizar, remover etc.
- **Validações**: nos eventos é possível validar entrada de dados.
  Ex.: CPF válido ou data em um intervalo.
- **Parâmetros**: é possível usar parâmetros na consulta.
  Ex.: filtrar usuários por uma idade passada no parâmetro.
- **Campos calculados**: é possível adicionar mais campos à Fonte de dados através.
  Ex.: calcular idade e exibir em um campo.
- **Tratamento de erro**: é possível personalizar mensagens de erro.
- **Segurança**: é possível definir segurança de ações, campos e dados (esse pode ser feito com blocos).
- Ex.: esconder campo salário de quem não for do grupo financeiro.

## Criar diagrama de dados

Ao criar um projeto autenticado, seja mobile ou web, ele sempre gera automaticamente o **app**, o diagrama de dados padrão do Cronapp que contém todas as tabelas de segurança da aplicação, como as relacionadas com a permissão de segurança e a de log de auditoria. Por isso, é recomendado criar outros diagramas para separar as tabelas de segurança e, desta forma, melhorar sua organização.

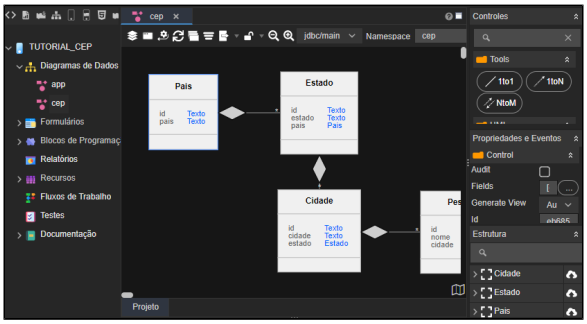


**Figura 1 - Novo diagrama de dados**

## Validações de entrada de dados

As validações de dados **devem** ser feitas nos eventos antes de inserir, remover ou atualizar de uma Fonte de dados **do lado do servidor** (Figura 2). Você pode validar do lado cliente algumas coisas, por questão de performance, mas elas precisam ser revalidadas do lado servidor. **A arquitetura impõe essa boa prática**, visto que traz mais segurança na entrada de dados e mais flexibilidade no uso de diferentes dispositivos alimentando a mesma fonte, como por exemplo, uma página web e um aplicativo, alimentando um cadastro e usando a mesma fonte. As Fontes de dados viram serviços REST, logo não impede que uma pessoa a chame diretamente através de qualquer outro meio. Logo, se sua validação estiver apenas no cliente, ele conseguirá entrar com dados errados no seu sistema.

Os desenvolvedores modernos precisam saber que banco, servidor e cliente não podem se misturar. Uma Fonte de dados é um REST e ela tem "vida" própria e desconhece quem o alimenta.

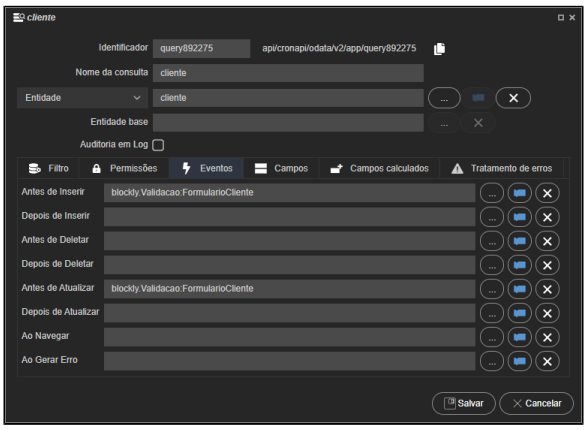


**Figura 2 - Usando eventos para validação de entrada de dados**

# Bloco cliente chamando Bloco servidor

O Cronapp fornece duas funções para um bloco cliente chamar um bloco servidor: a função Chamar Bloco e Chamar Bloco Assíncrono. A função **Chamar Bloco** usa uma especificação Ajax do navegador de bloqueio síncrono que está ameaçada de descontinuidade pelos navegadores modernos há algum tempo já. Logo, **não usem essa função**. Nós a mantemos apenas por compatibilidade com sistemas mais antigos (mas será removida no futuro).

Então, sempre use a função **Chamar Bloco Assíncrono** para esse fim, visto que não bloqueia o navegador e deixa seu sistema mais fluido.

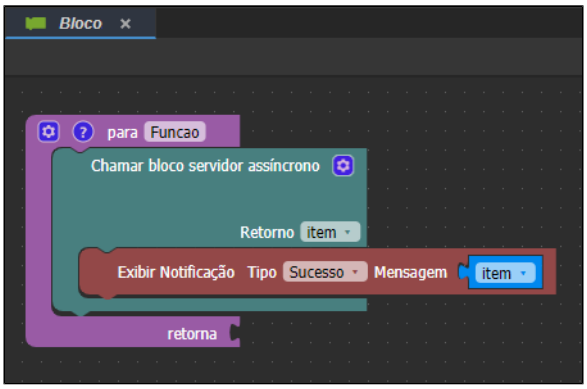


**Figura 3 - Forma correta de chamar um bloco servidor através de um bloco cliente**

# Valores padrões (Iniciais)

As Fontes de dados fornecem um local correto para definir valores padrões para entrada de dados. Muitos clientes usam eventos de cliente "**Ao Entrar**" e modificam os valores dos componentes. Mais uma vez, não aconselhamos isso, pois é inseguro, pelos mesmos argumentos colocados no tópico sobre Validações de Entrada de Dados. Logo, assim como as regras de validação, os valores padrões devem estar do lado do servidor.

**Observação**: Os valores padrões são usados também como valores iniciais de um formulário no Cronapp. Logo, ao clicar em **Novo**, o formulário será alimentado com os valores padrões definidos. Tais valores podem ser valores fixos ou valores dinâmicos, obtidos através de blocos (Figura 4).

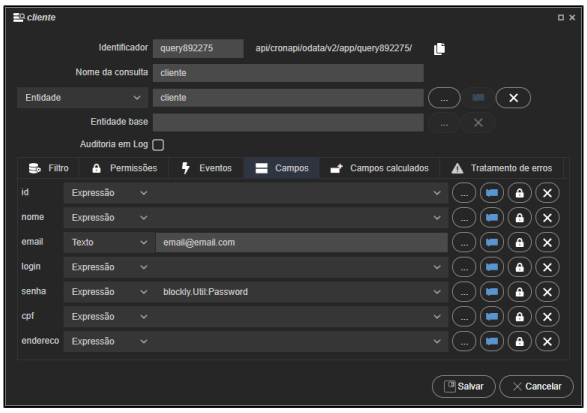


**Figura 4 - Definição dos valores iniciais dos atributos de uma entidade pela Fonte de dados**

# Padrões para nomes

Definir padrões e usar bons termos para nomear os recursos de um projeto é de suma importância para as futuras manutenções do sistema. Apesar de o Cronapp permitir, em alguns locais, nomear itens com letras acentuadas, espaços ou caracteres especiais, aconselhamos fortemente não utilizá-los e seguir alguns padrões de nomenclatura, melhorando a legibilidade do sistema.

Os padrões apresentados aqui seguem algumas regras já bem definidas na programação, como:

- **CamelCase**, esse padrão não contém espaços, sublinhado/*underline* "_", traços "-", caractere especial e o primeiro caractere de cada palavra fica em maiúsculo. Esse padrão se divide em 2 subpadrões:
  - **UpperCamelCase**: primeira letra maiúscula da primeira palavra (também conhecido como PascalCase).
    Exemplo: `UserLoginCount`, `CalcularFolhaDePagamento.`
  - **lowerCamelCase**: primeira letra minúscula da primeira palavra.
    Exemplo: `nomePrincipal`, `dataDeCriacao`,
- **Snake_case**, esse padrão combina palavras substituindo os espaços entre elas por um sublinhado/*underline* "_" e não contém caracteres especiais. Esse padrão se divide em 2 subpadrões:
  - **snake_lowercase**: todas as letras são apresentadas apenas em caixa baixa.
    Exemplo: `numero_usuarios_logados`.
  - **SNAKE_UPPERCASE**: todas as letras são apresentadas apenas em caixa alta.
    Exemplo: `USR_USUARIOS`
- **kebab-case**, esse padrão combina palavras substituindo os espaços entre elas por um traço "-" e não contém caracteres especiais.
  Exemplo: nota-fiscal-25864.pdf

Para mais detalhes sobre os padrões apresentados, recomendamos a leitura do artigo Case Styles: Camel, Pascal, Snake, and Kebab Case.

## Locais e seus padrões

Lembramos que os padrões listados aqui são apenas **sugestões**. O seu uso pode variar de projeto para projeto e depende de diversos fatores, como a equipe que vai trabalhar já estar acostumada com determinados padrões ou o projeto que será integrado a um sistema legado já possuir seu próprio padrão ou diversos outros motivos.

| Local | Padrão | Exemplos |
|---|---|---|
| **Blocos** | **UpperCamelCase** e não usar códigos ou siglas | `ValidarData`, `CalcularFolhaPagamento` |
| **Blocos - funções** | **UpperCamelCase** e não usar códigos ou siglas | `Calcular`, `ObterDataAtual` |
| **Blocos - variáveis** | **lowerCamelCase** | `listaUsuarios, somatorio` |
| **Arquivos Comuns** (xml, txt, json etc) | **kebab-case** | `meus-dados.xml`, `propriedades.txt`, `dados.json` |

| | | |
|---|---|---|
| **Formulários *** | **kebab-case** | `usuarios.view.html`, `meus-clientes.view.html` |
| **Formulários - identificador** | **kebab-case** | `input-primeiro-nome, lista-usuario` |
| **Formulários - valor / ng-model** | **lowerCamelCase** | `idUsuario, ultimoNome` |
| **Entidade** | **UpperCamelCase** e não usar plural | `Empresa`, `Usuario`, `Departamento` |
| **Atributo de Entidade** | **lowerCamelCase** | `id`, `nomePrincipal`, `dataDeCriacao`, `cpf`, `cnpj` |
| **Banco de dados - Tabela** | **SNAKE_UPPERCASE** | `APPLICATION_USER, ROLE_SECURABLE` |
| **Relatórios** | **kebab-case** | `lista-de-usuarios.report`, `folha .report` |
| **Pacotes ou Pastas** | tudo em minúsculo, sem caracteres especiais ou espaços. | `meusrelatorios`, `financeiro`, `dado sclientes` |
| **BPMN (Fluxo de trabalho) - arquivo** | **kebab-case** | `carrinho-compras, avaliacao-projeto` |
| **BPMN (Fluxo de trabalho) - identificador elementos** | **UpperCamelCase** | `IntermediateThrowEvent, EndEvent` |
| **Fonte de dados - identificador** | **lowerCamelCase** | `userRoleManager, userSecurableManager` |
| **Fonte de dados - nome da consulta** | frase curta que represente a consulta | `Gerenciador de Permissionáveis de Usuários, Lista de Usuários` |

* Além de seguir o padrão kebab-case nos formulários, recomendamos a leitura do artigo Keep a simple URL structure.

# Uso de Blocos clientes

É bem comum eventos de tela interagirem com a própria tela. Exemplo: ao selecionar tipo CPF ou CNPJ, o sistema mostra um campo ou outro da tela. Devido à abstração que o Cronapp traz, muitos usuários não se atentam em usar a camada correta para cada caso. O exemplo dado pode ser executado tanto no cliente quanto no servidor, mas é mais indicado o uso do cliente, evitando desperdício de recursos com uma chamada ao servidor com algo que pode ser resolvido no cliente.

# Organização de blocos em pacotes (pastas)

Devido à velocidade de desenvolvimento, muitos usuários deixam de lado a organização estrutural de seus blocos. É comum vermos listas longas de blocos em uma mesma pasta. Aconselhamos separar blocos em pastas (pacotes) e subpastas (subpacotes) por responsabilidade a fim de evitar um código fonte confuso e desorganizado. Exemplo de pacotes: `financeiro`, `administrativo`, `util`, `utildata` .

Outros exemplos de organização que facilitam a manutenção do código:

- Para validações de uma visão no cliente, crie um bloco cliente com o nome da visão, e coloque nesse bloco todas as funções de validação. Nesse bloco pode-se também incluir funções de interação com o usuário, tais como mostrar/esconder uma Modal, habilitar ou não um botão etc.
- Para validações ou outras ações em atributos de uma entidade, crie um bloco com o nome da entidade e funções como `PreInsert`, `PreUpdate` etc.

# Uso de Funções

O Cronapp permite a criação de várias funções dentro de um mesmo bloco. É comum vermos usuários criando uma função por bloco, sendo que aquela função só faz sentido para um determinado bloco. Outros casos comuns que vemos são blocos muito longos com pouco reaproveitamento. Tentem reaproveitar mais seus blocos e funções em outros blocos. Não criem blocos muito longos, pois dificulta a leitura e manutenção. Para rotinas comuns, criem blocos utilitários, exemplo, blocos de manipulação de datas, blocos de validação de dados etc.

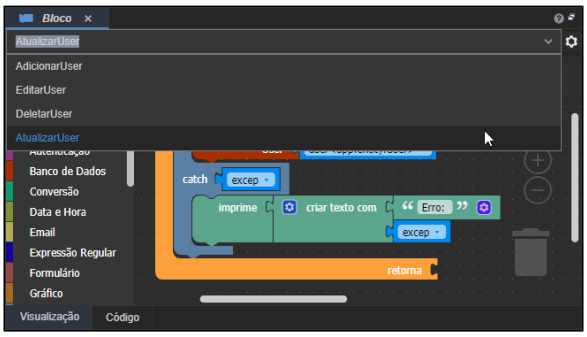


**Figura 5 - Exiba as funções do bloco na caixa de seleção superior**

# Adicionar comentários

O Cronapp permite a adição de comentários em suas funções ou seus blocos, assim, acesse a opção **Adicionar comentário** no menu de contexto do bloco para exibir o ícone ?. É recomendado o uso de comentários em funções mais complexas, facilitando o entendimento da função para as pessoas que irão manter o bloco futuramente.

No entanto, fique atento ao comentar: seja direto no comentário e comente somente o necessário.

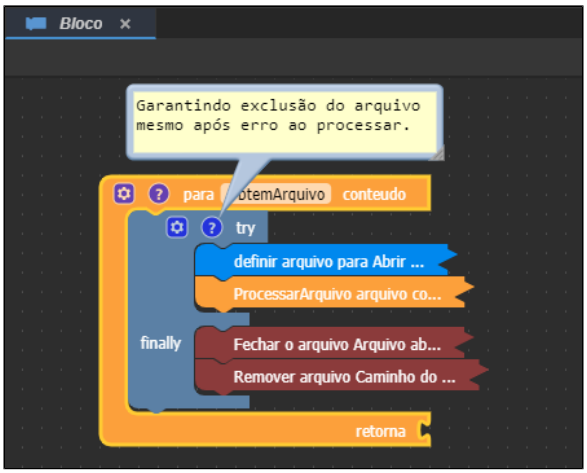


**Figura 6 - Adicionando comentário na função**

# Colapsar blocos

Longas funções atrapalham a sua leitura, assim, aconselhamos que sempre colapse trechos de blocos triviais ou que não estão sendo trabalhados no momento. Pois além de ser uma forma de organização, também evita alterações acidentais.

Para **colapsar** ou **expandir** um bloco, utilize as opções no menu de contexto ou dê um duplo clique sobre o bloco, colapsando o bloco selecionado e todos os blocos diretamente ligados a ele.

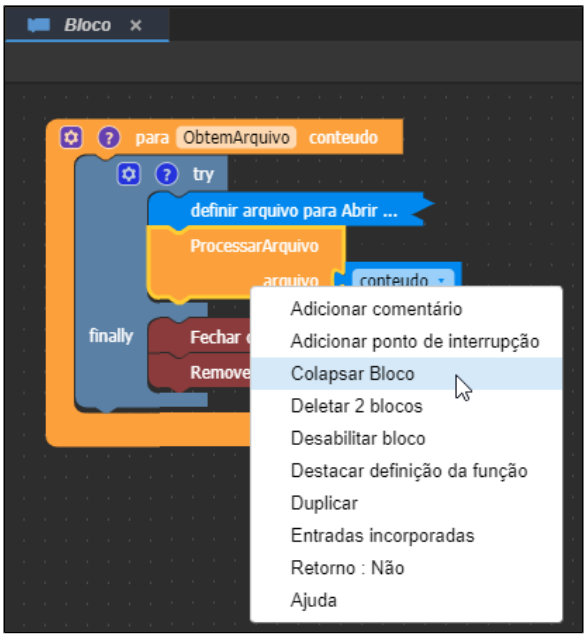


**Figura 7 - Colapsar bloco na função**

# Blocos organizadores

O bloco de programação Comentário permite adicionar comentários **em linha ou em bloco**, também é possível alterar a cor do bloco, o que possibilita uma melhor visualização e explicação de determinada funcionalidade. O bloco possui versões Cliente e Servidor.

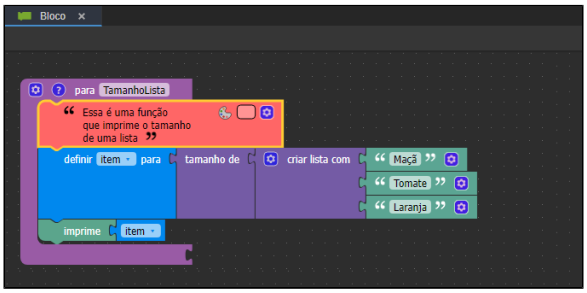


**Figura 8 - Adicionando comentário na função via bloco de programação**

O bloco de programação Grupo, demarca uma determinada área da função, indicando seu inicio e fim, facilitando assim a legibilidade e visualização de cada responsabilidade da função. O bloco possui versões Cliente e Servidor.

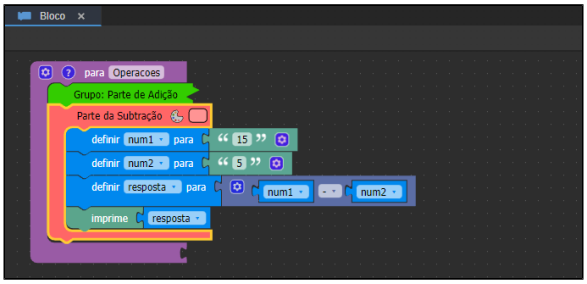


**Figura 9 - Adicionando comentário de grupo na função via bloco de programação**

# Uso de Variáveis de Sessão

Aconselhamos evitar uso de variáveis de sessão, pois isso é um limitante de escalabilidade em sistemas modernos. O uso de variável de sessão é permitido no Cronapp quando usamos o tipo de autenticação **N enhuma** ou **Sessão** ao invés de **Token**. Só mantemos essa opção por compatibilidade, porém ela será removida no futuro. O uso de **variável de sessão** vai obrigar o balanceador de cargas prender o usuário a um único servidor e isso não é uma boa prática para sistemas modernos, pois representa um grande ponto de falha e gargalo. Prefiram o uso de variáveis no cliente como: variáveis de escopo**, formulário**, session storage ou local sotorage. Um exemplo muito comum é o armazenamento de tokens de APIs. Tais tokens são obtidos pelo cliente e é recomendado que fossem armazenados no **local storage**.

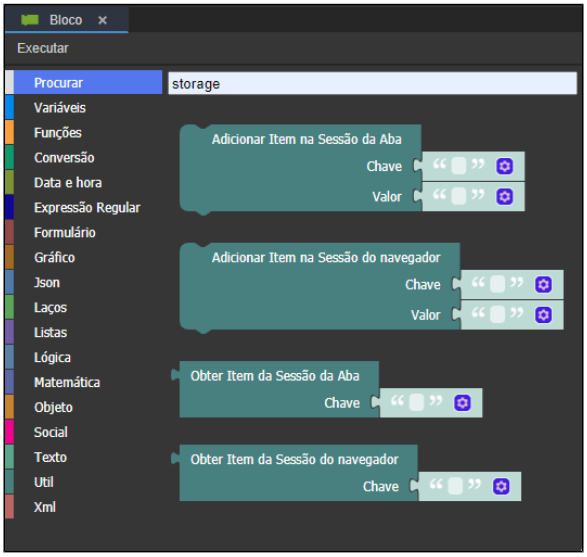


**Figura 10 - Funções de local storage (Sessão do Navegador) e session storage (Sessão da Aba)**

# Uso de Views de Banco de Dados

Aconselhamos uso comedido de *views* de banco de dados, pois isso pode representar um grande problema, caso seu sistema tenha um volume muito grande de dados. Se realmente for usar, procure usar índices e *views* materializadas.

# Uso de Datas e Horas

Trabalhar com **data e hora** em sistemas modernos requer entender completamente o uso de fuso horário. Uma aplicação web ou mobile é composta, na maioria das vezes, de banco de dados, servidor e cliente (navegador web ou aplicativo). É necessário entender bem que o fuso de cada uma dessas camadas pode ter especificações diferentes que podem gerar confusões ao implementar seu sistema. Sugerimos a leitura do documento Entendendo o funcionamento dos tipos data e hora para elucidar todas as dúvidas relacionadas ao uso de data e hora no Cronapp.

# Garantia de Execução de Rotinas

Uma prática muito comum é o uso do bloco Try com a opção Finally para garantir a execução de uma rotina, mesmo que ela falhe. Porém muitos usuários não utilizam esses blocos, podendo gerar **falha de segurança no sistema**. Imagine uma rotina que abre um arquivo temporário, faz uso desse arquivo e depois o apaga. A **Figura 11** mostra a forma insegura que muitos usuários costumam fazer para manipular arquivos, dessa forma, caso o bloco **Processa Arquivo** apresentar um problema, o arquivo ficará como lixo na pasta.


**Figura 11 - Exemplo de rotina não segura**

A **Figura 12** mostra como garantir que o arquivo será removido, independente de falhas no processamento.

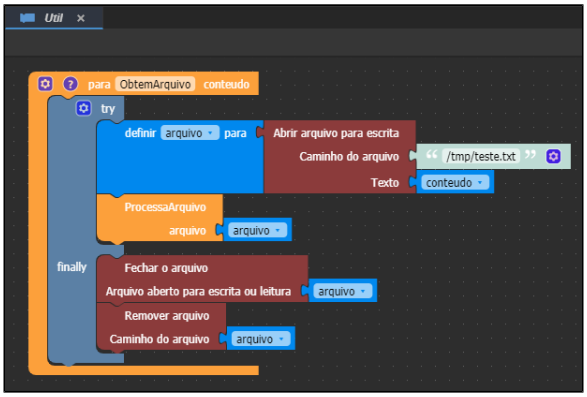


**Figura 12 - Exemplo de rotina segura**

# Tratamento de erros (Exceções)

O Cronapp fornece blocos para Tratar Exceções (try), Criar Exceções e Lançar Exceções.
Tais blocos devem ser usados para tratamento de erros para diversos fins, porém um bem comum é apresentar erros amigáveis para usuários. Exemplo: ao usar o bloco de Obter Conteúdo de uma URL e o retorno falhar por algum motivo, você pode capturar o erro e exibir uma mensagem amigável para o usuário. Observe a figura abaixo:

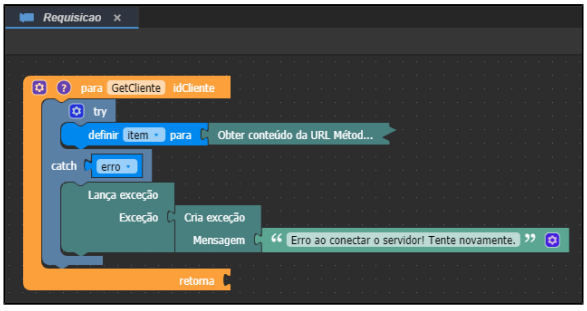


**Figura 13 - Tratamento de erros com informações amigáveis para o usuário**

# Log

Uma boa prática é gerar *log* das informações mais importantes do seu sistema, a fim de rastrear erros e monitorar execuções de rotinas. Para isso o Cronapp fornece uma função servidor chamada **Logar** com diversas opções para o desenvolvedor. Por padrão, essa função irá gerar *logs* em arquivos organizados por data no servidor de aplicações. Tais *logs* podem ser muito úteis para rastreabilidade em aplicações.

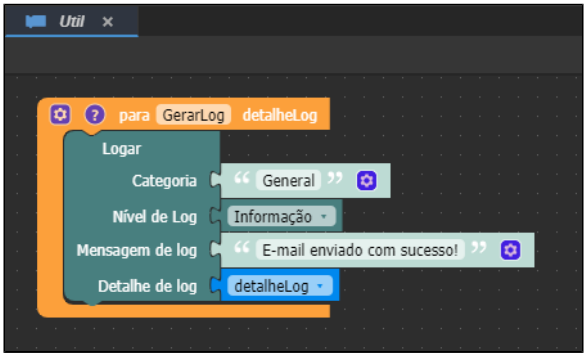


**Figura 14 - Bloco usado para gerar logs do Sistema**

# Agendador de tarefas vs bloco Agendar execução

O Agendador de tarefas foi introduzido no Cronapp a fim de permitir que o desenvolvedor agende execuções de blocos em *background*, podendo escolher entre diversas opções de disparadores. Já o Bloco Agendar Execução é uma forma manual de agendar a execução de uma rotina que pode ser executada uma única vez ou em loop. Muitos usuários usam o bloco **Agendar Execução** para iniciar execução de rotinas em *background* que se repetem com uma determinada periodicidade. Ex: envio de e-mail diário para usuários ou processamento de notas fiscais a cada 3 horas etc. É uma boa prática usar o Agendador de Tarefas para esses casos, visto que ele foi introduzido no Cronapp para esse fim. O **Agendador de Tarefas** tem inúmeras funcionalidades que já resolvem os diversos requisitos computacionais do desenvolvimento de uma aplicação. Alguns deles são:

- Garantia de Execução, mesmo que o sistema esteja offline;
- Execução única ou distribuída entre diversos servidores;
- Dezenas de opções de disparadores;
- Tolerância a falha;
- Entre outros...

**Observação**: para o exemplo citado (e-mail diário), se tivermos mais de um servidor e usarmos o bloco Agendar Execução (para esse caso seu uso é equivocado), cada servidor disparará o mesmo bloco e um usuário poderá receber e-mails repetidos. Conforme falamos, o Agendador de Tarefas tem funcionalidades para execução única mesmo usando vários servidores.

# Internacionalização

A grande maioria das aplicações hoje é feita para o mundo. O desenvolvimento de aplicações locais tem caído muito. Mesmo que a aplicação seja local, é uma boa prática criá-la já pensando nela como global. Isso significa que as mensagens precisam estar em pelo menos dois idiomas, as datas devem ser apresentadas em formatos diferentes, bem como unidades de medidas e máscara monetárias.
O Cronapp fornece dezenas de funcionalidades para esse fim.

Acesse a documentação de Internacionalização para entender como localizar seu projeto.

# Parametrização e Perfis

Ao criar um sistema, é bem comum que esse sistema tenha parametrizações diferentes para diferentes implantações (exemplo diferentes clientes). É uma boa prática usar os **Perfis** e **Parâmetros do Sistema** para tal fim. Na figura abaixo estamos parametrizando um servidor SAP de forma diferente no momento de desenvolvimento e produção.

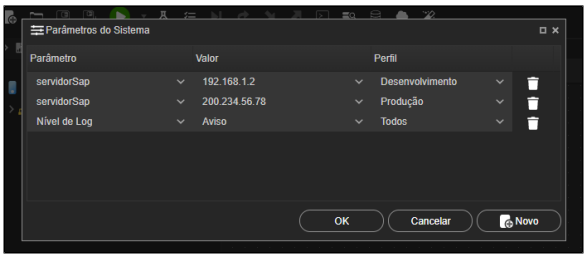


**Figura 15 - Exemplo de parametrização do sistema**

Acesse a documentação de Parâmetros do sistema para mais detalhes sobre essa ferramenta.

# Sistema de arquivos local

Sistemas modernos devem ser planejados para não terem estado fixo com servidores. Da mesma forma que é uma boa prática não usar variáveis de sessão, pedimos cuidado no uso de arquivos em sistemas modernos.

Uma arquitetura com mais de um servidor respondendo com alta disponibilidade a um sistema, pode gerar a seguinte situação: um usuário logado pode ter uma requisição direcionada para um servidor e, em seguida, outra requisição sendo direcionada a um segundo servidor. Agora imagine a seguinte situação, relativamente comum:  um sistema que permita o upload de um arquivo e um botão para processar este arquivo, o upload pode ser feito em um servidor, porém o processamento pode cair outro, onde esse arquivo não vai existir. Sendo assim, a boa prática para esses casos de múltiplas requisições acessando o mesmo arquivo é o uso de locais de armazenamento na nuvem, como por exemplo, o Armazenamento dos Serviços de Cloud do Cronapp. O Cronapp provê toda a abstração para manipulação de arquivos na nuvem. É possível manipular arquivos como se eles estivessem em banco. Observem a figura abaixo, ela mostra a definição de um campo, informando que seu conteúdo deve ser armazenado no serviço de nuvem informado (Cronapp Cloud Services, AWS S3, Dropbox entre outros):

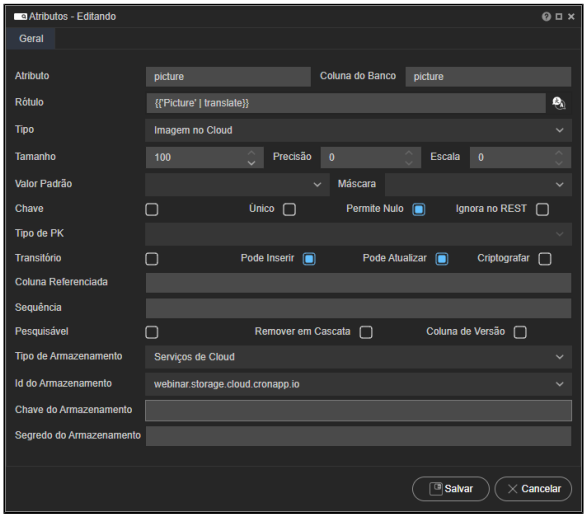


**Figura 16 - Armazenando o conteúdo de um campo de uma entidade na nuvem.**

# Imagens e arquivos em Banco

É uma péssima prática armazenar imagens em banco. O uso de serviços de armazenamento na nuvem para tal fim é o mais indicado. Exemplo: **Dropbox**, **S3**. Para mais informações, confira o tópico "Armazenamento" da documentação Serviços de Cloud.

# Controle de Versão

É de vital importância o uso do **controle de versão GIT** em seus projetos. O **GIT** é gratuito através do **GitHub**, **BitBucket** e outros. O uso do **GIT** trará segurança para seus códigos fontes e um histórico completo de toda a vida deles.

Acesse as documentações Versionamento usando Git e Fluxo de trabalho Git flow para mais detalhes.

Aviso

O Cronapp mantém o projeto de seus usuários em nuvem e possui sistema de backup durante o seu desenvolvimento, porém, não se responsabiliza pelos códigos fontes desses projetos, sendo necessário o uso de sistemas de controle de versão ou backups manuais.

# Arquivos e diretórios

A estrutura de arquivos dos projetos Cronapp possuem diversos diretórios que são específicos para determinadas funções - por exemplo, a pasta Bloco de programação (blockly) deve conter apenas os arquivos referente aos blocos (*.blockly, *.blockly,js, *.java e *.map), já a pasta **Relatórios** (reports) deve possuir apenas o conteúdo referente aos relatórios do sistema e assim por diante. Caso queira adicionar novos arquivos com funções distintas, recomendamos criar diretórios em níveis hierárquicos mais altos - por exemplo, criar um diretório abaixo da pasta `Código Fonte Servidor` (java) ou `Código Fonte Web` (webapp).

Com isso é possível manter uma organização em seu projeto e evitar problemas de direcionamento de arquivos.