Arquivo

Estrutura de arquivos

A estrutura de arquivos do projeto possui diferenças durante o desenvolvimento e em produção, após gerar o "**.war**". É importante entendermos essa diferença principalmente ao trabalharmos com os blocos Pasta da Aplicação, Pasta temporária e Pasta reciclável da aplicação. Mostraremos algumas diferenças e como tratar.

Arquivos em desenvolvimento

Os arquivos ficam localizados na árvore de arquivo no canto esquerdo da área de produção, como mostrado na figura 1. Note que a pasta *mobileapp* fica localizada no endereço "`src/main/mobileapp`", com isso, caso precise acessar a pasta, você precisaria informar esse caminho nos blocos de programação. No entanto, ao exportar seu projeto no formato "**.war"** , o caminho não será o mesmo, após compilado ele não ficará mais dentro de "**`src`**", e sim na raiz do projeto, como mostrado na figura 1.1.

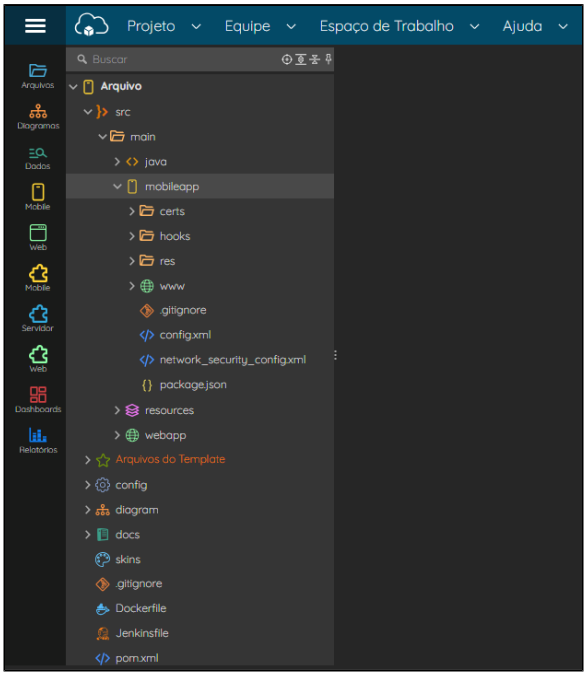

Figura 1 - Localização da pasta mobileapp em produção

Arquivos no .war

Na figura 1.1, verifique que a estrutura dos arquivos estão diferentes da estrutura de desenvolvimento, que o caminho para acessar a pasta **webapp** precisaria ser acessado pelo endereço "`src/main/webapp`", já quando o arquivo é extraído no formato "*.war*", o conteúdo da pasta `webapp` está na raiz, podendo também acessar dentro dela a pasta **mobileapp**. Para mais informações da estrutura de um arquivo *.war*, consulte a documentação de referência.

Nesta página

- Estrutura de arquivos
  - Arquivos em desenvolvimento
  - Arquivos no .war
  - Como tratar pastas em ambientes diferentes
- Upload de arquivos
  - CRUDs
    - Dropbox e S3
  - Componente visual
  - Bloco de programação
- Blocos da categoria Arquivo

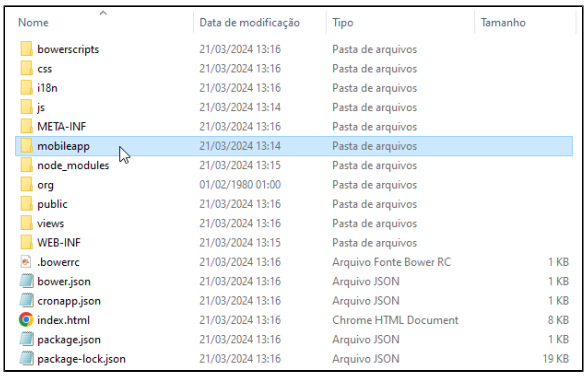

Figura 1.1 Localização da pasta mobileapp em .war

Para a pasta **mobileapp** ser visualizada, é preciso que ela seja selecionada na hora de efetuar a extração dos arquivos no formato "`.war`", para isso, acesse o tópico **War** da documentação Importar e exportar projetos.

Como tratar pastas em ambientes diferentes

Abaixo (Figura 1.2) usamos um exemplo para baixar o arquivo "meuarquivo.txt" que está dentro da pasta *views* do ambiente *web*. O código abaixo faz a verificação nas duas árvores de arquivos diferentes, tanto em desenvolvimento quanto em produção "*.war*".

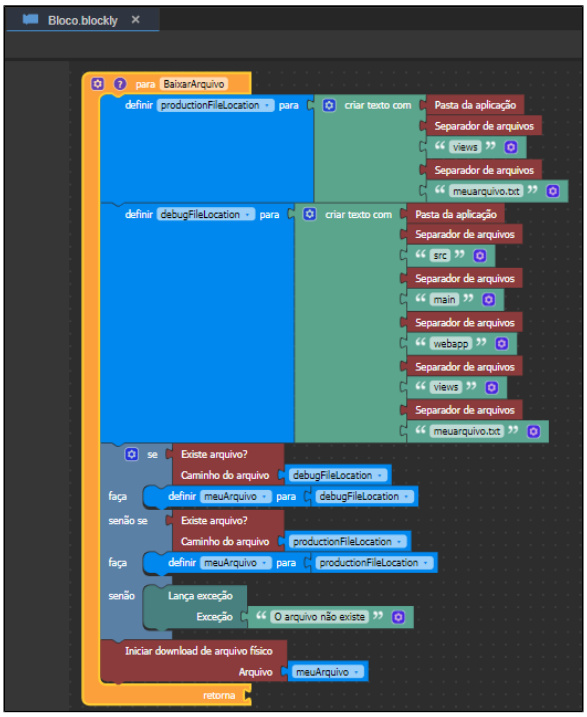

Figura 1.2 - função para verificar e baixar arquivo

Upload de arquivos

O Cronapp possui diversas abordagens para permitir que o usuário envie arquivos para o sistema, veremos aqui as principais.

CRUDs

As páginas CRUDs criadas a partir do Diagrama de dados permite tratar o envio de arquivos de forma simples e sem a necessidade de desenvolver uma lógica de controle. Nesse modo, a configuração é feita a partir da janela de edição do atributo de uma classe, onde é possível defini-lo como **Arquivo** ou **I magem** e em seguida selecionar o local onde serão hospedados: em **banco de dados** ou ***cloud***.

No exemplo da figura 2, o atributo "conteudo" da classe "Anexo" foi configurado com o tipo "Arquivo no Cloud" para o serviço de armazenamento do Cronapp. Dessa forma, ao criar um registro na página CRUD que representa a classe, o arquivo será enviado automaticamente para o local de armazenamento na nuvem, ficando apenas o registro do seu endereço.

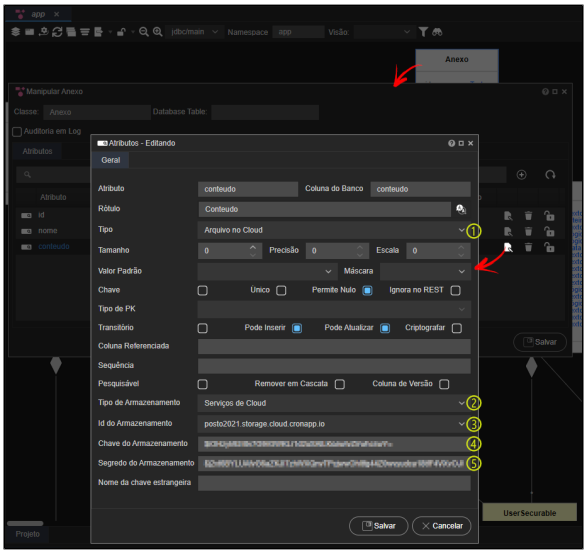

Figura 2 - Configuração de um atributo do tipo Arquivo na nuvem

Os campos **Tipo de Armazenamento**, **Id do Armazenamento**, **Chave do Armazenamento** e **Segredo do Armazenamento** só ficam habilitados ao definir o atributo como tipo "Arquivo no Cloud" ou "Imagem no Cloud".

1. **Tipo**: define o tipo do atributo, abaixo os 4 tipos específicos para arquivos.
    - **Arquivo no Banco**: o arquivo é convertido para base64 e salvo no banco de dados. Após gerar a página CRUD, esse atributo será tratado com o componente visual Envio de Arquivo Dinâmico.
    - **Arquivo no Cloud**: o arquivo é direcionado para o serviço da nuvem selecionado. Após gerar a página CRUD, esse atributo será tratado com o componente visual Envio de Arquivo Dinâmico.
    - **Imagem no Banco**: a imagem é convertida para base64 e salva no banco de dados. Após gerar a página CRUD, esse atributo será tratado com o componente visual Image m dinâmica.
    - **Imagem no Cloud**: a imagem é direcionada para o serviço da nuvem selecionado. Após gerar a página CRUD, esse atributo será tratado com o componente visual Image m dinâmica.
2. **Tipo de Armazenamento**: local onde serão hospedados os arquivos.
    - **Serviço de Cloud**: o Cronapp possui seu próprio serviço de armazenamento na nuvem que provê alta disponibilidade, durabilidade, armazenamento ilimitado e baixo custo. Veja mais detalhes no tópico "Armazenamento" dos Serviços de cloud.
    - **S3**: serviço de armazenamento Amazon.
    - **Dropbox**.
3. **Id do Armazenamento**: utilizado apenas para selecionar a conta do serviço de cloud do Cronapp. Após selecionar, os campos abaixo serão preenchidos automaticamente.
4. **Chave do Armazenamento**: chave disponibilizada por uma das opções de armazenamento.
5. **Segredo do Armazenamento**: segredo disponibilizado por uma das opções de armazenamento.

Configurado essa etapa, basta salvar, gerar as camadas de persistências e a página CRUD. Todos os arquivos/imagens enviados por essa página serão direcionados para o local de armazenamento selecionado.

Dropbox e S3

O Cronapp possui uma rotina interna para o armazenamento e organização de imagens e arquivos enviados ao Dropbox ou S3 quando o atributo de uma classe é configurado com os tipos **Arquivo /imagem no Cloud** e uma página CRUD é gerada. A hierarquia dos diretórios não é parametrizável e seguirá a entidade de origem, independente da plataforma. A estrutura de pastas é construída da seguinte forma:

- **Para o Dropbox**, inclui uma pasta para aplicativos, o nome do aplicativo em questão e, em seguida, a organização das pastas por entidade e campo, seguindo o formato: `Aplicativos /nomeDoAppCriado/NomeDaEntidade/CampoDaEntidade/Id-ValorDoCampoId.extensao`
- **Para o Amazon S3**, inclui o nome do bucket e o ID da aplicação, seguido pela organização por entidade e campo, no formato: `NomeDoBucket/IdAplicacao/NomeDaEntidade /CampoDaEntidade/Id-ValorDoCampoId.extensao`

Essa abordagem garante consistência na organização dos arquivos pois é baseada nas entidades de origem, adaptando-se às particularidades de cada plataforma.

Vale salientar que a estrutura gerada a partir da página CRUD não realiza, de forma nativa, exclusão dos arquivos dentro das plataformas Dropbox e S3, apenas inserção e leitura.

Componente visual

Existem dois componentes visuais que tratam do envio de arquivos, em ambos os casos serão necessários criar uma regra de negócio para tratar o conteúdo recebido. Acesse suas documentações para mais detalhes.

- O Envio de Arquivo Dinâmico (Mobile ou Web) encaminha o arquivo para um diretório temporário da aplicação até que o sistema faça o direcionamento para o local correto.
- O Imagem Dinâmica (Mobile ou Web) converte o arquivo imagem para base64 e, em seguida, o sistema deve tratar do seu destino.

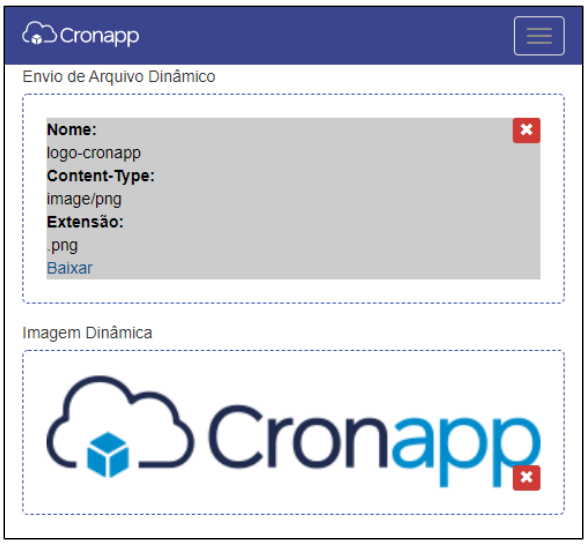

Figura 2.1 - Componentes Envio de Arquivo Dinâmico e Imagem Dinâmica

Bloco de programação

O bloco de programação servidor Upload de Arquivo exibe automaticamente uma janela modal para o usuário enviar seus arquivos. Esse bloco permite configurar restrições por extensões e tamanho, além de permitir o envio de um ou múltiplos arquivos.

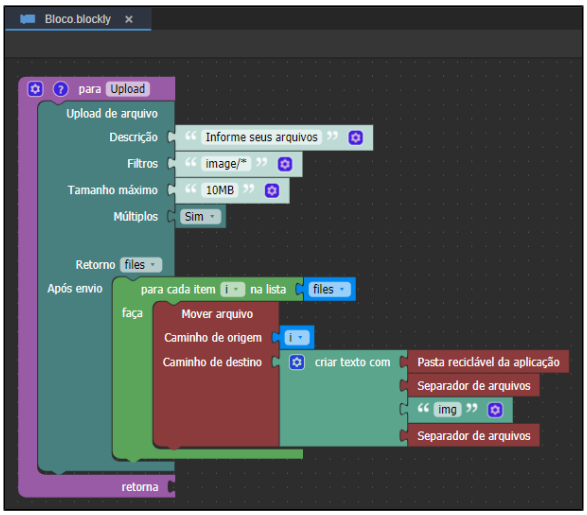

Figura 2.2 - Bloco Upload de arquivo

No exemplo da figura 2.2, os arquivos serão encaminhados para a **Pasta reciclável da aplicação**. Porém, em um exemplo real, eles poderiam ser enviados para um *bucket*, como o sistema de armazenamento S3 da Amazon. Esse processo poderia ser feito utilizando o plugin Cronapp **Amazon S3**.

Blocos da categoria Arquivo

Lista dos blocos servidores da categoria Arquivo.

- cronapi.io.Operations.compressImage
- cronapi.io.Operations.cropImage
- cronapi.io.Operations.downloadFileFromUrl
- cronapi.io.Operations.downloadUrltoFile
- cronapi.io.Operations.fileAppClassesDir
- cronapi.io.Operations.fileAppDir
- cronapi.io.Operations.fileAppend
- cronapi.io.Operations.fileAppReclycleDir
- cronapi.io.Operations.fileCanRead
- cronapi.io.Operations.fileCanWrite
- cronapi.io.Operations.fileClose
- cronapi.io.Operations.fileCopy
- cronapi.io.Operations.fileCreate
- cronapi.io.Operations.fileCreateWithByteArray
- cronapi.io.Operations.fileDownload
- cronapi.io.Operations.fileExists
- cronapi.io.Operations.fileFlush
- cronapi.io.Operations.fileGetNumberOfLines
- cronapi.io.Operations.fileGetParent
- cronapi.io.Operations.fileGetSize
- cronapi.io.Operations.fileMD5
- cronapi.io.Operations.fileMove
- cronapi.io.Operations.fileOpenToRead
- cronapi.io.Operations.fileOpenToWrite
- cronapi.io.Operations.fileRead
- cronapi.io.Operations.fileReadAll
- cronapi.io.Operations.fileReadAllToBytes
- cronapi.io.Operations.fileReadContentWithCharset
- cronapi.io.Operations.fileRemove
- cronapi.io.Operations.fileRemoveAllFolder
- cronapi.io.Operations.fileRename
- cronapi.io.Operations.fileSeparator
- cronapi.io.Operations.fileTempDir
- cronapi.io.Operations.folderCreate
- cronapi.io.Operations.getFileExtension
- cronapi.io.Operations.isDirectory
- cronapi.io.Operations.isFileEoF
- cronapi.io.Operations.listFiles
- cronapi.io.Operations.readBytesFromStream
- cronapi.io.Operations.readLine
- cronapi.io.Operations.resizeImage
- cronapi.io.Operations.rotateImage
- cronapi.io.Operations.startDownload

- cronapi.io.Operations.unZip
- cronapi.io.Operations.zipFile
- Obter caminho de arquivo / Get file path
- Obter nome do arquivo / Get file name
- Obter pasta pai / Get parent