run2biz

Low Code Studio

1. A Plataforma

1 - Bem-vindo ao Low Code Studio

O Low Code Studio é a plataforma da Run2Biz para criar aplicativos web e mobile de forma visual e intuitiva — sem precisar escrever código do zero. Com ele, você monta telas, conecta dados e publica seu aplicativo em muito menos tempo do que com o desenvolvimento tradicional.

> **Para quem é esta documentação?**
>
> Para você que vai usar o Low Code Studio no dia a dia — seja para criar novas telas, gerenciar projetos ou publicar aplicativos. Não é necessário ter experiência em programação para começar.

---

2 - O que você pode fazer com o Low Code Studio

Criar telas visualmente

Monte a interface do seu aplicativo arrastando e soltando botões, listas, campos e outros elementos.

Publicar em web e mobile

O mesmo projeto funciona no navegador e em dispositivos iOS e Android, sem retrabalho.

Conectar com dados reais

Integre seu aplicativo a bancos de dados e APIs para exibir e salvar informações do seu negócio.

Organizar múltiplos projetos

Gerencie diferentes aplicativos em um só lugar, com controle de versão e ambientes separados.

---

3 - Como a plataforma está organizada

Ao abrir o Low Code Studio, você encontra um painel de projetos onde todos os seus aplicativos ficam reunidos. Ao entrar em um projeto, o ambiente de desenvolvimento se divide em algumas áreas principais:

Editor visual

Onde você monta as telas do aplicativo, posiciona elementos e define a aparência de cada parte.

Biblioteca de elementos

O catálogo de componentes prontos para usar: botões, campos de texto, listas, menus e muito mais.

Blocos de código

Onde ficam as regras de funcionamento do aplicativo — ações que acontecem quando o usuário interage com a tela.

Configurações do projeto

Opções gerais do aplicativo: idiomas, integrações, ambientes de publicação e configurações de loja.

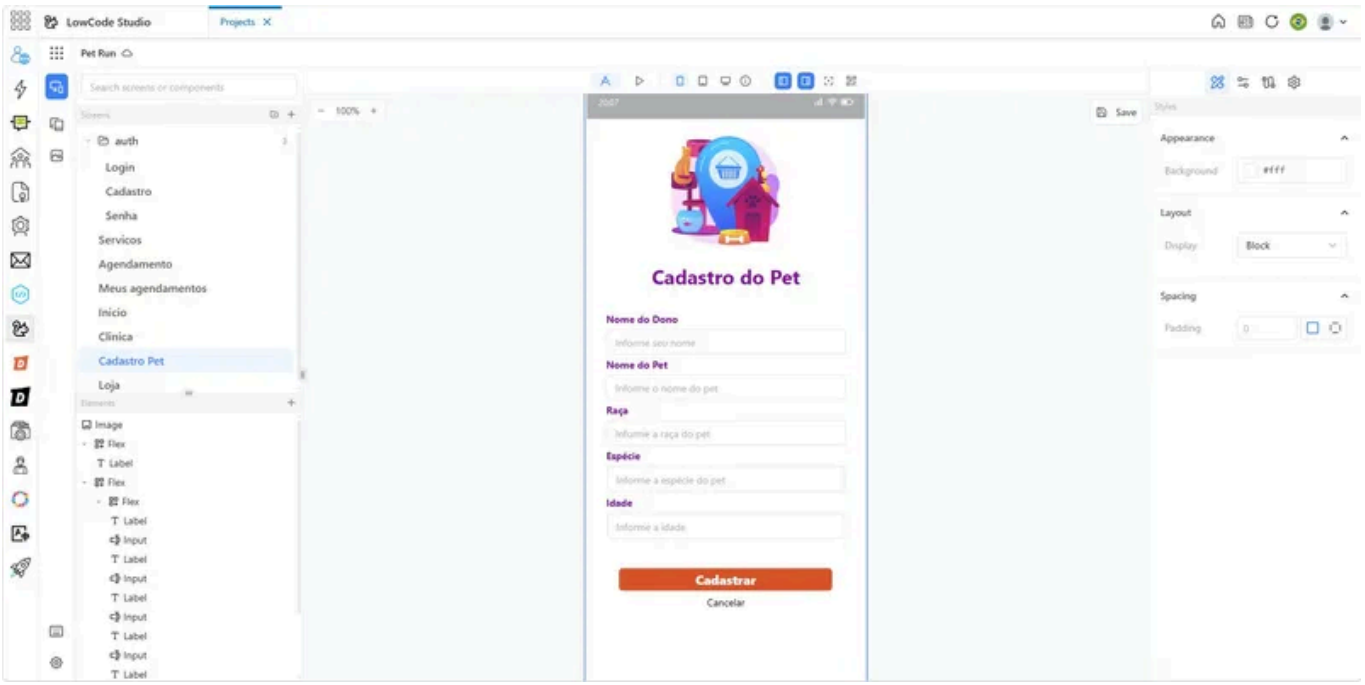
Nas próximas seções desta documentação, vamos detalhar cada uma dessas áreas com exemplos práticos e passo a passo.

2. Principais Características

1 - Principais Características

1.1 - Ferramentas de Desenvolvimento

- **Editor Visual:** Interface de arrastar e soltar para projetar telas com componentes
- **Biblioteca de Componentes:** Elementos da interface incluindo botões, entradas, seleções e contêineres de layout
- **Editor de Estilo:** Opções abrangentes de estilo para tipografia, espaçamento, bordas e layout
- **Gerenciamento de Blocos de Código:** Crie e gerencie blocos de código reutilizáveis
- **Painel de Projeto:** Organizar e gerenciar múltiplos projetos de aplicação
- **Desfazer/Refazer:** Suporte completo para desfazer e refazer para todas as operações de edição visual



Tela LowCode Studio com uma app de pet shopp.

1.2 - Desfazer/Refazer

O editor visual suporta operações de desfazer e refazer para todas as operações de edição de tela e composição.

Atalhos de teclado

- `Cmd+Z` (macOS) / (Windows/Linux): Desfazer a última ação `Ctrl+Z`
- `Cmd+Shift+Z` (macOS) / (Windows/Linux): Refazer a última ação desfeita `Ctrl+Shift+Z`

Operações de Apoio

O acompanhamento histórico inclui:

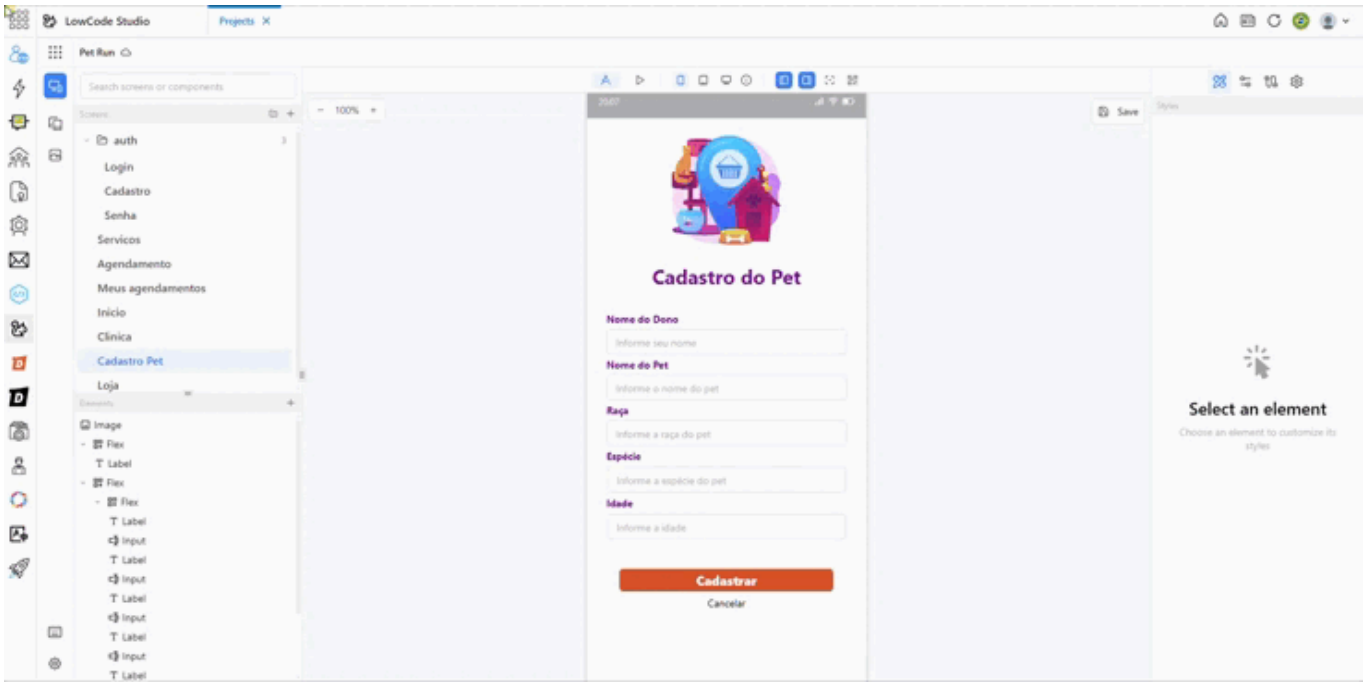
- Adicionar ou remover elementos de telas e composições
- Mover elementos por meio de arrastar e soltar ou ajustes de posição
- Modificar propriedades dos elementos (texto, marcadores de posição, estado desativado, etc.)
- Mude os estilos dos elementos (cores, fontes, espaçamento, bordas, etc.)
- Ajuste pontos de interrupção responsivos (padrão, sm, md, lg)
- Expandir/colapsar estados de elementos na visualização em árvore

Gestão da História

- **História Separada:** Cada tela e composição mantém sua própria história independente
- **Limite de Capacidade:** Máximo de 50 ações armazenadas por tela ou composição
- **Agrupamento Automático:** Mudanças rápidas em 500ms são agrupadas em um único passo de desfazer (por exemplo, digitar texto ou arrastar um controle deslizante)
- **Navegação:** A história é preservada ao alternar entre telas ou composições
- **Integração com Salvamento Automático:** Operações de desfazer/refazer marcam o documento como sujo e acionam o salvamento automático após 2 segundos

Limitações Atuais

- Operações de binding (criar, editar ou excluir) não estão incluídas em undo/redo
- O histórico não persiste entre as recargas da página
- Desfazer/refazer está disponível apenas no modo de design (não no modo de prévia)



> O sistema de desfazer/refazer usa rastreamento de histórico baseado em patches para eficiência de memória, consumindo aproximadamente 500 bytes por ação, em comparação com snapshots de estado completo, que exigiriam de 5 a 20KB por ação

---

1.3 - Desenvolvimento Móvel

- **Integração com React Native:** Construa aplicativos móveis multiplataforma
- **Suporte a Módulos Expo:** Acesso às capacidades do dispositivo por meio de módulos configuráveis:
  - Câmera e seletor de imagem
  - Serviços de localização
  - Acesso ao sistema de arquivos
  - Autenticação
  - Contatos e calendário
  - Suporte de áudio
- **Expo Build System:** Processo de compilação para aplicações iOS e Android
- **Configuração da App Store:** Configurações para Apple App Store e Google Play Store

---

1.4 - Recursos de Integração

- **Conectividade de Banco de Dados:** Integração com banco de dados SQL com migrações
- **Gerenciamento de APIs:** Criar e gerenciar endpoints de API
- **Soluções de Armazenamento:** Capacidades de armazenamento de arquivos
- **Autenticação:** Sistema de autenticação baseado em token

1.5 - Opções de Implantação

- **Conteinerização:** suporte a Docker com gráficos Helm Kubernetes
- **Gerenciamento de Ambiente:** Configurar diferentes ambientes de implantação
- **Configuração do Serviço:** Configurações de rede e recursos

---

2 - Arquitetura

A plataforma é construída com uma arquitetura moderna:

- **Estrutura do Monorepo** Tudo: Organizada em aplicativos e bibliotecas compartilhadas
  - **Engine:** Runtime para aplicações móveis
  - **Servidor:** API backend e serviços de banco de dados
  - **Studio:** UI do ambiente de desenvolvimento
- **React Native Frontend:** interface multiplataforma com estilo NativeWind
- **Hono Backend:** endpoints de API para gerenciamento de projetos
- **PostgreSQL com Drizzle:** Armazenamento de dados com suporte à migração

---

3 - Componentes do Projeto

- **Telas:** Contêineres visuais para elementos de interface
- **Elementos:** Componentes de interface que podem ser adicionados às telas
- **Estilos:** Configuração visual da aparência

- **Blocos de Código**: Implementação de funcionalidades personalizadas
- **Ativos**: Imagens e outros recursos
- **Configurações**: Opções de configuração do aplicativo
- **Internacionalização**: Suporte multilíngue === Documentação de Elementos

---

3.1 - Componentes vs Elementos

Compreender a distinção entre Componentes e Elementos é crucial para trabalhar com a plataforma Lowcode Studio:

Componente

Um Componente é um blueprint ou definição armazenado na biblioteca de componentes. Ele representa um tipo de elemento de interface com:

- Propriedades e valores padrão
- Encadernações disponíveis (propriedades e eventos)
- Estilos e configurações base
- Comportamento predefinido

Os componentes são definidos em e servem como modelos. Exemplos: Botão, Entrada, Visualização, Lista.components.json

---

4 - Elemento

Um Elemento é uma instância de um Componente colocada em uma tela ou composição. Cada elemento tem:

- Um identificador UUID único
- Valores de propriedade personalizados específicos para aquela instância
- Estilos individuais e configurações responsivas
- Ligações específicas a variáveis e funções

Quando você arrasta um Componente da paleta, uma instância de Element é criada na sua tela.

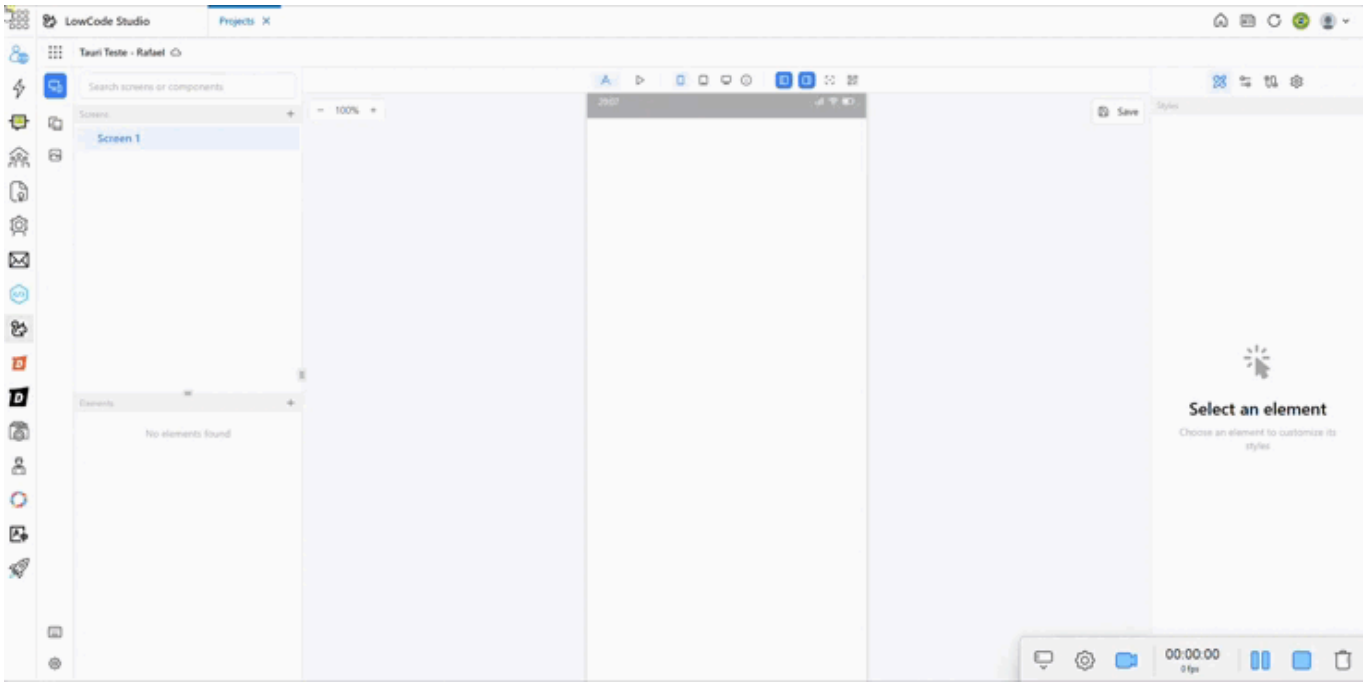
Exemplo 1. Exemplo

Componente: "Botão" (blueprint com propriedades padrão)

↓

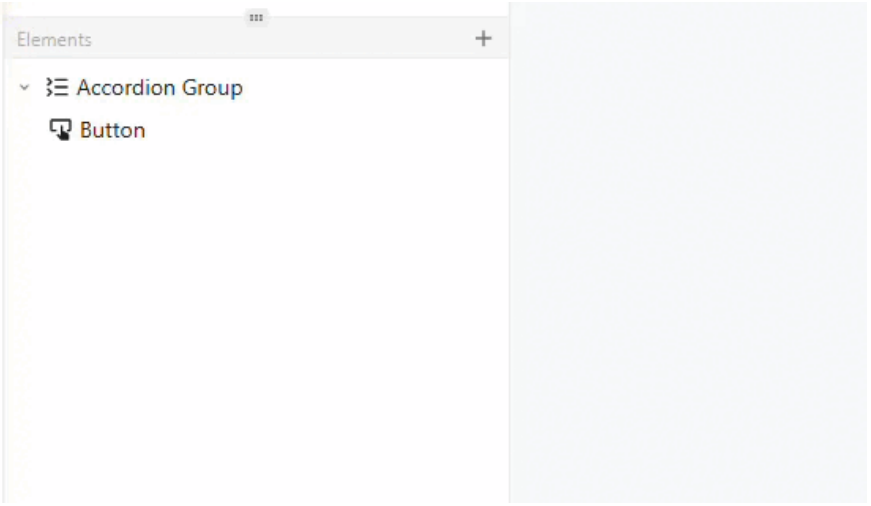
Element: Instância de botão com UUID "abc-123"

- Propriedades: `{ text: "Submit", disabled: false }`
- Encadernações: `{ press: [functionId: "xyz"] }`
- Estilos: `{ backgroundColor: "#007bff", borderRadius: "4px" }`



3. Categorias de Elementos

Categorias de Elementos



Os elementos são organizados em três categorias principais:

---

1 - Elementos de Layout

Contêineres visuais e componentes de exibição usados para estruturar telas e apresentar informações.

- Botão - Botão interativo com múltiplas variantes
- Flex - Contêiner flexível para layouts responsivos
- Ícone - Exibição de ícones da biblioteca Iconify
- Imagem - Exibição de imagem com gestão de ativos
- Rótulo - Componente de exibição de texto
- Lista - Componente de lista repetível para coleções
- Dica de ferramenta - Ajuda contextual

- Diálogo - Sistema de diálogo modal
- Popover - Sistema de sobreposição Popover
- Gaveta - Painel deslizante para fora
- Tabs - Sistema de conteúdo com abas
- Grupo de Acordeão - Recipiente de acordeão dobrável
- Item do acordeão - Painel individual do acordeão
- Componente de renderização condicional - Condicional
- Paginação - Componente de navegação de página
- Screen - Elemento de contêiner de tela
- Assinatura - Componente de captura de assinatura
- WebView - Visualizador de conteúdo web embutido
- Instância de Composição - Instância de composição reutilizável
- Carrossel - Carrossel de imagens/conteúdo (Em desenvolvimento)
- Câmera - Componente de captura de câmera (Em desenvolvimento)
- Mapas - Componente de exibição de mapas (em desenvolvimento)

---

2 - Elementos de Forma

Componentes de entrada para coleta e validação de dados do usuário.

- Entrada - Campo de entrada de texto
- Select - Componente de seleção suspensa
- Área de Texto - Entrada de texto multilinha
- Formulário - Contêiner de formulário com validação
- Campo de Caixa de Seleção - Entrada de caixa de seleção com etiqueta
- Grupo de Rádio - Grupo de botões de rádio
- Entrada de Arquivo - Componente de upload de arquivo
- Selector de Cor - Componente de seleção de cores

---

3 - Elementos de Solicitação

Componentes para obtenção de dados e integrações externas.

- Get - componente de requisição HTTP GET (Em desenvolvimento)
- Busca de Listas - Busca de dados de lista (Em desenvolvimento)

4. Documentação Relacionada

4.1. Funções

1 - Funções

Funções são blocos de código que podem ser reutilizados em diferentes partes do seu projeto.

1.1 - Estrutura geral da função

```
const params = {
  username: String,
  password: String,
}
const handler = (ctx, args) => {
  const { event, router, element } = ctx
  const { username, password } = args
}
module.exports = {
  params,
  handler
}
```

1.2 - Parâmetros (opcionais)

```
const params = {
  username: String,
  password: String,
}
```

Esse objeto é opcional e define os parâmetros que a função receberá. - Opções de tipo: - Corda - Número - Booleano - Objeto - Array - Data

1.3 - Manipulador

```
const handler = (ctx, args) => {
  const { event, router, element } = ctx
  const { username, password } = args
}
```

O handler é a função que será executada quando a função for chamada. Ele recebe dois parâmetros:

`ctx` : Contexto da função, contém informações sobre o evento, funções de navegação e elemento atual

```
const { event, router, element } = ctx
const {
  navigate,
  ...rest,
} = router
```

Podem ser diferentes dependendo do evento que acionou a função. `Event`

`args`: Argumentos de função, contém os valores dos parâmetros da função. E se estiver dentro de um componente, contém um objeto item com o item do elemento que chamou essa função. `List`

1.4 - Exportação

A função precisa ser exportada para poder ser usada em outros lugares.

```
module.exports = {
  params,
  handler
}
```

1.5 - Funções de importação

Melhores práticas: Importar módulos fora do handler, porque se importados dentro do handler, eles serão carregados toda vez que a função for executada, impactando o desempenho da aplicação.

Esta é uma função que carrega um módulo e é necessário usar a sintaxe para importar o módulo. `require~<module-name>`

- Módulos padrão:
- `~libs/router`
- `~libs/elements`

```
const params = {
  username: String,
  password: String,
}

const auth = require("~libs/router")
const elements = require("~libs/elements")

const handler = (ctx, args) => {}

module.exports = {
  params,
  handler
}
```

4.2. Bidings

Bindings

Bindings (Vinculação) são conexões feitas entre elementos e outros objetos de aplicação.
Tipos de Bindings:

---

1 - Propriedades e Vinculações Variáveis (Property Binding)

A Vinculação de propriedades é uma forma de vincular a propriedade de um elemento a uma variável. Quando o valor da variável muda, a propriedade do elemento é atualizada, e vice-versa.

---

2 - Eventos e Vinculação de Funções (Event Binding)

Vinculação de eventos é uma forma de vincular o evento de um elemento a uma função. Quando esse evento é acionado, a função é executada.

---

3 - Variáveis e Vinculação de Funções (Watcher)

Um Whatcher é uma forma de vincular uma ou mais variáveis a uma função. Um Whatcher monitora os valores das variáveis e executa a função quando os valores das variáveis mudam.

4.3. Composições

Composições

Composições são componentes de UI que podem ser aninhados em telas, permitindo construir bibliotecas de componentes e manter a consistência entre suas aplicações.

---

1 - O que é uma Composição?

Uma composição é basicamente um fragmento de tela reutilizável. Ele contém:

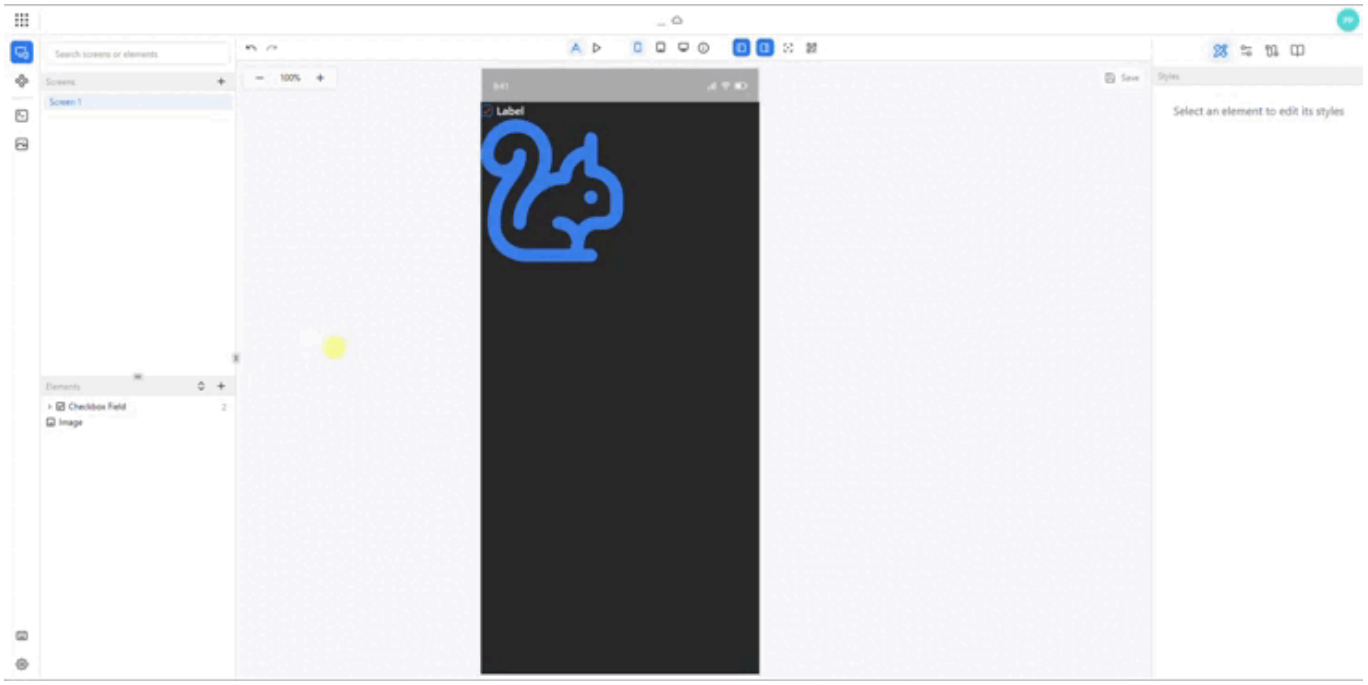
- Uma coleção de elementos (botões, comandos, contêineres, etc.)
- Estilos responsivos para diferentes pontos de interrupção (celular, tablet, desktop)
- Propriedades que podem ser sobrescrevidas por instância
- Suporte para aninhamento de outras composições (até 5 níveis de profundidade)

> □ As composições são expandidas no lado do servidor durante a prévia, garantindo um comportamento consistente entre o estúdio web e os aplicativos móveis.

---

2 - Criando uma Composição

1. Clique no botão "+" no painel Composições na barra lateral esquerda
2. Nomeie sua composição (por exemplo, "UserCard", "ProductItem")
3. Adicione elementos por meio de arrastar e soltar a partir da paleta
4. Configure propriedades e estilos para cada elemento
5. O save ocorre automaticamente após 2 segundos de inatividade




```
// Example composition structure
{
  uuid: "comp-123",
  name: "UserCard",
  category: "ui",
  elements: [
    {
      type: "Flex",
      properties: { direction: "column" },
      children: [
        { type: "Text", properties: { text: "Username" } },
        { type: "Button", properties: { label: "View Profile" } }
      ]
    }
  ],
  styles: {
    default: { padding: "16px" },
    sm: { padding: "12px" },
    md: { padding: "16px" },
    lg: { padding: "20px" }
  }
}
```

3 - Uso de Composições em Telas

Uma vez criadas, as composições aparecem na paleta Componentes, na seção "Composições".

1. Navegue até uma tela no seu projeto
2. Encontre sua composição na paleta
3. Arraste para a tela
4. A composição é inserida como um elemento `CompositionInstance`

```
// CompositionInstance structure
{
  type: "CompositionInstance",
  uuid: "instance-456",
  properties: {
    compositionId: "comp-123",
    overrides: {
      // Per-instance style overrides
      "element-789": {
        styles: {
          default: { backgroundColor: "blue" }
        }
      }
    }
  }
}
```

---

4 - Modo de Edição

O sistema opera em dois modos de edição:

Modo	Descrição
Modo Tela	Modo padrão. Edite o layout da tela e adicione composições às telas.
Modo de composição	Editar o conteúdo da composição. Ativado clicando em uma composição na lista.

> ! Ao editar uma composição, as alterações se aplicam a **todas as instâncias** dessa composição em todas as telas.

---

5 - Composições Aninhadas

Composições podem ser aninhadas dentro de outras composições com até 5 níveis de profundidade.

```
// Example: UserCard composition contains Avatar composition
UserCard (Level 1)
└── Flex
    ├── Avatar (Level 2) <- Another composition
    │   └── Image
    └── Text
```

Regras de Nidificação:

- Profundidade máxima de nidificação: 5 níveis
- Referências circulares são impedidas (Composição A não pode conter Composição B se B já contém A)
- Avisos visuais aparecem na paleta para composições com profundidade >= 4

---

6 - Substituições de Estilo

Cada instância de composição pode ter estilos personalizados por elemento.

⚠ As sobrescrituras de estilo são aplicadas por instância e não afetam outras instâncias da mesma composição.

Exemplo de estrutura de substituição:

```
{
  type: "CompositionInstance",
  properties: {
    compositionId: "user-card",
    overrides: {
      "button-123": {
        styles: {
          default: { backgroundColor: "#007bff" },
          sm: { fontSize: "14px" }
        }
      }
    }
  }
}
```

---

7 - Expansão no Lado do Servidor

Breaking Change (dezembro de 2025): Visualização móvel não recebe mais telas via . Agora ele busca dados diretamente do servidor `.postMessage`

Impacto:

- A primeira carga adiciona ~200ms de latência (depois armazenada no cache)
- Requer conexão ativa com servidor
- Permite composições em visualização móvel
- Melhor consistência de dados

O servidor expande automaticamente as composições durante a prévia:

1. Solicitações de aplicativos móveis /projects/:id/preview

2. Servidor carrega todas as telas e composições
3. Para cada tela, expande recursivamente os elementosCompositionInstance
4. Retorna dados expandidos com UUIDs únicos por instância

```
// Before expansion
{
  type: "CompositionInstance",
  properties: { compositionId: "comp-123" }
}

// After expansion (server-side)
{
  type: "Flex",
  uuid: "instance-456-element-789", // Prefixed with instance UUID
  properties: { direction: "column" },
  children: [...]
}
```

---

8 - Validação e Segurança

O sistema inclui várias salvaguardas:

Detecção de Referência Circular

```
// This is prevented:
Composition A contains Composition B
Composition B contains Composition A
// ✕ Error: "Circular reference detected"
```

Validação da Profundidade de Aninhamento

- Profundidade máxima: 5 níveis
- Aviso em profundidade >= 4

- Blocos adicionando composições que excedem o limite

Detecção de Composição Órfã

- Detecta composições deletadas ainda referenciadas nas telas
- Mostra faixa de aviso com contagem
- Atualiza automaticamente a cada 30 segundos

---

9 - Encadernações e Eventos

As ligações (eventos e propriedades) funcionam corretamente com instâncias de composição:

1. O servidor duplica vinculações para cada elemento expandido
2. Gera IDs de vinculação únicos: `{instanceUuid}-{originalBindingId}`
3. UUIDs de elementos são rastreados no mapeamento: `{instanceUuid}-{elementUuid}`

```
// Original binding in composition
{
  id: "binding-1",
  elementId: "button-123",
  eventType: "click",
  functionId: "func-456"
}

// Expanded binding in instance
{
  id: "instance-789-binding-1",
  elementId: "instance-789-button-123",
  eventType: "click",
  functionId: "func-456"
}
```

---

10 - Melhores Práticas

Prática	Descrição
Modular Design	Mantenha as composições pequenas e focadas em responsabilidades únicas
Convenção de nomeação	Use nomes descritivos: , , `serCardProductListNavigationMenu`
Evite nidificações profundas	Fique dentro de 2-3 níveis para melhor desempenho
Instâncias de Teste	Verifique todos os casos após editar uma redação
Usar categorias	Organize composições por categoria (ui, layout, formas)

---

11 - Solução de problemas

Composição Não Aparece na Visualização do Celular

1. Verifique se a composição existe no banco de dados
2. Verifique está correto `CompositionInstance` `compositionId`
3. Certifique-se de que o celular esteja buscando do servidor (não usando dados em cache)
4. Verifique os logs do servidor para erros de expansão

Avisos de Chave Duplicada

- Não deve ocorrer após as atualizações de dezembro de 2025
- Atualmente, UUIDs são precedidos por UUID de instância

Fixações Não Funcionando

- Verifique se a duplicação de binding do lado do servidor está funcionando
- Verifique se os UUIDs dos elementos correspondem na tabela de ligações
- Confirme que o ID da função é válido

---

12 - Endpoints de API

Método	Endpoint	Descrição
OBTER	`/projects/:id/compositions`	Liste todas as composições de um projeto
OBTER	`/projects/:id/compositions/:uuid`	Obtenha uma composição individual
POSTAR	`/projects/:id/compositions`	Crie nova composição
PATCH	`/projects/:id/compositions/:uuid`	Composição atualizada
DELETE	`/projects/:id/compositions/:uuid`	Excluir composição
OBTER	`/projects/:id/preview`	Obtenha dados expandidos do projeto para prévia

---

13 - Esquema de Banco de Dados

```sql
CREATE TABLE lowcode_studio.compositions (
  uuid UUID PRIMARY KEY,
  project_id TEXT NOT NULL,
  name VARCHAR(255) NOT NULL,
  category VARCHAR(100),
  description TEXT,
  icon VARCHAR(50),
  elements JSONB NOT NULL DEFAULT '[]',
  properties JSONB,
  styles JSONB,
  order INTEGER DEFAULT 0,
  created_at BIGINT DEFAULT extract(epoch from now()) * 1000,
  updated_at BIGINT,
  deleted_at BIGINT
);

CREATE INDEX idx_compositions_project
  ON lowcode_studio.compositions(project_id);
```

---

14 - Arquitetura Técnica

Lado do Cliente (Web Studio)

- React Query para busca de composição
- Sinais de Preact para estado reativo
- Edição de modo duplo (tela vs composição)
- Salvamento automático com dequique de 2 segundos

Lado do Servidor (API)

- Estrutura Deno + Hono
- Drizzle ORM com PostgreSQL
- Expansão de composição recursiva
- Mapeamento UUID para duplicação de ligações

Prévia Móvel

- Buscas a partir do endpoint `/projects/:id/preview`
- Recebe dados de composição totalmente expandidos
- Renderizações usando lowcode-engine-tauri

---

15 - Migração a partir de versões anteriores

Projetos criados antes das composições são totalmente compatíveis:

- JSONs de projetos antigos importam sem problemas
- O sistema lida com elegância com dados de composição ausentes
- Sem mudanças significativas para projetos existentes
- Somente novos projetos podem criar composições

---

16 - Considerações de Desempenho

Web Studio

- Composições carregam sob demanda
- Tempo de 5 minutos para armazenamento de composição
- Invalidação automática do cache em atualizações

Prévia Móvel

- Carga inicial: ~200ms de sobrecarga para expansão
- Carregamentos subsequentes: armazenados em cache pelo React Query
- A expansão acontece uma vez por sessão de pré-visualização

Recomendações

- Mantenha as composições abaixo de 50 elementos
- Limite o aninhamento a 3 níveis para desempenho ideal
- Use categorias de composição para uma melhor organização
- Limpeza regular de composições não utilizadas

17 - Segurança

- Suporte a múltiplas inquilinações via isolamento de esquema de banco de dados
- Autenticação baseada em token para endpoints de API
- Validação do lado do servidor de todos os dados de composição
- Proteção contra ataques circulares de referência
- Prevenção de injeção SQL via Drizzle ORM

---

18 - Melhorias Futuras

Melhorias planejadas:

- Variantes de composição (temas claro/escuro)
- Biblioteca de modelos de composição
- Composições de importação/exportação entre projetos
- Histórico de versões das composições
- Análise de composição (acompanhamento de uso)
- Ferramenta de diferença de composição visual
- Mercado de composições

---

19 - Suporte e Recursos

Para ajuda adicional:

- Verifique os logs do servidor para erros de expansão
- Use o DevTools do navegador para inspecionar os dados de composição
- Verifique o esquema do banco de dados com `deno task db:check`
- Consulte o guia de implementação em `docs/en-us/platform/compositions.md`

 Ative o registro de depuração definindo variáveis de ambiente para ver logs detalhados de expansão de composição. `DEBUG=true`

4.4. Bloqueios de Recursos

1 - Visão geral

O sistema Resource Lock oferece proteção colaborativa de edição ao impedir que múltiplos usuários editem simultaneamente a mesma tela ou composição. Isso garante a integridade dos dados e previne conflitos em ambientes multiusuário.

Principais Recursos

- **Gerenciamento Automático de Bloqueios:** Recursos são automaticamente bloqueados quando abertos para edição
- **Indicadores Visuais:** Feedback claro da interface mostrando quem está editando o quê
- **Proteção contra Tempo Morto:** Bloqueios expiram automaticamente após 5 minutos de inatividade
- **Mecanismo de Batimentos Cardíacos:** Sessões de edição ativa renovam automaticamente os bloqueios
- **Liberação forçada:** Administradores podem liberar bloqueios à força quando necessário
- **Atualizações em tempo real:** Atualizações de status do bloqueio via mecanismo de polling
- **Desabilitação Abrangente:** Todas as superfícies de edição são desativadas quando bloqueadas por outro usuário

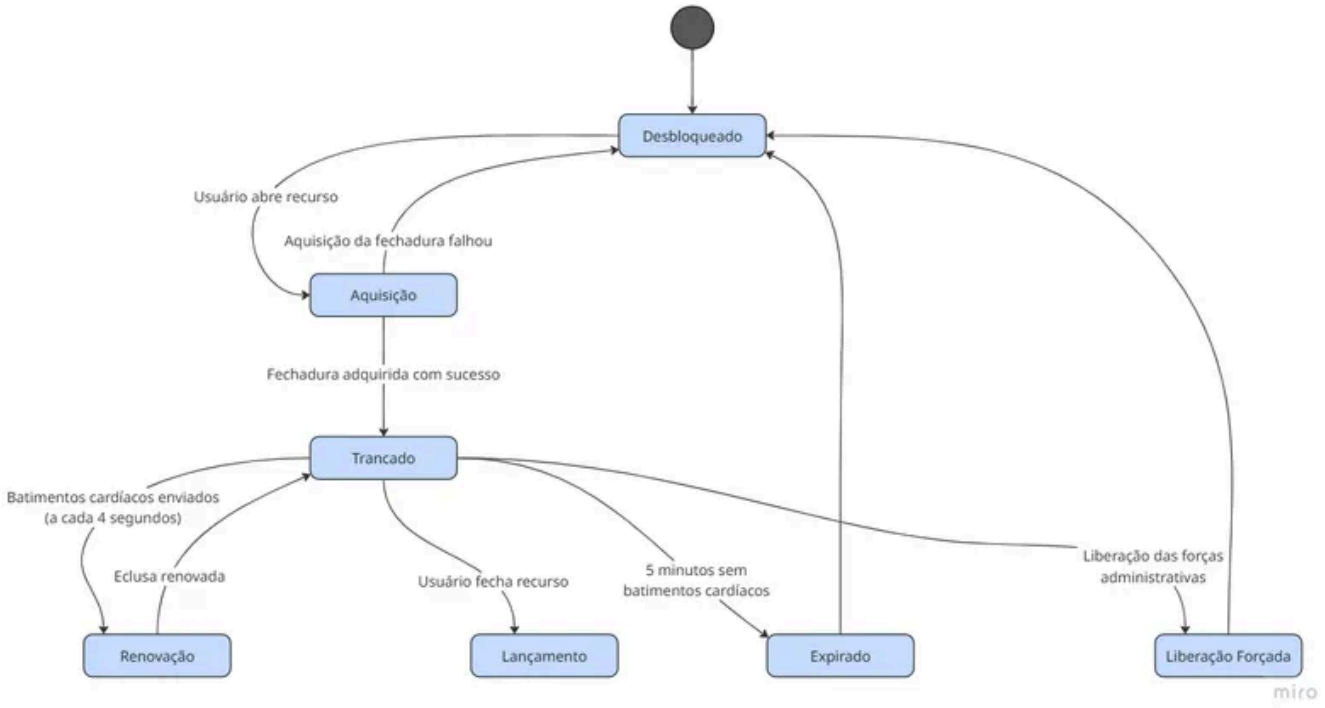
Recursos Suportados

- **Screens:** Screens individuais de aplicação
- **Composições:** Composições componentes reutilizáveis

---

2 - Como Funciona

Ciclo de Vida da Trava



Fluxo de Aquisição de Locks

1. **Usuário Abre Recurso**: O usuário navega até uma tela ou composição
2. **Solicitação de Bloqueio**: A interface envia a solicitaçãoPOST /api/locks/acquire
3. **Verificação de Bloqueio**: O backend verifica se o recurso já está bloqueado
4. **Conceda ou negue**:
    - Se desbloqueado: O bloqueio é concedido e armazenado no banco de dados
    - Se bloqueado pelo mesmo usuário: O bloqueio é concedido (reentrada permitida)
    - Se bloqueado por outro usuário: Solicitação negada com detalhes do bloqueio
5. **Batimentos cardíacos iniciados**: O frontend começa a enviar batimentos cardíacos a cada 4 segundos
6. **Atualizações da interface**: O estado travado é refletido em todos os componentes da interface

Mecanismo do Batimento Cardíaco

O mecanismo do batimento cardíaco mantém as travas ativas durante a edição ativa:

- **Intervalo**: A cada 4 segundos
- **Ponto final**: `POST /api/locks/renew`
- **Propósito**: Estende o tempo de expiração do bloqueio

- **Automático:** Executa em segundo plano enquanto o recurso está aberto
- **Limpeza:** Para quando o usuário navega para fora ou fecha o recurso

Expiração da Fechadura

Fechaduras expiram automaticamente após 5 minutos sem batimentos cardíacos:

- **Tempo:** 300 segundos (5 minutos)
- **Propósito:** Evitar que travas obsoletas bloqueiem recursos
- **Limpeza Automática:** O backend remove automaticamente fechaduras vencidas
- **Período de Carência:** Garante que bloqueios não persistam caso o usuário feche o navegador inesperadamente

---

3 - Implementação Backend

Esquema de Banco de Dados

```
CREATE TABLE lowcode_studio.resource_locks (
  id TEXT PRIMARY KEY,
  resource_type TEXT NOT NULL,  -- 'screen' or 'composition'
  resource_id TEXT NOT NULL,
  locked_by TEXT NOT NULL,      -- User ID
  locked_at BIGINT NOT NULL,    -- Unix timestamp in milliseconds
  expires_at BIGINT NOT NULL,   -- Unix timestamp in milliseconds
  UNIQUE(resource_type, resource_id)
);

CREATE INDEX idx_resource_locks_resource
ON lowcode_studio.resource_locks(resource_type, resource_id);

CREATE INDEX idx_resource_locks_expires
ON lowcode_studio.resource_locks(expires_at);
```

API Endpoints

Adquirir Lock

```
POST /api/locks/acquire
Content-Type: application/json

{
  "resourceType": "screen",
  "resourceId": "uuid-here",
  "userId": "user-id-here"
}
```

Resposta (Sucesso - 200):

```
{
  "success": true,
  "lock": {
    "id": "lock-id",
    "resourceType": "screen",
    "resourceId": "uuid-here",
    "lockedBy": "user-id-here",
    "lockedByUser": {
      "id": "user-id-here",
      "firstName": "John",
      "lastName": "Doe"
    },
    "lockedAt": 1737648000000,
    "expiresAt": 1737648300000
  }
}
```

Resposta (Já Bloqueada - 409):

```
{
  "success": false,
  "error": "Resource is locked by another user",
  "lock": {
    "id": "lock-id",
    "resourceType": "screen",
    "resourceId": "uuid-here",
    "lockedBy": "other-user-id",
    "lockedByUser": {
      "id": "other-user-id",
      "firstName": "Jane",
      "lastName": "Smith"
    },
    "lockedAt": 1737647900000,
    "expiresAt": 1737648200000
  }
}
```

Renovar Trava

```
POST /api/locks/renew
Content-Type: application/json

{
  "resourceType": "screen",
  "resourceId": "uuid-here",
  "userId": "user-id-here"
}
```

Resposta (200):

```
{
  "success": true,
  "expiresAt": 1737648600000
}
```

Trava de Liberação

```
POST /api/locks/release
Content-Type: application/json

{
  "resourceType": "screen",
  "resourceId": "uuid-here",
  "userId": "user-id-here"
}
```

Resposta (200):

```
{
  "success": true
}
```

Trava de Liberação Forçada

```
POST /api/locks/force
Content-Type: application/json

{
  "resourceType": "screen",
  "resourceId": "uuid-here"
}
```

Resposta (200):

```
{
  "success": true
}
```

Conseguir Fechaduras por Recurso

```
GET /api/locks?resourceType=screen&resourceId=uuid-here
```

Resposta (200):

```
{
  "locks": [
    {
      "id": "lock-id",
      "resourceType": "screen",
      "resourceId": "uuid-here",
      "lockedBy": "user-id",
      "lockedByUser": {
        "id": "user-id",
        "firstName": "John",
        "lastName": "Doe"
      },
      "lockedAt": 1737648000000,
      "expiresAt": 1737648300000
    }
  ]
}
```

---

4 - Implementação Frontend

Arquitetura de Loja

O sistema de bloqueio utiliza sinais Preact para gerenciamento de estado reativo:

Arquivo: `client/src/features/project/stores/resource-locks.ts`

```
// Global signals
export const resourceLocks = signal<Map<string, ResourceLock>>(new Map());
export const lockHeartbeats = signal<Map<string, number>>(new Map());

// Computed signal for editing state
export const isEditingDisabled = computed(() => {
  const currentActive = editMode.value === "composition"
    ? activeComposition.value
    : activeScreen.value;

  if (!currentActive) return false;

  const lockStatus = getResourceLockStatus(
    editMode.value === "composition" ? "composition" : "screen",
    currentActive.uuid,
    userProfile.value?.id || ""
  );

  return lockStatus.isLocked && !lockStatus.isOwnedByCurrentUser;
});
```

Funções de Trava

adquireResourceLock

Tentativas de obter um bloqueio em um recurso:

```
const result = await acquireResourceLock('screen', screenUuid, userId);
if (result.success) {
  // Lock acquired, start heartbeat
} else {
  // Lock denied, show notification
}
```

renewResourceLock

Renova uma fechadura existente (chamada por batimentos cardíacos):

```
await renewResourceLock('screen', screenUuid, userId);
```

releaseResourceLock

Libera um bloqueio quando o usuário termina de editar:

```
await releaseResourceLock('screen', screenUuid, userId);
```

getResourceLockStatus

Verifica o status atual do bloqueio de um recurso:

```
const status = getResourceLockStatus('screen', screenUuid, userId);
// Returns: { isLocked, isOwnedByCurrentUser, lock }
```

Componentes da interface

Notificações do Canvas

Arquivo: `client/src/features/project/details/components/layout/canvas-notifications.tsx`

Exibe notificações de bloqueio e composições órfãs no topo da tela:

- **Notificação de Bloqueio:** Mostra quem está editando o recurso
- **Avatar do Usuário:** Avatar codificado por cores com base no ID do usuário
- **Exibição de Horas:** Mostra quando a fechadura foi adquirida
- **Ícone de Cadeado:** Indicador visual no final do banner

Lista de Tela

Arquivo: `client/src/features/project/details/components/left-panel/screen-list.tsx`

Mostra indicadores de bloqueio ao lado das telas bloqueadas:

- **Ícone de Cadeado:** Ícone pequeno de cadeado ao lado do nome de usuário
- **Avatar do Usuário:** Avatar minúsculo mostrando quem está com o bloqueio
- **Dica de ferramenta:** Passe o mouse para ver o nome completo do usuário

Árvore de Elementos

Arquivo: `client/src/features/project/details/components/left-panel/elements-tree/`

Desativa manipulação de elementos quando bloqueado:

- **Botão de Adicionar:** Oculto quando bloqueado
- **Arrastar e Soltar:** Desativado quando bloqueado
- **Menu de Contexto:** Vazio quando bloqueado
- **Reordenamento:** Impedido quando travado

Painel Direito

Arquivo: `client/src/features/project/details/components/right-panel/index.tsx`

Torna as abas legíveis, mas o conteúdo não editável:

- **Abas :** Permanecam clicáveis para navegação
- **Conteúdo:** Desativado com pointer-events-none opacity-60
- **Todos os Painéis:** Propriedades, Estilos, Encadernações, Documentação, Configurações

Ações de Tela

Arquivo: `client/src/features/project/details/components/renderer/index.tsx`

Ocultar botões de ação de elemento quando bloqueados:

- **Mover-se para cima/baixo:** Oculto
- **Excluir:** Oculto
- **Cópia:** Oculta
- **Largura do Rótulo:** Definido para 0 quando bloqueado

Estratégia de Pesquisa

O sistema utiliza polling adaptativo para verificar o status do bloqueio:

```
// With active locks: Poll every 5 seconds
// Without locks: Poll every 30 seconds
const refetchInterval = hasActiveLocks ? 5000 : 30000;

useQuery({
  queryKey: ['resource-locks', projectId],
  queryFn: () => fetchResourceLocks(projectId),
  refetchInterval,
  staleTime: 0,
});
```

---

5 - Experiência do Usuário

Ao abrir um recurso bloqueado

1. **Navegação:** Usuário clica em uma tela/composição bloqueada
2. **Checagem de Bloqueio:** Sistema verifica se o recurso está bloqueado
3. **Notificação:** Banner aparece no topo da tela
4. **UI desativada:** Todos os controles de edição estão desativados
5. **Modo Somente Leitura:** O usuário pode visualizar, mas não editar

Indicadores Visuais

Faixa de Fechadura (Tela)

```
┌──────────────────────────────────────────────────────────┐
│ John Doe is editing this screen (2 minutes ago)          │
└──────────────────────────────────────────────────────────┘
```

Características: * Fundo vermelho () * Avatar do usuário com fundo codificado por cores * Exibição do nome completo * Tempo desde que a fechadura foi adquirida * Ícone de cadeado no final `bg-red-50 border-red-200`

Indicador de Lista de Tela

```
Home Screen        [JD]
Profile Screen
Settings Screen    [JS]
```

Características: * Ícone pequeno de cadeado ao lado do nome * Avatar de usuário pequeno (8px) * Dica de ferramenta ao passar o mouse mostrando o nome completo

Quando outro usuário adquire o bloqueio

1. **Detecção de Polling:** Frontend detecta novo lock via polling
2. **Atualização da interface:** Indicadores de trava aparecem imediatamente
3. **Edição desativada:** Todos os controles se tornam não interativos
4. **Edições Atuais:** As alterações locais do usuário permanecem, mas não podem ser salvas
5. **Notificação:** Notificação opcional de toast (não implementada atualmente)

Quando a Tranca é Liberada

1. **Detecção de Liberação:** O sondamento detecta remoção de trava
2. **Remoção de notificações:** Banner de bloqueio desaparece
3. **UI Ativada:** Todos os controles de edição ficam ativos

4. **Auto-Travamento**: Se o usuário ainda estiver no recurso, ele pode adquirir o bloqueio

---

6 - Configuração

Tempo de Espera do Bloqueio

Altere o tempo de validade da fechadura:

Backend (): `server/src/routes/locks/handlers.ts`

```
const LOCK_TIMEOUT_MS = 5 * 60 * 1000; // 5 minutes
```

Intervalo de Batimentos cardíacos

Altere a frequência com que os cadeados são renovados:

Frontend (): `client/src/features/project/stores/resource-locks.ts`

```
const HEARTBEAT_INTERVAL = 4000; // 4 seconds
```

Intervalo de votação

Alterem a frequência de pesquisa de status do bloqueio:

Frontend (): `client/src/features/project/queries/resource-locks.ts`

```
const WITH_LOCKS_INTERVAL = 5000;   // 5 seconds
const WITHOUT_LOCKS_INTERVAL = 30000; // 30 seconds
```

---

7 - Solução de problemas

Trava não soltando

Sintomas: Usuário fechou o navegador, mas o bloqueio persiste

Solução: Espere 5 minutos pela expiração automática ou use liberação forçada:

```
curl -X POST http://localhost:8000/api/locks/force \
  -H "Content-Type: application/json" \
  -d '{
    "resourceType": "screen",
    "resourceId": "uuid-here"
  }'
```

Múltiplos usuários veem diferentes estados de bloqueio

Sintomas: Usuários veem informações conflitantes sobre bloqueios

Causa: Intervalo de pesquisa muito longo ou problemas de rede

Soluções:

1. Reduzir o intervalo de votação na produção.
2. Verifique a conectividade da rede.
3. Verificar se o backend está respondendo.
4. Verifique a conexão com o banco de dados.

Bloqueio adquirido, mas interface não atualizando

Sintomas: O bloqueio existe no banco de dados, mas a interface mostra desbloqueada

Causa: Problema com o cache de React Query ou sinal que não está sendo atualizado

Soluções:

1. Verifique o console do navegador para erros.

2. Verificar é chamado em componentes.
3. Limpar o cache de Consultas React.
4. Atualizar a página `useSignals()`

Batimentos cardíacos não enviando

Sintomas: O bloqueio expira enquanto o usuário está editando ativamente

Causa: Intervalo de batimentos cardíacos parou ou problemas na rede

Soluções:

1. Verifique o console do navegador para erros de rede.
2. Verificar se o usuário ainda está autenticado.
3. Verifique se o intervalo está sendo liberado incorretamente.
4. Verificar se o endpoint backend está funcionando `/api/locks/renew`

As cores dos avatares são idênticas

Sintomas: Usuários diferentes têm a mesma cor de avatar

Causa: Colisão de hash no algoritmo de geração de cor

Solução: Isso foi corrigido com o hash rolante polinomial. Se ainda ocorrer:

1. Verificar se o código mais recente foi implantado
2. Função de verificação
3. Garantir que está sendo passado para o componente `generateAvatarColor()userIdUserAvatar`

---

8. Melhores Práticas

Para desenvolvedores

1. **Sempre Limpe:** Garanta que as travas sejam liberadas quando o componente for desmontado
2. **Erros de Manipulação:** Gerencie com elegância falhas de aquisição de travas
3. **Use Sinal Calculado:** Referência em vez de recalcularisEditingDisabled
4. **Teste de Concorrência:** Teste com múltiplos usuários editando simultaneamente
5. **Monitore o desempenho:** Fique atento a excesso de polling ou chamadas API

Para usuários

1. **Fechar Trancas de Liberação:** Feche recursos ao terminar a edição
2. **Não force o fechamento:** Use a navegação adequada para garantir a limpeza
3. **Aguarde a expiração:** Se a trava estiver presa, espere 5 minutos
4. **Comunique:** Coordena com a equipe quem edita o quê
5. **Atualize se necessário:** Se a interface parecer travada, atualize a página

Para Administradores

1. **Fechaduras de monitor:** Verifique periodicamente se estão desgastadas
2. **Definir Tempos Razoáveis:** Equilíbrio entre usabilidade e disponibilidade de recursos
3. **Liberação Forçada com Parcimónia:** Use apenas quando necessário
4. **Manutenção do Banco de Dados:** Limpeza regular de registros de fechaduras vencidas
5. **Análise de Log:** Monitorar padrões de aquisição de bloqueios para detectar problemas

---

9 - Melhorias Futuras

Características Planejadas

- **Suporte a WebSocket:** Notificações de bloqueio em tempo real em vez de sondagem

- **Roubo de Fechaduras:** Permitir que administradores forcem a aquisição de travas
- **Histórico de Bloqueios:** Rastreie quem travou recursos e quando
- **Análise de Fechaduras:** Monitore os padrões de uso das fechaduras
- **Cursores Colaborativos:** Mostre onde outros usuários estão trabalhando
- **Resolução de Conflitos:** Mesclar mudanças conflitantes quando múltiplos usuários editam
- **Solicitações de Trava:** Solicita liberação do bloqueio do atual detentor
- **Fila de Bloqueio:** Usuários em fila aguardando um recurso travado

Otimizações de desempenho

- **Redis Cache:** Armazene bloqueios no Redis para acesso mais rápido
- **Assinaturas GraphQL:** Atualizações em tempo real via assinaturas
- **Bloqueio Otimista:** Permitir edição com detecção de conflitos
- **Pool de Bloqueios:** Agrupar múltiplos recursos sob um único bloqueio
- **Pesquisa Inteligente:** Ajuste a frequência das pesquisas com base na atividade

4.5. Desenvolvimento

Desenvolvimento

1 - Pré-requisitos

- Docker e Docker Compose instalados
- Tempo de execução Bun instalado (**https://bun.sh**)
- Repositório Lowcode Studio clonado
- Repositório Tauri do Motor Lowcode clonado (opcional, para funcionalidade de pré-visualização)

---

2 - Instalação

2.1 - Dependências de Instalação

A partir da raiz do projeto, instale todas as dependências do workspace:

```
bun install
```

2.2 - Iniciar Serviços de Infraestrutura

Lowcode Studio requer PostgreSQL e Redis. A é fornecido na raiz do projeto:docker-compose.yml

```
docker compose up -d
```

Começa assim:

- **PostgreSQL 17** na porta (usuário: , senha: `5432` `run2biz` `run2biz`)
- **PgBouncer** na porta (pooler de conexão para PostgreSQL, modo transação) `6432`
- **Redis 7** na porta (senha: `6379` `run2biz`)

-

> A API se conecta via PgBouncer (porta) em vez de diretamente ao PostgreSQL (porta). O PgBouncer oferece pooling de conexões no modo de transação, essencial para cargas de trabalho multi-inquilino. `6432` `5432`

2.3 - Configuração do Ambiente

Configuração do servidor

Crie um arquivo no diretório após o modelo. `.env/` `server` `.env` `.example`

Para desenvolvimento local, configure o banco de dados e URLs do Redis apontando para via PgBouncer: `localhost`

```
DATABASE_URL=postgresql://run2biz:run2biz@localhost:6432/postgres
REDIS_URL=redis://default:run2biz@localhost:6379
```

> **Não** inclua no — o PgBouncer no modo de transação não suporta parâmetros de inicialização. O esquema de Drizzle já é qualificado com , então todas as consultas são prefixadas automaticamente. `search_path` `DATABASE_URL` `pgSchema("lowcode_studio")`
>
> Migrações de banco de dados usam que reescrevem a porta de para conectar diretamente ao PostgreSQL (contornando o PgBouncer), já que precisa do parâmetro de inicialização. `drizzle.config.ts` `6432` `5432` `drizzle-kit` `search_path`

> Ao rodar dentro do Docker (por exemplo, via Edge-runtime), use os nomes de host dos contêineres:
>
> ```
> DATABASE_URL=postgresql://run2biz:run2biz@pgbouncer:6432/postgres
> REDIS_URL=redis://default:run2biz@redis:6379
> ```

Configuração do Cliente

Crie um arquivo no diretório após o modelo. `.env` `/client` `.env` `.example`

Para desenvolvimento local, o cliente aponta diretamente para a API do servidor:

```
VITE_API_URL=http://localhost:8080/api
VITE_ENGINE_URL=http://localhost:1420/tauri-engine
```

Variável	Descrição	Padrão
`VITE_API_URL`	URL completa para o servidor API do Lowcode Studio	`/lowcode-studio-api/api` (produção)
`VITE_ENGINE_URL`	URL onde a Tauri Engine é servida	`/tauri-engine` (produção)

 Em produção, ambos e usam caminhos relativos porque a API e o motor são servidos pela mesma origem em tempo de execução de borda. No desenvolvimento local, eles devem apontar para as URLs reais dos serviços, já que não há proxy reverso.

`VITE_API_URL` `VITE_ENGINE_URL`

2.4 - Executar migrações de banco de dados

Após iniciar o PostgreSQL, execute as migrações do banco de dados:

```
cd server
bun run db:migrate
```

2.5 - Iniciar Servidores de Desenvolvimento

A partir da raiz do projeto, inicie tanto o cliente quanto o servidor simultaneamente:

```
bun run dev
```

Começa assim:

- **Servidor** (Hono API): `http://localhost:8080`
- **Cliente** (Vite): `http://localhost:4000`

Você também pode começar individualmente:

```
bun run dev:client  # Vite dev server on port 4000
bun run dev:server  # Hono server with --hot flag on port 8080
```

Iniciar o motor Tauri (opcional)

Para usar o recurso de pré-visualização móvel, inicie o servidor de desenvolvimento do Tauri Engine em seu próprio repositório:

```
cd /path/to/lowcode-engine-tauri
bun run dev
```

Isso serve ao motor em `.http://localhost:1420/tauri-engine`

Veja Integração do Motor para detalhes da configuração do Motor Tauri.

2.6. Definir o Cookie de Autenticação

A aplicação requer um token JWT válido. Defina o cookie no seu navegador para `.HYPER-AUTH-TOKENlocalhost:4000`

Você pode fazer isso via `DevTools` do navegador:

1. Abra no seu navegador **http://localhost:4000**
2. Abrir DevTools → Cookies de Aplicação → → **http://localhost:4000**

3. Adicione um cookie com nome e um token JWT válido como valor `HYPER-AUTH-TOKEN`

Alternativamente, se o tempo de execução de borda estiver disponível, use o endpoint set-cookie:

```
http://localhost:3000/set-cookie?token=YOUR_TOKEN_HERE
```

2.7. Acesse ao Lowcode Studio

Quando ambos os serviços estiverem em execução, acesse o Lowcode Studio em:

```
http://localhost:4000
```

---

3. Notas de Arquitetura

3.1. Stack de Middleware de Servidor

O middleware da API executa nesta ordem:

1. `prettyJSON` → → `trimTrailingSlashcors`
2. `/health` endpoint (não autenticado)
3. `/webhooks` Rotas (não autenticadas)
4. `setTenantDb` → validateToken
5. Rotas de domínio (, , , `/components/projects/permissions/users`)

> O middleware CORS é colocado antes da autenticação para garantir que as solicitações de pré-voo tenham sucesso. Isso é essencial quando o cliente e o servidor rodam em origens diferentes (por exemplo, e).OPTIONSlocalhost:4000localhost:8080

3.2. Solicitações de Origem Cruzada

No desenvolvimento local, o cliente () faz requisições diretas ao servidor (). Isso exige: `localhost:4000localhost:8080`

- **Servidor:** CORS configurado com origem dinâmica e `credentials: true`
- **Cliente:** Todas as solicitações de busca incluem enviar cookies de origem cruzada `credentials: "include"`
- **Tauri Engine:** Usa em seu cliente HTTP pelo mesmo motivo `withCredentials: true`

Em produção, todos os serviços estão atrás da mesma origem (tempo de execução de borda), então a origem cruzada não é um problema.

---

4. Implantação via tempo de execução de borda

Para implantação via Edge-runtime (configuração semelhante a produção), consulte a integração original Edge-runtime:

4.1. Configurar o Volume do Docker

No runtime do Edge, adicione o volume correspondente para o servidor Lowcode Studio: `docker-compose.yml`

```
volumes:
  - ../edge-functions/lowcode-studio/server:/home/deno/functions/lowcode-studio-api/1.0.(
```

4.2. Iniciar Serviços de Execução em Borda

Navegue até o diretório raiz Edge-runtime e execute:

```
docker-compose up -d --build
```

 Ao iniciar, o tempo de execução da borda irá automaticamente:

- Executar migrações de banco de dados
- Configure o esquema necessário para o Lowcode Studio
- Preencher dados iniciais de seed

Uma vez em execução, acesse o Lowcode Studio via Edge-runtime em:

```
http://localhost:3000
```

---

5. Solução de problemas

5.1. Problemas de Conexão com Banco de Dados

Se você encontrar erros na conexão do banco de dados:

1. Verifique se o PostgreSQL está rodando: `docker compose ps`
2. Confirme que o parâmetro está corretamente definido em `search_pathDATABASE_URL`
3. Verifique se o esquema existe no banco de dados `lowcode_studio`
4. Certifique-se de estar usando (desenvolvedor local) ou o nome de host Docker (container) `localhost:5432`

5.2. Falhas na migração

Se as migrações falharem em executar:

1. Verifique se o PostgreSQL está acessível: `docker compose logs postgres`
2. Verificar credenciais de banco de dados em `server/.env`
3. Certifique-se de que o formato da URL do banco de dados esteja correto
4. Execute migrações manualmente: `cd server && bun run db:migrate`

5.3. Problemas de Conexão com o Cliente

Se o cliente não conseguir se conectar à API:

1. Verifique se o servidor está rodando na porta 8080
2. Verifique os pontos de entrada para `VITE_API_URLclient/.envhttp://localhost:8080/api`
3. Verifique o console do navegador para erros CORS — certifique-se de que o middleware CORS do servidor esteja antes da autenticação
4. Verifique se o cookie está configurado para o domínio correto `HYPER-AUTH-TOKEN`

5.4. Prévia Não Funcionando

Se a prévia móvel mostrar "Projeto não encontrado":

1. Verifique se o Motor Tauri está rodando em **`http://localhost:1420/tauri-engine`**
2. Pontos de verificação para `VITE_ENGINE_URLclient/.env` **`http://localhost:1420/tauri-engine`**
3. Verifique se a Tauri Engine tem e `.env` `VITE_API_URL=http://localhost:8080/api` `VITE_ENGINE_MODE=PREVIEW`
4. Verifique no console do navegador para erros de CORS ou autenticação
5. Certifique-se de que o cliente HTTP do Tauri Engine esteja ativado `withCredentials: true`

4.6. Implantação

1. Imagem de Construção

```
./scripts/build.sh
./scripts/publish.sh -r portus
```

2. Implantar Aplicação

```
helm install lowcode-studio charts/studio \
  -f charts/studio/values.yaml \
  -f charts/studio/values-oxygen.yaml \
  --kubeconfig ~/.kube/oxygen.yaml \
  --namespace 01-hyper-apps \
  --set image.tag=1.0.0 \
  --set env.DATABASE_URL=postgresql://run2biz:run2biz@host.docker.internal:5432/postgres
```

3. Desinstalar o Aplicativo

```
helm uninstall lowcode-studio \
  --kubeconfig ~/.kube/oxygen.yaml \
  --namespace=01-hyper-apps
```

4. Variáveis Ambientais

```
# Required for nexus deploy
NEXUS_URL=nexus.centralit.io:9091
NEXUS_USER=run2biz
NEXUS_PASS=nexus...

# Required for portus deploy
PORTUS_URL=registry.cloud4biz.com
PORTUS_USER=automacao-cdi
PORTUS_PASS=UZ4...
```

---

5. Integração CI/CD do GitLab

O pipeline CI/CD do GitLab automatiza o processo de construção e implantação tanto para versões web quanto mobile do aplicativo studio. Ele se integra com o Expo para builds móveis e gerencia a implantação na web.

5.1. Arquitetura de Pipeline Estágios do Pipeline

O pipeline CI/CD consiste em uma única etapa de construção abrangente que executa múltiplos passos sequenciais:

Palco	Descrição
`build_and_deploy`	Processo completo de construção e implantação, incluindo download de esquemas, configuração de dependências, configuração de projetos, geração de ativos e implantação específica da plataforma

Fluxo de Trabalho de Processo de Construção

O processo de compilação executa os seguintes 6 passos sequenciais via o script: `01-build-and-deploy.sh`

Passo 1: Download do Esquema

- Baixa o esquema do projeto a partir do endpoint da API
- Salva o esquema para o aplicativo usar `schema.json`
- Usa ID do projeto e token de autenticação para acesso seguro

Passo 2: Configuração de Dependência

- Configura dependências dinâmicas com base nos requisitos do projeto
- Atualizações com bibliotecas especificadas na configuração do projeto `package.json`
- Gerencia configurações específicas da biblioteca e plugins

Passo 3: Instalação de Dependência

- Instala todas as dependências necessárias usando o gerenciador de pacotes Bun
- Garante que todas as bibliotecas e plugins estejam disponíveis para a build

Passo 4: Configuração do Projeto

- Configura com configurações específicas do projeto `app.json`
- Configura configurações específicas de plataforma (Android/iOS)
- Gerencia códigos de versão, identificadores de pacotes e configurações da Expo
- Configura o roteamento da URL base se especificado

Passo 5: Geração de Ativos

- Baixa imagens específicas do projeto (tela de inicialização e ícone)
- Busca ativos da API usando requisições autenticadas
- Salva os ativos nos diretórios apropriados

Passo 6: Implantação da Plataforma

- **Configuração de Webhook:** Configura webhooks da Expo para notificações de compilação
- **Build específico para plataforma:**
- **Mobile (Android/iOS):** Aciona a construção EAS com perfis específicos de plataforma
- **Web:** Desenvolve aplicações web e implanta na hospedagem da Expo
- **Rastreamento de builds:** Atualiza o status da build via callbacks da API

Configuração

Variáveis Ambientais Obrigatórias

As seguintes variáveis de ambiente devem ser configuradas nas configurações do seu projeto no GitLab:

Variáveis de Integração do GitLab (devem ser configuradas no mapa-configuração do servidor)

Variável	Descrição	Exemplo/Notas
`GITLAB_PROJECT_ID`	Identificador único do projeto de motor lowcode	ID numérico nas configurações do projeto GitLab
`GITLAB_PIPELINE_TOKEN`	Token de autenticação para acionar pipelines	Gerar no GitLab → Configurações de projeto → Triggers de Pipeline de → CI/CD
`GITLAB_WEBHOOK_TOKEN`	Token de segurança para validação de webhooks	String gerada aleatoriamente para segurança
`GITLAB_URL`	URL base da sua instância do GitLab	`http://gitlab.home` ou seu domínio do GitLab

Build Process Variables (está nas variáveis do pipeline)

Variável	Descrição	Exemplo/Notas
`API_URL`	URL base da API lowcode-studio	Usado para download de esquemas e atualizações de compilação
`TOKEN`	Token de autenticação para acesso à API	Token portador para comunicação segura de API
`PROJECT_ID`	Identificador do projeto alvo para build	Projeto específico em construção
`BUILD_ID`	Identificador único de build	Usado para acompanhar o progresso da construção
`BUILD_NUMBER`	Número de compilação sequencial	Usado para versão Android Código e Número de compilação para iOS
`PLATFORM`	Plataforma alvo para implantação	`web` , , ou `android` `ios`
`PROFILE`	Configuração do perfil de construção	`development` ou `production`
`VALUES`	Objeto de configuração JSON	Contém configurações de projeto, expo, android, ios e biblioteca

Variáveis de Integração Expo

Variável	Descrição	Exemplo/Notas
`EXPO_WEBHOOK_TOKEN`	Token secreto para validação de webhooks na Expo	Usado para notificações de compilação e envio
`EAS_NO_VCS`	Desativa integração do VCS no CLI EAS	Definido como no ambiente CI `1`

5.2. Geração de Tokens

Pipeline Token

1. Navegue até seu projeto de lowcode no GitLab
2. Vá em Configurações → Tokens de Acesso
3. Crie um novo token com escopo `api`
4. Copie o token gerado para `GITLAB_PIPELINE_TOKEN`

Webhook Token

Gerar uma string aleatória segura para validação do webhook:

```
openssl rand -hex 32
```

5.3. Detalhes do Roteiro

Visão Geral dos Roteiros Principais

O pipeline utiliza 6 scripts especializados localizados em: **apps/engine/scripts/**

Roteiro	Propósito	Funções-chave
`01-build-and-deploy.sh`	Roteiro principal de orquestração	Coordena todas as etapas de build e cuida da implantação específica da plataforma
`02-configure-dependencies.js`	Gerenciamento dinâmico de dependências	Atualizações package.json com bibliotecas e plugins específicos de projetos
`03-configure-project.js`	Configuração do projeto	Configura app.json com configurações de plataforma, códigos de versão e configuração da Expo
`04-generate-images.js`	Gestão de ativos	Baixa telas de abertura e ícones específicos de cada projeto a partir da API
`05-check-webhooks.sh`	Gerenciamento de Webhooks	Garante que os webhooks da Expo estejam devidamente configurados para notificações de compilação
`06-deploy-expo.sh`	Implantação da plataforma	Cuida das builds de EAS para implantação móvel e web para hospedagem em Expo

Comportamento Específico da Plataforma

Plataforma Web

- Constrói aplicação web usando `bun run build:web`
- Implanta na hospedagem da Expo usando EAS CLI
- Gerencia a configuração da URL base para roteamento
- Retorna URL de implantação para rastreamento

Plataformas Móveis (Android/iOS)

- Gatilhos EAS construídos com perfis específicos de plataforma
- Suporta auto-submissão para builds de produção
- Gerencia códigos de versão e números de compilação
- Retorna URL de build para monitoramento

Webhooks

O pipeline usa webhooks para notificar o servidor sobre o status da build.

Para cada inquilino, precisamos criar um webhook no gitlab para notificar o servidor sobre o status da build.

Exemplo:

Propriedade	Valor
URL	**https://tenant.cloud4biz.com.br/lwc-studio/api/webhooks/gitlab/projects**
token secreto	`GITLAB_WEBHOOK_TOKEN`
Gatilho	[eventos do oleoduto]
Verificação SSL	habilitado

Corredores

O pipeline deve ter um runner para executá-lo.

Artefatos Gerados

O pipeline gera os seguintes artefatos: - - Construção de aplicação web (apenas plataforma web) - - Configurações configuradas da aplicação Expo - - Atualizado com dependências dinâmicas - - Esquema de projeto para uso em tempo de execução :sectnums: :secnumníveis: 3dist/app.jsonpackage.jsonschema.json

---

6. Configuração MinIO

O estúdio usa MinIO exclusivamente para armazenar arquivos APK para o arquivo `.dev-client`

O APK deve ser solicitado à equipe de desenvolvimento `.dev-client`

6.1. Variáveis do Ambiente

Configure o MinIO usando as seguintes variáveis de ambiente:

Variável	Descrição	Exemplo
`MINIO_ACCESS_KEY`	Chave de acesso MinIO para autenticação	`UfT…`
`MINIO_SECRET_KEY`	Chave secreta MinIO para autenticação	`jX2c…`
`MINIO_BUCKET`	Nome do balde alvo para armazenamento	`lwc-studio`
`MINIO_ENDPOINT`	Endpoint do servidor MinIO	`minio.home`
`MINIO_PORT`	Porta para servidor MinIO	`0`
`MINIO_USE_SSL`	Habilitar conexão SSL/TLS	`true`

6.2. Exemplo de Configuração

```
MINIO_ACCESS_KEY=UfT...
MINIO_SECRET_KEY=jX2c...
MINIO_BUCKET=lwc-studio
MINIO_ENDPOINT=minio.home
MINIO_PORT=0
MINIO_USE_SSL=true
```

6.3. Exemplo de Desenvolvimento Local

Para uma instância local de MinIO rodando sobre HTTP simples, use valores semelhantes a:

```
MINIO_ACCESS_KEY=minioadmin
MINIO_SECRET_KEY=minioadmin
MINIO_BUCKET=lwc-studio
MINIO_ENDPOINT=localhost
MINIO_PORT=9000
MINIO_USE_SSL=false
```

O Studio não exige que o diretório de dados MinIO esteja dentro do repositório. Você pode iniciar o MinIO com qualquer caminho gravável, por exemplo:

```
mkdir -p ~/minio-data
minio server ~/minio-data --console-address ":9001"
```

Usar um diretório externo ajuda a manter o armazenamento local de objetos separado da base de código e evita preencher o volume do workspace acidentalmente. :sectnums: :secnumníveis: 3server/

---

7. Configuração do Inquilino

O estúdio exige que os seguintes componentes sejam configurados no front manager:

- **Aplicação:** Deve ser criada no front manager
- **Versão do aplicativo e menu:** Obrigatório para acesso ao gerente de frente
- **Associação de Funções:** O cardápio deve ter uma função associada para acesso do gerente de frente
- **Inquilino de Recursos:** Fornece strings de conexão ao banco de dados para cada locatário. Sem isso, o estúdio usa por padrão a variável ambiente `DATABASE_URL`

7.1. Configuração

`RESOURCE_TENANT_NAME` é o nome da aplicação que deve ser configurada na variável de ambiente e será usada para obter a string de conexão com o banco de dados.

Configure o locatário de recursos no front manager em: `hyper-admin/resource-tenant/resource-manager`

Inquilino de Recursos

Configuração da conexão com banco de dados:

Campo	Descrição	Exemplo
`host`	Endereço do host do banco de dados	`pgbouncer.<namespace-kubernetes>`
`port`	Número da porta do banco de dados	`6432`
`databaseName`	Nome do banco de dados alvo	`<db-name>`
`username`	Nome de usuário do banco de dados	`<db-username>`
`password`	Senha do banco de dados	`<db-password>`
`dbms`	Sistema de gerenciamento de banco de dados	`postgres`
schema	Esquema de banco de dados	`lowcode_studio`

Aplicação

Configurações de registro de inscrição:

Campo	Descrição	Valor
`name`	Identificador de aplicação	`@hyper/lwc-studio`
`description`	Nome de exibição do aplicativo	`Lowcode Studio`
`type`	Tipo de gerenciamento de aplicações	`Managed by platform`
`render_type`	Método de renderização	`Legacy`
`status`	Estado da aplicação	`Enabled`

Versão do Aplicativo

A versão do aplicativo pode ser configurada com qualquer valor.

Cardápio

O menu pode ser configurado com qualquer valor.

Papel no Cardápio

Deve corresponder ao valor da variável ambiente. `ADMIN_ROLE`

Solução de problemas

O Studio armazena em cache a string de conexão do banco de dados quando tentar obter a string de conexão do front manager no REDIS.

Campo	Exemplo
Chave	`edge-runtime::resource-tenants::<RESOURCE_TENANT_NAME>:<TENANT_ID>`

Se a string de conexão de banco de dados não estiver correta, você pode limpar o cache excluindo os in redis. :sectnums: :secnumníveis: 3 :imagesdir: docs/imageskey

---

8. Integração do Motor

Existem dois motores suportados: - Expo - Tauri (trabalho em andamento)

8.1. Tauri

Visão geral

O motor Tauri exibe prévias móveis de projetos do Lowcode Studio. Ele roda como um iframe dentro do editor Studio e se comunica via `.postMessage`

O fluxo de comunicação entre Studio e Tauri Engine:

1. Studio renderiza o motor em e usando o `<iframe>` `VITE_ENGINE_URL`
2. O motor envia para a janela mãe `engine:loaded`
3. Studio responde com contendo `studio:schema:get` `{ projectId, initialScreenId }`
4. O motor busca dados do projeto da API em `GET /projects/{projectId}/preview`
5. O motor renderiza o projeto e envia para o pai `engine:ready`

> O Studio não envia dados completos do projeto via . Ele envia apenas o , e o Engine busca os dados completos do servidor API. Isso garante que as composições sejam expandidas no lado do servidor `.postMessage` `projectId`

Configuração do Desenvolvimento Local

Para rodar o Motor Tauri localmente para desenvolvimento:

1 - Clone o repositório `lowcode-engine-tauri`

2 - Dependências de instalação:

```
cd lowcode-engine-tauri
bun install
```

3 - Configure o arquivo:.env

```
VITE_API_URL=http://localhost:8080/api
VITE_ENGINE_MODE=PREVIEW
```

Variável Descrição Padrão

Variável	Descrição	Padrão
`VITE_API_URL`	URL do servidor API do Lowcode Studio	`/api`
`VITE_ENGINE_MODE`	Modo do motor: (busca da API) ou (lê JSON local) `PREVIEWPRODUCTION`	`PRODUCTION`

4 - Comece o servidor de desenvolvimento:

```
bun run dev
```

O motor estará disponível em **`http://localhost:1420/tauri-engine`**

5 - No Lowcode Studio , defina a URL do motor: `client/.env`

```
VITE_ENGINE_URL=http://localhost:1420/tauri-engine
```

Configuração de Origem Cruzada

Quando rodando localmente, o Motor Tauri () faz requisições de API para o servidor Studio (). Essa configuração de origem cruzada exige:localhost:1420 `localhost:8080`

- O cliente HTTP do Engine deve ser usado para enviar cookies de autenticação `withCredentials: true`
- O servidor Studio deve ter CORS configurado com uma origem dinâmica `credentials: true`

O Engine usa Axios (baseados em busca) configurados em: `redaxios` `src/helpers/api.ts`

```
export const api = axios.create({ baseURL: API_URL, withCredentials: true });
```

Implantação em Produção (MinIO)

Implantações Tauri são entregues por meio de um tempo de execução de borda e exigem que arquivos estáticos sejam armazenados no MinIO.

A equipe de motores fornecerá os arquivos estáticos necessários.

Para implantar o motor Tauri:

- Crie uma pasta chamada dentro do bucket no MinIO. `studio-engine-tauri` `webapps`
- Dentro dessa pasta, crie uma subpasta correspondente à versão do motor Tauri (por exemplo,). `1.0.0`
- Faça upload do conteúdo fornecido do motor para esta subpasta versionada.

Solução de problemas

Sintoma	Solução
"Projeto não encontrado" em prévia	Verifique se isso aponta para a URL correta da API e está definido `VITE_API_URL` `VITE_ENGINE_MODE=PREVIEW`
Erros de CORS no console do navegador	Garantir que o cliente API do Engine e o CORS estejam no servidor `withCredentials: true` `credentials: true`
O motor mostra o spinner de carregamento indefinidamente	Verifique o fluxo: verifique o console do navegador para mensagens e `postMessage` `engine:loaded` `studio:schema:get`
401 Não autorizado no pedido de prévia	O cookie não está sendo enviado. Verificação e configuração do CORS `HYPER-AUTH-TOKEN` `withCredentials`