# Guia Definitivo para Desenvolvimento de Plugins para Ulanzi Studio

## Introdução

Este guia detalhado tem como objetivo fornecer instruções precisas e melhores práticas para o desenvolvimento de plugins para o Ulanzi Studio, garantindo que eles sejam reconhecidos e funcionem corretamente na plataforma. Baseado em extensas análises de plugins originais, documentação disponível e testes práticos, este documento visa capacitar desenvolvedores (humanos ou IA) a criar plugins compatíveis e funcionais.

O Ulanzi Studio, assim como outras plataformas de controladores de streaming, utiliza uma estrutura específica para seus plugins. A compreensão e adesão a essa estrutura são cruciais para o sucesso do reconhecimento e carregamento do plugin pelo software. As lições aprendidas durante o processo de portabilidade de plugins do Stream Deck para o Ulanzi Studio foram incorporadas para oferecer um caminho claro e eficaz.

## 1. Estrutura Essencial do Plugin

Para que um plugin seja reconhecido pelo Ulanzi Studio, ele deve seguir uma estrutura de diretórios e nomenclatura muito específica. A pasta raiz do plugin deve ser nomeada seguindo o padrão com.ulanzi.ulanzideck.<nome\_do\_plugin>.ulanziPlugin.

Dentro desta pasta raiz, os seguintes diretórios e arquivos são essenciais:

* **manifest.json**: O arquivo de configuração principal do plugin. Ele define metadados, ações, ícones e caminhos para os arquivos de código e Property Inspectors.
* **assets/**: Contém todos os ícones e imagens utilizados pelo plugin. É fundamental que os caminhos para esses assets sejam corretamente referenciados no manifest.json e nos arquivos HTML.
* **plugin/**: Este diretório contém o arquivo HTML principal do plugin (app.html), que será carregado pelo Ulanzi Studio para a lógica principal do plugin e a interface do botão.
* **property-inspectors/**: Este diretório contém subpastas para cada ação definida no manifest.json. Cada subpasta deve conter um arquivo HTML (inspector.html) que define a interface de configuração daquela ação.
* **libs/**: Contém as bibliotecas JavaScript e CSS fornecidas pelo SDK do Ulanzi, essenciais para a comunicação entre o plugin e o Ulanzi Studio. A ausência ou referências incorretas a esses arquivos são uma causa comum de falha no reconhecimento do plugin.

**Exemplo de Estrutura de Diretórios:**

com.ulanzi.ulanzideck.meuplugin.ulanziPlugin/

├── manifest.json

├── assets/

│ ├── pluginIcon.png

│ ├── actionIcon.png

│ └── ...

├── plugin/

│ └── app.html

├── property-inspectors/

│ ├── minhaacao/

│ │ └── inspector.html

│ └── outraacao/

│ └── inspector.html

└── libs/

├── css/

│ └── udpi.css

└── js/

├── ulanzideckApi.js

└── ...

É crucial que a pasta libs seja copiada de um plugin de demonstração funcional do Ulanzi SDK (como o com.ulanzi.analogclock.ulanziPlugin encontrado no repositório UlanziDeckPlugin-SDK/demo/) para garantir que todas as dependências corretas estejam presentes e que as referências nos arquivos HTML estejam apontando para os locais corretos.

## 2. Configuração do manifest.json

O arquivo manifest.json é o coração do seu plugin, definindo como o Ulanzi Studio o reconhece e interage com ele. A sintaxe é JSON e deve ser rigorosamente seguida. Abaixo estão os campos essenciais e suas configurações, baseados em plugins originais e funcionais:

{

"UUID": "com.ulanzi.ulanzideck.nome\_do\_plugin",

"Name": "Nome do Plugin",

"Category": "Categoria",

"Author": "Seu Nome/Empresa",

"Version": "1.0.0",

"Description": "Uma breve descrição do seu plugin.",

"Icon": "assets/pluginIcon.png",

"CodePath": "plugin/app.html",

"Type": "JavaScript",

"Actions": [

{

"UUID": "com.ulanzi.ulanzideck.nome\_do\_plugin.nome\_da\_acao",

"Name": "Nome da Ação",

"Icon": "assets/actionIcon.png",

"States": [

{

"Image": "assets/actionIcon.png",

"TitleAlignment": "middle",

"FontSize": "17"

}

],

"SupportedInMultiActions": false,

"Tooltip": "Descrição da ação.",

"PropertyInspectorPath": "property-inspectors/nome\_da\_acao/inspector.html"

}

]

}

**Explicação dos Campos Chave:**

* **UUID (Plugin)**: Identificador único do seu plugin. Deve seguir o padrão com.ulanzi.ulanzideck.<nome\_do\_plugin>. É crucial que seja único e consistente com o nome da pasta raiz do plugin (sem o .ulanziPlugin final).
* **Name**: Nome amigável do plugin que aparecerá no Ulanzi Studio.
* **Category**: Categoria sob a qual o plugin será listado no Ulanzi Studio (ex: "System", "Audio", "Productivity").
* **Author**: Seu nome ou o nome da sua organização.
* **Version**: Versão atual do seu plugin.
* **Description**: Uma breve descrição da funcionalidade do plugin.
* **Icon**: Caminho relativo para o ícone principal do plugin, exibido na lista de plugins do Ulanzi Studio. Deve apontar para um arquivo dentro da pasta assets/.
* **CodePath**: **Este é um campo crítico.** Ele define o arquivo principal que o Ulanzi Studio carregará para a lógica do plugin. Para plugins baseados em HTML/JavaScript, o valor **deve ser plugin/app.html**. Mesmo que a lógica principal seja JavaScript, o Ulanzi Studio espera um arquivo HTML como ponto de entrada para o plugin.
* **Type**: **Outro campo crítico.** Para plugins baseados em HTML/JavaScript, o valor **deve ser JavaScript**. Isso informa ao Ulanzi Studio o tipo de ambiente de execução esperado.
* **Actions**: Um array de objetos, onde cada objeto representa uma ação que o seu plugin pode realizar e que pode ser arrastada para um botão no Ulanzi Deck.
  + **UUID (Ação)**: Identificador único para cada ação. Deve seguir o padrão com.ulanzi.ulanzideck.<nome\_do\_plugin>.<nome\_da\_acao>.
  + **Name (Ação)**: Nome amigável da ação.
  + **Icon (Ação)**: Caminho relativo para o ícone da ação, exibido no Ulanzi Studio. Deve apontar para um arquivo dentro da pasta assets/.
  + **States**: Define os estados visuais da ação. Geralmente, um único estado é suficiente para a maioria dos plugins. O Image aponta para o ícone da ação, e TitleAlignment e FontSize controlam a exibição de texto no botão.
  + **PropertyInspectorPath**: Caminho relativo para o arquivo HTML que define a interface de configuração para esta ação. Deve seguir o padrão property-inspectors/<nome\_da\_acao>/inspector.html.

**Melhores Práticas para manifest.json:**

* **Consistência:** Mantenha os UUIDs e caminhos consistentes com a estrutura de diretórios e nomes de arquivos. Erros de digitação aqui são uma causa comum de falha no reconhecimento.
* **Caminhos Relativos:** Todos os caminhos (Icon, CodePath, PropertyInspectorPath) devem ser relativos à pasta raiz do plugin.
* **Validação:** Embora o Ulanzi Studio não forneça uma ferramenta de validação de manifest.json pública, seguir o formato de plugins originais é a melhor forma de garantir a conformidade.

## 3. Desenvolvimento dos Arquivos HTML e JavaScript

Os arquivos HTML (app.html e inspector.html) são a interface do seu plugin e o ponto de interação com a API do UlanziDeck. Eles são essencialmente páginas web que rodam dentro do Ulanzi Studio.

### 3.1. plugin/app.html (Lógica Principal do Plugin)

Este arquivo é o ponto de entrada visual e lógico do seu plugin para cada botão no Ulanzi Deck. Ele é responsável por exibir o estado atual do botão e se comunicar com a API do UlanziDeck e, se necessário, com serviços externos (como o servidor de monitoramento que desenvolvemos).

**Estrutura Básica:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Nome do Plugin</title>

<link rel="stylesheet" href="../libs/css/udpi.css" />

<style>

/\* Estilos CSS específicos para o seu botão \*/

body { margin: 0; padding: 0; overflow: hidden; }

.button-content {

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

height: 100vh;

background: #1a1a1a;

color: white;

font-family: Arial, sans-serif;

font-size: 14px;

}

.value { font-size: 24px; font-weight: bold; }

.label { font-size: 12px; opacity: 0.8; }

</style>

</head>

<body>

<div class="button-content">

<div class="value" id="displayValue">--</div>

<div class="label" id="displayLabel">Carregando...</div>

</div>

<script src="../libs/js/ulanzideckApi.js"></script>

<script>

// Variáveis globais para a API do UlanziDeck

let globalSettings = {};

let globalActionInfo = {};

let globalUUID = null;

// Função de inicialização da API do UlanziDeck

// Esta função é chamada pelo Ulanzi Studio quando o plugin é carregado

function connectElgatoStreamDeckSocket(inPort, inUUID, inMessageType, inApplicationInfo, inActionInfo) {

globalUUID = inUUID;

globalActionInfo = JSON.parse(inActionInfo);

// globalSettings = globalActionInfo.payload.settings; // Se houver configurações salvas

console.log("Plugin carregado para ação: ", globalActionInfo.action);

// Exemplo de como se comunicar com o Property Inspector (se necessário)

// ulanzideck.on("sendToPlugin", (event) => {

// const payload = event.payload;

// if (payload.type === "updateSettings") {

// globalSettings = payload.settings;

// // Atualizar a interface do botão com as novas configurações

// }

// });

// Exemplo de como atualizar o título/imagem do botão

// ulanzideck.setTitle(globalUUID, "Novo Título");

// ulanzideck.setImage(globalUUID, "data:image/png;base64,...");

// Sua lógica de comunicação com serviços externos (ex: WebSocket)

// Exemplo de conexão WebSocket (adaptado do System Monitor):

const WS\_URL = `ws://localhost:3002`; // Ou o IP/porta do seu servidor

let ws = null;

let reconnectInterval = null;

function connectWebSocket() {

ws = new WebSocket(WS\_URL);

ws.onopen = () => { console.log("WS Conectado"); /\* Atualizar UI \*/ };

ws.onmessage = (event) => {

const message = JSON.parse(event.data);

if (message.type === "system\_data") {

// Lógica para atualizar displayValue e displayLabel

const actionId = globalActionInfo.action;

let value = "--";

let label = "Monitor";

switch (actionId) {

case "com.ulanzi.ulanzideck.systemmonitor.cpu":

value = message.data.cpu.usage + "%";

label = "CPU";

break;

case "com.ulanzi.ulanzideck.systemmonitor.memory":

value = message.data.memory.percentage + "%";

label = "RAM";

break;

case "com.ulanzi.ulanzideck.systemmonitor.gpu":

value = message.data.gpu.usage + "%";

label = "GPU";

break;

}

document.getElementById("displayValue").textContent = value;

document.getElementById("displayLabel").textContent = label;

}

};

ws.onclose = () => { console.log("WS Desconectado"); /\* Tentar reconectar \*/ };

ws.onerror = (error) => { console.error("WS Erro:", error); };

}

connectWebSocket();

}

// Garante que a função de conexão seja chamada quando o DOM estiver pronto

document.addEventListener("DOMContentLoaded", () => {

// O Ulanzi Studio injeta a função connectElgatoStreamDeckSocket globalmente

// e a chama. Não é necessário chamá-la manualmente aqui.

});

</script>

</body>

</html>

**Pontos Chave para app.html:**

* **Inclusão de udpi.css e ulanzideckApi.js**: Estes arquivos são essenciais para o estilo básico e para a comunicação com o Ulanzi Studio. Eles devem ser referenciados corretamente via ../libs/css/udpi.css e ../libs/js/ulanzideckApi.js.
* **connectElgatoStreamDeckSocket**: Esta função é o ponto de entrada para a comunicação do plugin com o Ulanzi Studio. Ela é chamada automaticamente pelo Ulanzi Studio e recebe informações importantes como inUUID (identificador único da instância do botão) e inActionInfo (informações sobre a ação configurada).
* **ulanzideck API**: Dentro de connectElgatoStreamDeckSocket, você terá acesso ao objeto ulanzideck (ou streamDeck em algumas versões da API) que permite interagir com o Ulanzi Studio, como ulanzideck.setTitle(), ulanzideck.setImage(), ulanzideck.on() para eventos, etc.
* **Lógica de Exibição**: Use JavaScript para manipular o DOM (displayValue, displayLabel) e exibir as informações relevantes no botão.
* **Comunicação Externa**: Se o seu plugin precisa de dados de um serviço externo (como o servidor de monitoramento), a lógica de conexão (ex: WebSocket, Fetch API) deve ser implementada aqui.

### 3.2. property-inspectors/<nome\_da\_acao>/inspector.html (Interface de Configuração)

Este arquivo define a interface que o usuário verá no Ulanzi Studio para configurar uma ação específica do seu plugin. Ele também se comunica com a API do UlanziDeck para salvar e carregar configurações.

**Estrutura Básica:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Configurações da Ação</title>

<link rel="stylesheet" href="../../libs/css/udpi.css" />

</head>

<body>

<div class="sdpi-wrapper">

<div class="sdpi-item">

<div class="sdpi-item-label">Exemplo de Configuração</div>

<div class="sdpi-item-value">

<input class="sdpi-item-value-field" type="text" id="mySetting" value="">

</div>

</div>

<div class="sdpi-item">

<div class="sdpi-item-label">Status do Servidor</div>

<div class="sdpi-item-value">

<span id="serverStatus">Verificando...</span>

</div>

</div>

<!-- Outros elementos de UI para configurar a ação -->

</div>

<script src="../../libs/js/ulanzideckApi.js"></script>

<script>

let globalSettings = {};

let globalUUID = null;

// Função de inicialização da API do UlanziDeck para o Property Inspector

function connectElgatoStreamDeckSocket(inPort, inUUID, inMessageType, inApplicationInfo, inActionInfo) {

globalUUID = inUUID;

const actionInfo = JSON.parse(inActionInfo);

globalSettings = actionInfo.payload.settings;

console.log("Property Inspector carregado para ação: ", actionInfo.action);

// Carregar configurações salvas na UI

document.getElementById("mySetting").value = globalSettings.mySetting || "";

// Exemplo de como salvar configurações

document.getElementById("mySetting").addEventListener("change", (event) => {

globalSettings.mySetting = event.target.value;

ulanzideck.setSettings(globalUUID, globalSettings);

// Opcional: enviar para o plugin principal para atualização imediata

// ulanzideck.sendToPlugin(globalUUID, { type: "updateSettings", settings: globalSettings });

});

// Lógica para verificar o status de um servidor externo (adaptado do System Monitor)

const serverStatus = document.getElementById("serverStatus");

function checkServerStatus() {

fetch("http://localhost:3001/api/status") // Ou o IP/porta do seu servidor

.then(response => response.json())

.then(data => {

serverStatus.textContent = "Conectado ✓";

serverStatus.style.color = "#4CAF50";

})

.catch(error => {

serverStatus.textContent = "Servidor não rodando ✗";

serverStatus.style.color = "#f44336";

});

}

checkServerStatus();

setInterval(checkServerStatus, 5000);

}

</script>

</body>

</html>

**Pontos Chave para inspector.html:**

* **Inclusão de udpi.css e ulanzideckApi.js**: Assim como no app.html, são essenciais para estilo e comunicação.
* **connectElgatoStreamDeckSocket**: Também é o ponto de entrada para o Property Inspector. Recebe as configurações salvas da ação (actionInfo.payload.settings).
* **ulanzideck.setSettings()**: Esta função é usada para salvar as configurações modificadas pelo usuário. O Ulanzi Studio as persistirá e as enviará de volta ao app.html quando o botão for carregado.
* **ulanzideck.sendToPlugin()**: Permite enviar mensagens diretamente do Property Inspector para a instância do plugin (app.html) associada àquela ação, útil para atualizações em tempo real sem precisar salvar as configurações.
* **Elementos de Formulário**: Use elementos HTML padrão (<input>, <select>, etc.) para criar a interface de configuração. O udpi.css fornece estilos básicos para esses elementos.

## 4. Empacotamento e Instalação do Plugin

Após desenvolver a estrutura e os arquivos do seu plugin, o próximo passo é empacotá-lo corretamente e instalá-lo no Ulanzi Studio. Este processo é crucial para que o software reconheça e carregue seu plugin.

### 4.1. Empacotamento (Criação do Arquivo ZIP)

O Ulanzi Studio espera que os plugins sejam distribuídos como arquivos ZIP. O conteúdo do arquivo ZIP deve ser a pasta raiz do seu plugin.

**Passos para Empacotar:**

1. **Navegue até o diretório pai** da pasta raiz do seu plugin. Por exemplo, se a pasta do seu plugin é com.ulanzi.ulanzideck.meuplugin.ulanziPlugin, navegue até o diretório que contém essa pasta.
2. **Compacte a pasta raiz do plugin.** Certifique-se de que o arquivo ZIP contenha diretamente a pasta raiz do plugin, e não um diretório pai adicional. O nome do arquivo ZIP deve ser o mesmo nome da pasta raiz do plugin, com a extensão .zip.
   * **Exemplo (Linha de Comando - Linux/macOS):**

cd /caminho/para/seus/plugins

zip -r com.ulanzi.ulanzideck.meuplugin.ulanziPlugin.zip com.ulanzi.ulanzideck.meuplugin.ulanziPlugin

* + **Exemplo (Linha de Comando - Windows PowerShell):**

Set-Location C:\caminho\para\seus\plugins

Compress-Archive -Path .\com.ulanzi.ulanzideck.meuplugin.ulanziPlugin -DestinationPath .\com.ulanzi.ulanzideck.meuplugin.ulanziPlugin.zip

* + **Exemplo (Interface Gráfica - Windows):**
    1. Clique com o botão direito na pasta com.ulanzi.ulanzideck.meuplugin.ulanziPlugin.
    2. Selecione "Enviar para" > "Pasta compactada (zipada)".
    3. Renomeie o arquivo ZIP resultante para com.ulanzi.ulanzideck.meuplugin.ulanziPlugin.zip se necessário.

### 4.2. Instalação Manual

O Ulanzi Studio não possui um instalador de plugins integrado para arquivos ZIP. A instalação é feita manualmente, copiando a pasta do plugin (descompactada) para um diretório específico.

**Passos para Instalar:**

1. **Localize o diretório de plugins do Ulanzi Studio.** Este diretório geralmente está localizado em:
   * **Windows:** C:\Users\<SeuNomeDeUsuário>\AppData\Roaming\Ulanzi\UlanziDeck\Plugins
   * **macOS:** ~/Library/Application Support/Ulanzi/UlanziDeck/Plugins (o diretório Library pode estar oculto; você pode acessá-lo via Finder > Ir > Ir para Pasta... e digitar ~/Library)
2. **Descompacte o arquivo ZIP do seu plugin.** Isso criará a pasta raiz do seu plugin (ex: com.ulanzi.ulanzideck.meuplugin.ulanziPlugin).
3. **Copie a pasta raiz do plugin** (a pasta descompactada, não o arquivo ZIP) para o diretório de plugins do Ulanzi Studio identificado no passo 1.
4. **Reinicie o Ulanzi Studio.** É fundamental reiniciar o software para que ele possa escanear o diretório de plugins e carregar as novas adições.

Após reiniciar, seu plugin deve aparecer na lista de plugins disponíveis no Ulanzi Studio, pronto para ser arrastado para um botão do seu Ulanzi Deck.

## 5. Depuração e Solução de Problemas

Mesmo seguindo todas as instruções, é comum encontrar problemas durante o desenvolvimento de plugins. Esta seção aborda os problemas mais frequentes e como depurá-los.

### 5.1. O Plugin Não Aparece no Ulanzi Studio

Este é o problema mais comum e geralmente indica um erro na estrutura do plugin ou no manifest.json.

* **Verifique o Nome da Pasta Raiz:** Certifique-se de que a pasta raiz do seu plugin segue o padrão com.ulanzi.ulanzideck.<nome\_do\_plugin>.ulanziPlugin e que o <nome\_do\_plugin> corresponde ao UUID principal no manifest.json (sem o .ulanziPlugin).
* **Verifique o manifest.json:**
  + **Sintaxe JSON:** Use um validador JSON online para garantir que não há erros de sintaxe (vírgulas ausentes, chaves não fechadas, etc.).
  + **UUID:** O UUID principal e os UUIDs das ações devem ser únicos e seguir o padrão com.ulanzi.ulanzideck.<nome\_do\_plugin> e com.ulanzi.ulanzideck.<nome\_do\_plugin>.<nome\_da\_acao>, respectivamente.
  + **CodePath:** Deve ser plugin/app.html.
  + **Type:** Deve ser JavaScript.
  + **Caminhos de Ícones e Property Inspectors:** Verifique se os caminhos em Icon e PropertyInspectorPath estão corretos e apontam para os arquivos existentes dentro da estrutura do seu plugin.
* **Estrutura de Diretórios:** Confirme que os diretórios assets/, plugin/, property-inspectors/ e libs/ estão presentes e contêm os arquivos esperados nos locais corretos.
* **Pasta libs/:** Certifique-se de que a pasta libs/ foi copiada de um plugin de demonstração funcional do Ulanzi SDK e que contém todos os arquivos necessários (udpi.css, ulanzideckApi.js, etc.).
* **Instalação:** Verifique se a pasta descompactada do plugin foi copiada para o diretório correto do Ulanzi Studio (C:\Users\<SeuNomeDeUsuário>\AppData\Roaming\Ulanzi\UlanziDeck\Plugins no Windows) e se o Ulanzi Studio foi reiniciado.

### 5.2. O Plugin Aparece, mas Não Funciona ou Exibe Erros

Se o plugin é reconhecido, mas não se comporta como esperado, o problema geralmente está na lógica JavaScript ou nas referências de arquivos.

* **Console do Desenvolvedor:** O Ulanzi Studio é baseado em tecnologias web, o que significa que você pode usar as ferramentas de desenvolvedor do navegador para depurar seu plugin. Embora o Ulanzi Studio não exponha diretamente um console, você pode tentar:
  + **console.log():** Use console.log() extensivamente em seu JavaScript (app.html e inspector.html) para imprimir mensagens e valores de variáveis. Essas mensagens podem aparecer em logs do sistema operacional ou em alguma ferramenta de depuração interna do Ulanzi Studio (se disponível).
  + **Alerts:** Para depuração rápida, `alert(

"Minha Mensagem")` pode ser usado para verificar se o código está sendo executado em determinados pontos, embora seja intrusivo.

* **Verifique as Referências de Scripts e Estilos:** Certifique-se de que todos os arquivos JavaScript e CSS estão sendo carregados corretamente. Erros no console do navegador (se acessível) ou no log do sistema podem indicar que um arquivo não foi encontrado (404 Not Found). Verifique os caminhos relativos em suas tags <script> e <link>.
  + Exemplo de caminho correto para ulanzideckApi.js em app.html: ../libs/js/ulanzideckApi.js
  + Exemplo de caminho correto para ulanzideckApi.js em inspector.html: ../../libs/js/ulanzideckApi.js
* **Comunicação com o Servidor Intermediário:** Se você estiver usando um servidor intermediário (como o que desenvolvemos para o monitoramento de sistema), verifique:
  + **Servidor Rodando:** Confirme que o servidor está ativo e acessível no IP e porta configurados.
  + **Firewall:** Verifique se o firewall do seu sistema operacional não está bloqueando as conexões para as portas usadas pelo servidor (ex: 8123).
  + **CORS:** Certifique-se de que o servidor intermediário está configurado para permitir requisições Cross-Origin Resource Sharing (CORS) do seu plugin. O servidor que desenvolvemos já inclui essa configuração.
  + **Logs do Servidor:** Monitore os logs do seu servidor intermediário para identificar erros de conexão ou processamento de dados.

### 5.3. Problemas de Funcionalidade Específica

Quando o plugin aparece e se comunica, mas uma funcionalidade específica não funciona, o problema geralmente está na lógica de programação.

* **Lógica JavaScript:** Revise seu código JavaScript em app.html e inspector.html. Verifique se a manipulação do DOM está correta, se os eventos estão sendo tratados adequadamente e se a lógica de comunicação com a API do UlanziDeck ou serviços externos está implementada conforme o esperado.
* **API do UlanziDeck:** Consulte a documentação da API do UlanziDeck (se disponível) ou o código-fonte de plugins originais para entender como usar corretamente as funções como ulanzideck.setTitle(), ulanzideck.setImage(), ulanzideck.setSettings(), etc.
* **API Externa:** Se o plugin interage com uma API externa (como a do Soundpad), verifique a documentação dessa API para garantir que as requisições e respostas estão sendo tratadas corretamente.

## 6. Conclusão

O desenvolvimento de plugins para o Ulanzi Studio, embora não seja amplamente documentado publicamente, é totalmente viável seguindo as convenções e estruturas corretas. A chave para o sucesso reside na replicação precisa da estrutura de diretórios, na configuração correta do manifest.json e na inclusão das bibliotecas do SDK do Ulanzi. Para funcionalidades que exigem acesso a recursos do sistema operacional (como monitoramento de CPU/RAM/GPU ou controle de aplicações via named pipe), a criação de um componente intermediário (servidor local) é uma solução robusta e necessária.

Este guia, construído a partir de experiências práticas e engenharia reversa de plugins funcionais, serve como um roteiro para superar os desafios iniciais e criar plugins que são reconhecidos e funcionais no Ulanzi Studio. Com paciência e atenção aos detalhes, você pode expandir significativamente as capacidades do seu Ulanzi Deck.

## Referências

[1] UlanziDeckPlugin-SDK GitHub Repository. Disponível em: <https://github.com/UlanziTechnology/UlanziDeckPlugin-SDK>

[2] Ulanzi Studio Plugin Usage Guide. Disponível em: <https://www.ulanzi.com/pages/downloads> (Referência ao documento fornecido pelo usuário)

[3] Ulanzi Studio Forum Thread. Disponível em: <https://bbs.ulanzistudio.com/forum.php?mod=viewthread&tid=64>

[4] Soundpad API Documentation. Disponível em: <https://www.leppsoft.com/soundpad/en/rc/documentation.html>

[5] Seblor/Soundpaddon GitHub Repository. Disponível em: <https://github.com/Seblor/Soundpaddon>

[6] systeminformation npm package. Disponível em: <https://www.npmjs.com/package/systeminformation>