# Cielo Pay - Mini Apps

v7.0

Data: 11/08/2020

# Documentação Mini Apps - Cielo Pay

## Sumário

**Sumário** **1**

**O que é o Mini App** **3**

**Especificações** **3**

Integração com a Web View 4

Integração com Módulo JS 4

**Integração Mini App e Cielo Pay** **4**

**Instalação** **5**

**Utilização do Gateway** **6**

**Fluxos de integração** **8**

Inicialização 8

Autenticação 9

Pagamento e Reembolso 10

Status Pagamento 11

Escanear código 12

Localização 13

Modal Loading e Fechar Mini App 14

**Funcionalidades** **15**

Inicialização - init() 15

Autenticação - askMeStartAuth() 17

Pagamento - startPaymentsFlow() 18

Reembolso - startRefundFlow() 21

Consultar pagamento - askPaymentStatus() 23

Leitura de códigos - showScannerCodeReader() 26

Localização do dispositivo - askLocation() 28

Funçoẽs extras 30

Tela de loading 30

Fechar Mini App 30

**Entendendo o Emulador** **30**

**Comportamento em caso de erro** **32**

**Ícone do Mini App** **32**

**Diretrizes de integração** **32**

Pode 32

Não pode 32

Recomendado 33

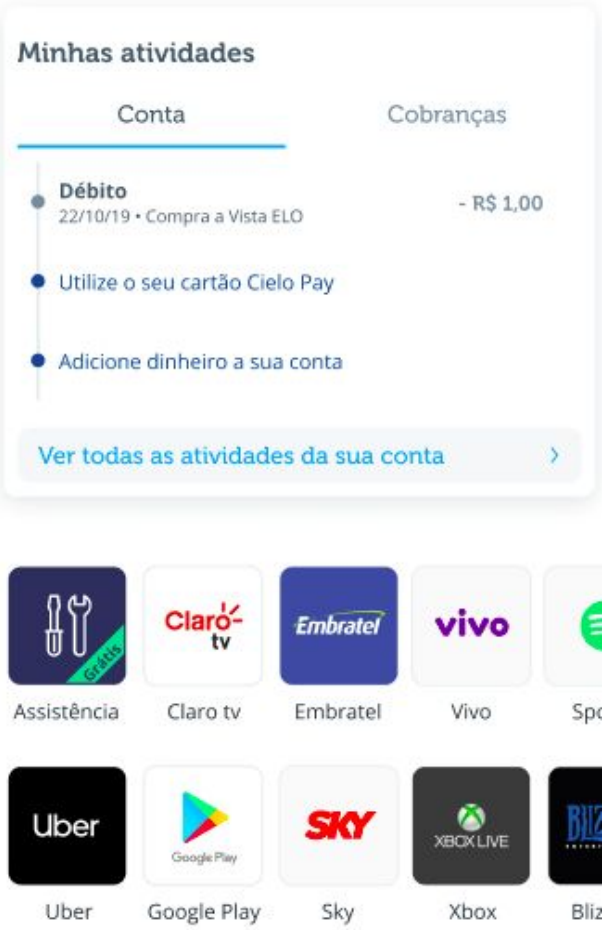
**Como publicar um Mini App** **33**

# 1. O que é o Mini App

Os Mini Apps são soluções disponibilizadas aos usuários do Cielo Pay que, através de parceiros, oferecem diferentes tipos de serviços. Essas soluções são aplicações web capazes de utilizar funcionalidades de integração com o aplicativo, como o fluxo de pagamentos do Cielo Pay.

# 2. Especificações

O Cielo Pay contém uma área onde é exibida uma lista de Mini Apps de parceiros. Cada Mini App será renderizado no Cielo Pay por meio de uma web view e poderá estar integrado a outras funcionalidades através de um pacote JavaScript (JS), que será o responsável pela comunicação entre Cielo Pay e Mini App.




*Listagem de Mini Apps no Cielo Pay*

### Integração com a Web View

- Renderização do Mini App através da web view.

### Integração com Módulo JS

- Identificação de qual sistema operacional o usuário está acessando.
- Identificação do usuário Cielo Pay logado.
- Pagamentos de produtos e/ou serviços via Cielo Pay.
- Reembolso de pagamentos.
- Consulta de pagamentos.
- Leitura de código de barras via câmera.
- Modal de loading, bloqueando interações na tela.
- Fechamento do Mini App.
- Solicitar localização do dispositivo.
- Scanner de códigos de pagamentos
- Upload de arquivos

## 3. Integração Mini App e Cielo Pay

Para estabelecer a comunicação entre o Mini App e o Cielo Pay a página do parceiro deverá se integrar ao **Gateway**, pacote JS disponibilizado pela Cielo. O Gateway contém ações e callbacks pré-estabelecidos de onde serão acionados os eventos, estando o aplicativo Cielo Pay preparado para interpretá-los. Ao clicar em um Mini App, o endereço disponibilizado pelo parceiro será renderizado na web view e o Gateway importado na página será inicializado.

## 4. Instalação

O gateway pode ser instalado via npm (gerenciador de pacotes), a partir da adição do registro:

```
http://ec2-54-233-106-212.sa-east-1.compute.amazonaws.com:4873
```

Será necessário efetuar o login:

```
npm login --registry http://ec2-54-233-106-212.sa-east-1.compute.amazonaws.com:4873
Username: miniapps
Password: C!elo_2020
Email: {emailParceiro}
```

❗ **Esta credencial é de propriedade da Cielo e não deve ser distribuída**

Uma segunda abordagem seria ter um arquivo `.npmrc` na raiz do projeto, com o conteúdo:

```
registry = http://ec2-54-233-106-212.sa-east-1.compute.amazonaws.com:4873
_auth = "bWluaWFwcHM6QyFlbG9fMjAyMA=="
email = {emailParceiro}
always-auth = true
```

Onde `_auth` seria o resultado do encode para base64 de `{user}:{password}`. Pode-se obter este encode a partir do comando:

```
echo -n 'miniapps:C!elo_2020' | openssl base64
```

Por fim, instalar o gateway

```
npm install gateway --registry
http://ec2-54-233-106-212.sa-east-1.compute.amazonaws.com:4873
```

A dependência deverá aparecer no package.json:

```
...
"dependencies": {
   "gateway": "^1.0.2"
 }
...
```

## 5. Utilização do Gateway

Após a instalação do gateway, pode-se realizar o seu import no projeto. No exemplo a seguir estamos importando apenas o método `init()`:

```
import { init } from 'gateway';
```

Para vários métodos do gateway é esperado um ou mais callbacks, ou seja, um método para ser chamado após concluir sua execução. Todos os métodos exportados pelo gateway podem ser conferidos na seção Funcionalidades.

Por exemplo, o primeiro método que deve ser chamado no gateway é o `init()` e com ele fornecido um callback:

```
import { init } from 'gateway';

function afterInitMiniApp(data){
  console.log("afterSetupMiniApp: ", data);
}

init(afterInitMiniApp);
```

💡 Você pode definir qualquer método como callback, basta passá-lo como parâmetro

💡 Os fluxos de integração com o Cielo Pay através do gateway podem ser encontrados em Fluxos de integração.

Quando o `init()` terminar de ser executado, será chamado `afterInitMiniApp()` fornecendo um parâmetro, o qual chamou-se de `data` neste exemplo. Para o método `init()` em especifico, `data` retornará o sistema operacional do usuário, entre outras informações:

```
{
  miniAppId: "12345"
  miniAppName: "Nome MiniApp"
  platform: "ANDROID"
  userName: "user@email.com"
  version: "1.4.35-xyz"
}
```

Inclusive `init()` irá lançar mensagens no console ao ser concluída:

```
Módulo instanciado, iniciando configuração...
Gateway Interceptor: Adicionando token de autorização.
Configuração efetuada com sucesso!
```

Cada método do gateway retornará um objeto condizente com seu contexto e estes estão descritos em Funcionalidades.

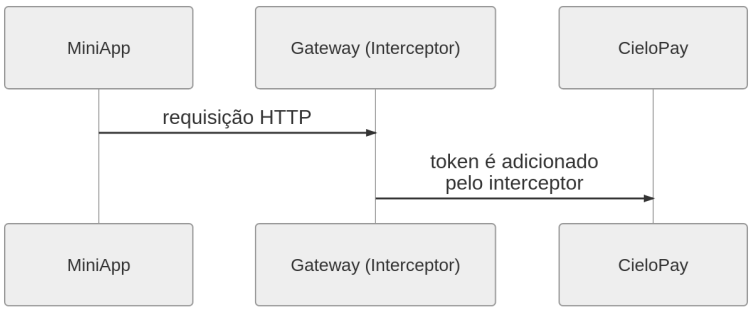
❗ Para cada página que precisar se comunicar com o Cielo Pay, será necessário importar os métodos e inicializar o gateway

# 6. Fluxos de integração

## Inicialização

Todo processo se inicia com a chamada de `init()`. É um processo importante, pois a partir dele o gateway irá fazer o setup para saber como se comunicar com o Cielo Pay, dentre outras funcionalidades.

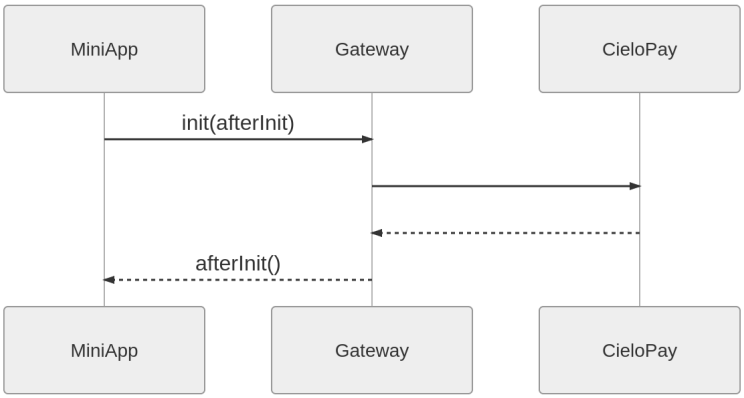
Durante processo de inicialização é feito o instanciamento do Interceptor, que é responsável por interceptar as requisições HTTP realizadas e adicionar um token de autorização do usuário logado.




*Interceptor Gateway*

Então pelas mensagens emitidas no console durante processo de inicialização, pode-se interpretar:

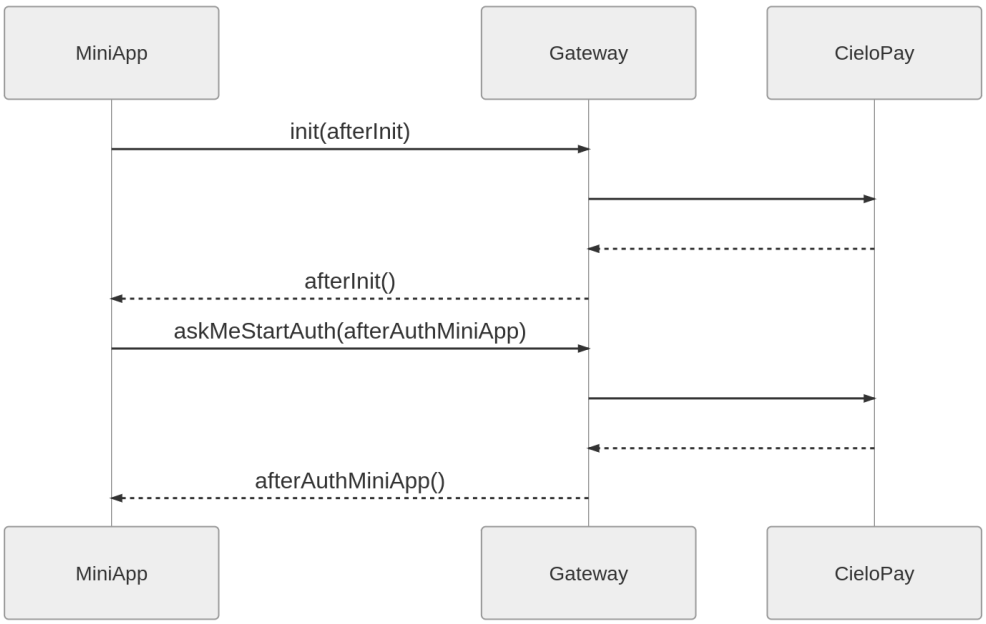
- Inicialização: `Módulo instanciado, iniciando configuração...`
- Interceptor: `Gateway Interceptor: Adicionando token de autorização.`
- Setup: `Configuração efetuada com sucesso!`




*Fluxo inicialização*

## Autenticação

Fluxo necessário para solicitar informações do usuário. Este fluxo será necessário durante o processo de solicitar status de pagamento, por exemplo.

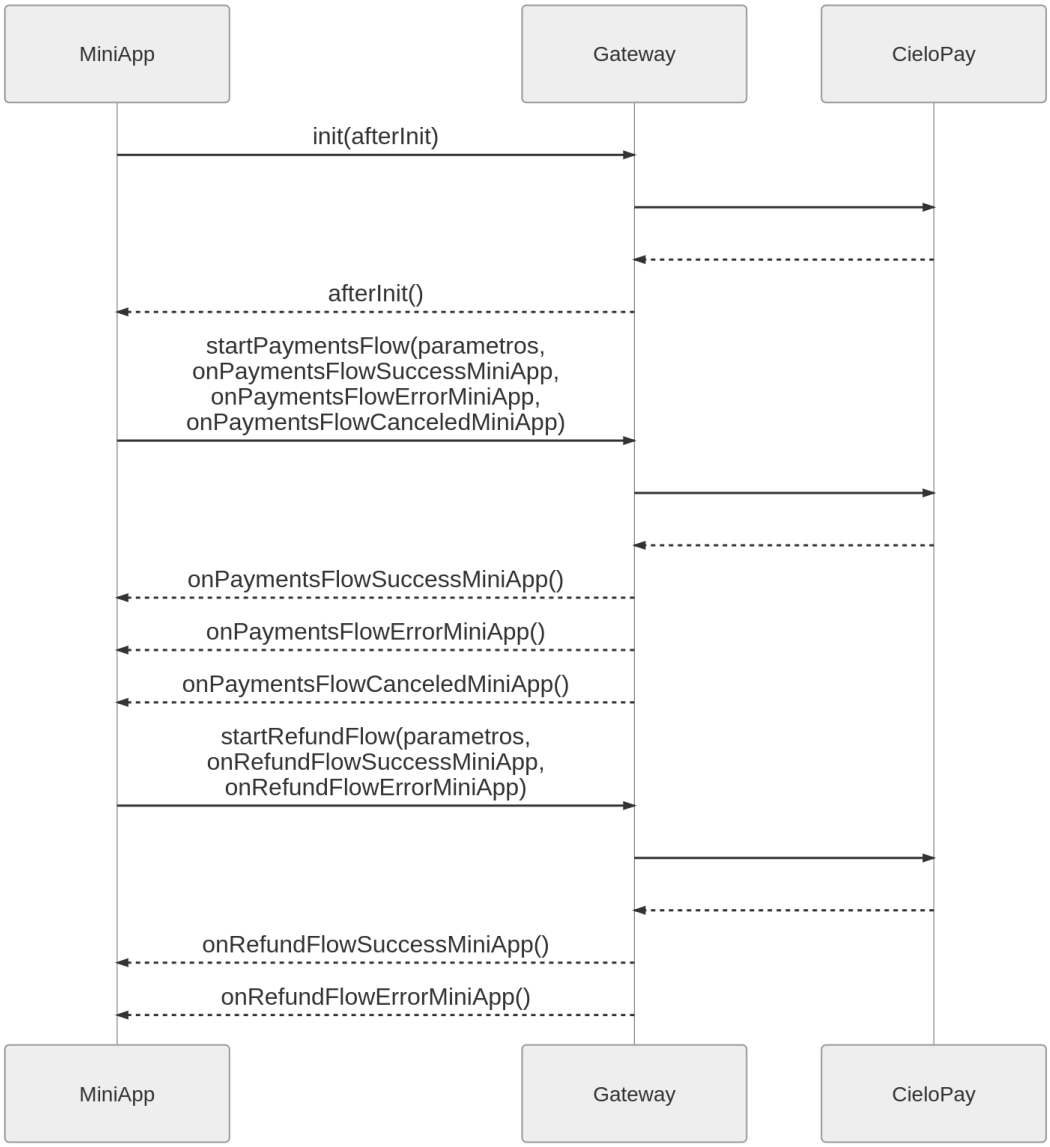



*Fluxo autenticação*

## Pagamento e Reembolso

No fluxo de pagamento serão passados os parâmetros do pagamento e mais 3 métodos de callback para os cenários de pagamento efetuado com sucesso, erro ou cancelado.

O reembolso similar ao pagamento necessita de 3 parâmetros: parâmetro para o reembolso, callback para sucesso e callback para erro.

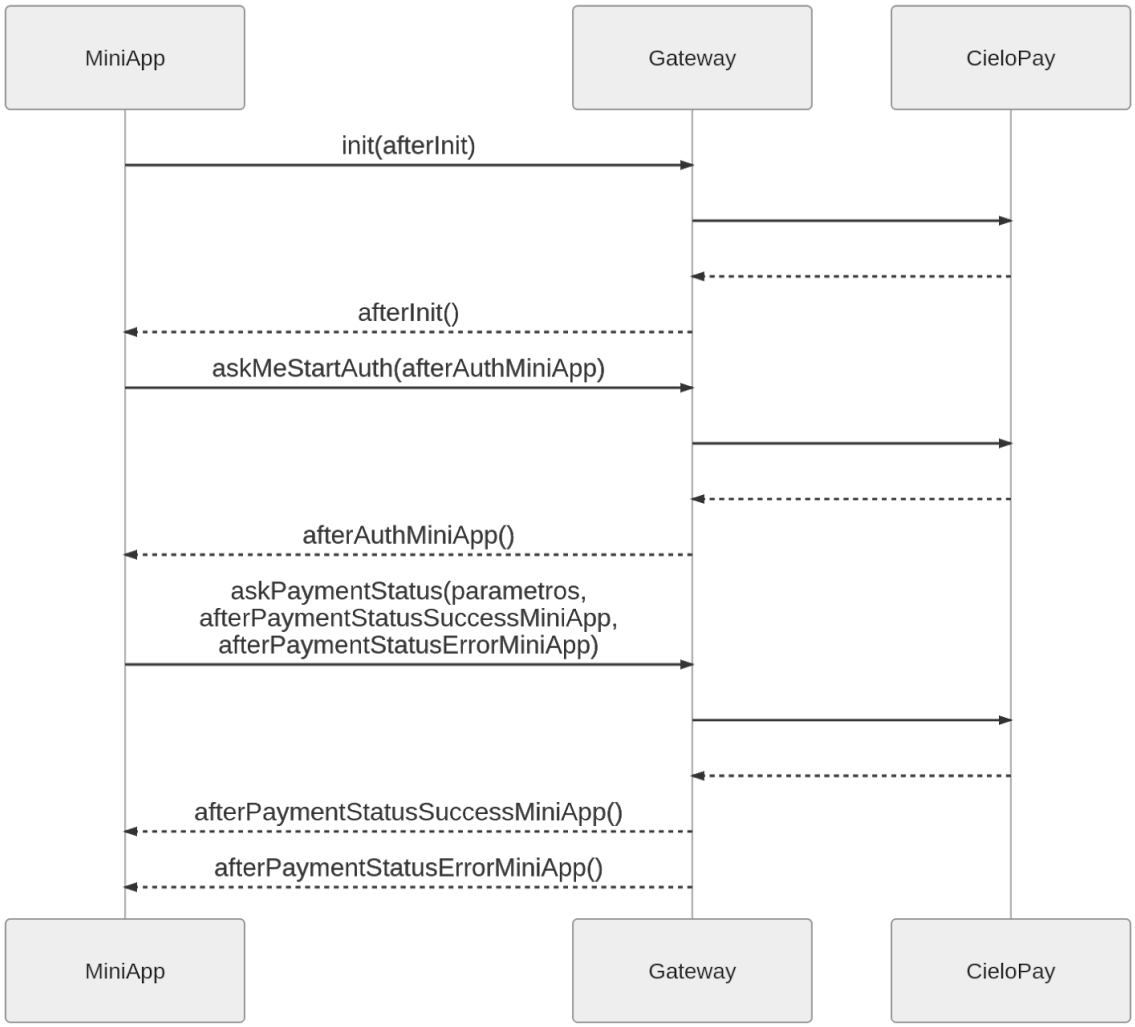



*Fluxo pagamento e reembolso*

### Status Pagamento

O status do pagamento pode ser requerido fornecendo os parâmetros do pagamento a ser consultado e mais 2 métodos para callback, contemplando cenários de sucesso e erro.

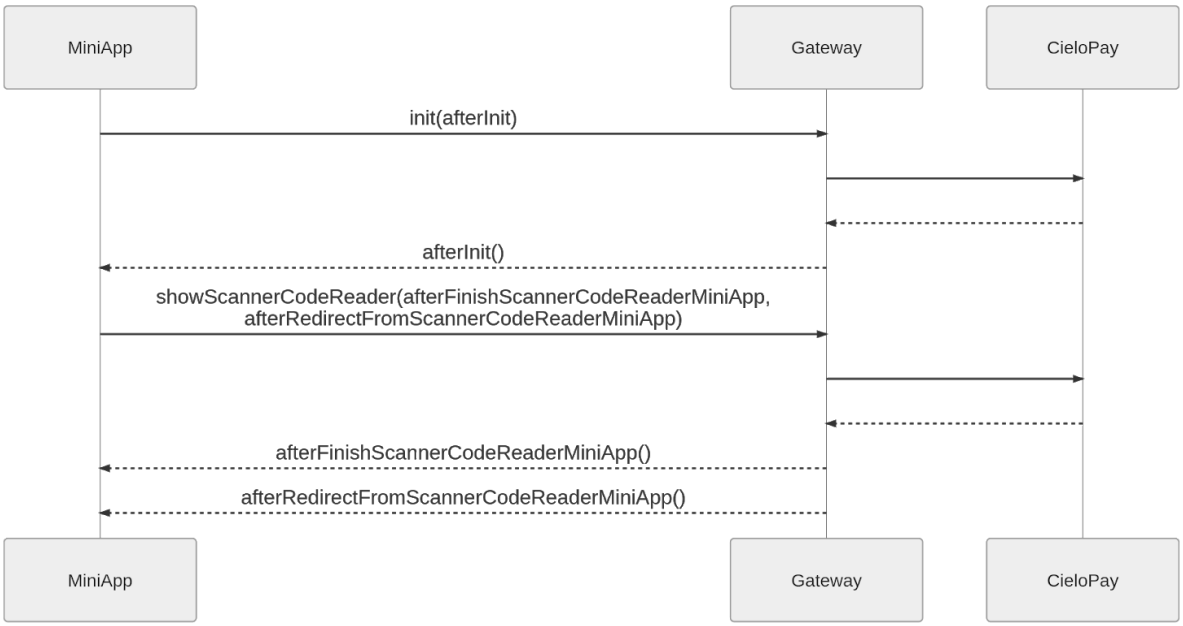
❗ Para este fluxo será necessário realizar a autenticação




*Fluxo status pagamento*

### Escanear código

Pode-se escanear um código de barras ou QR Code através do método `showScannerCodeReader()`.



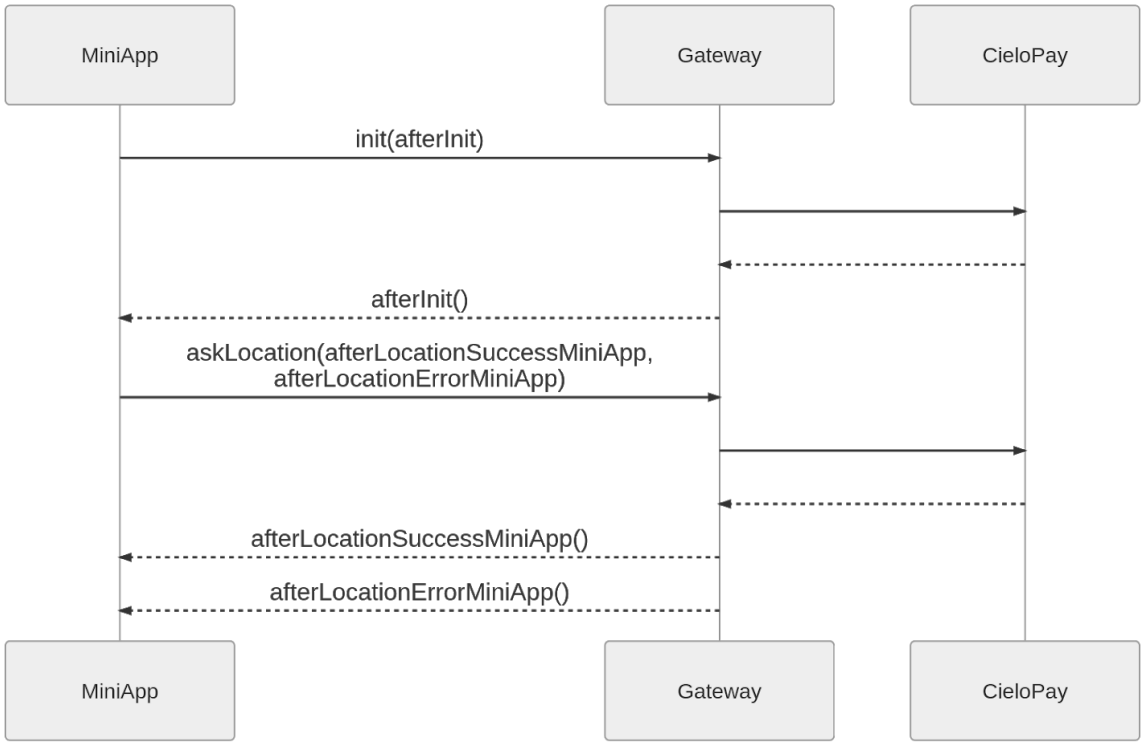

*Fluxo escanear código*

A seguinte tela será apresentada ao usuário quando utilizado este método:

*Tela para escaneamento de código*

## Localização

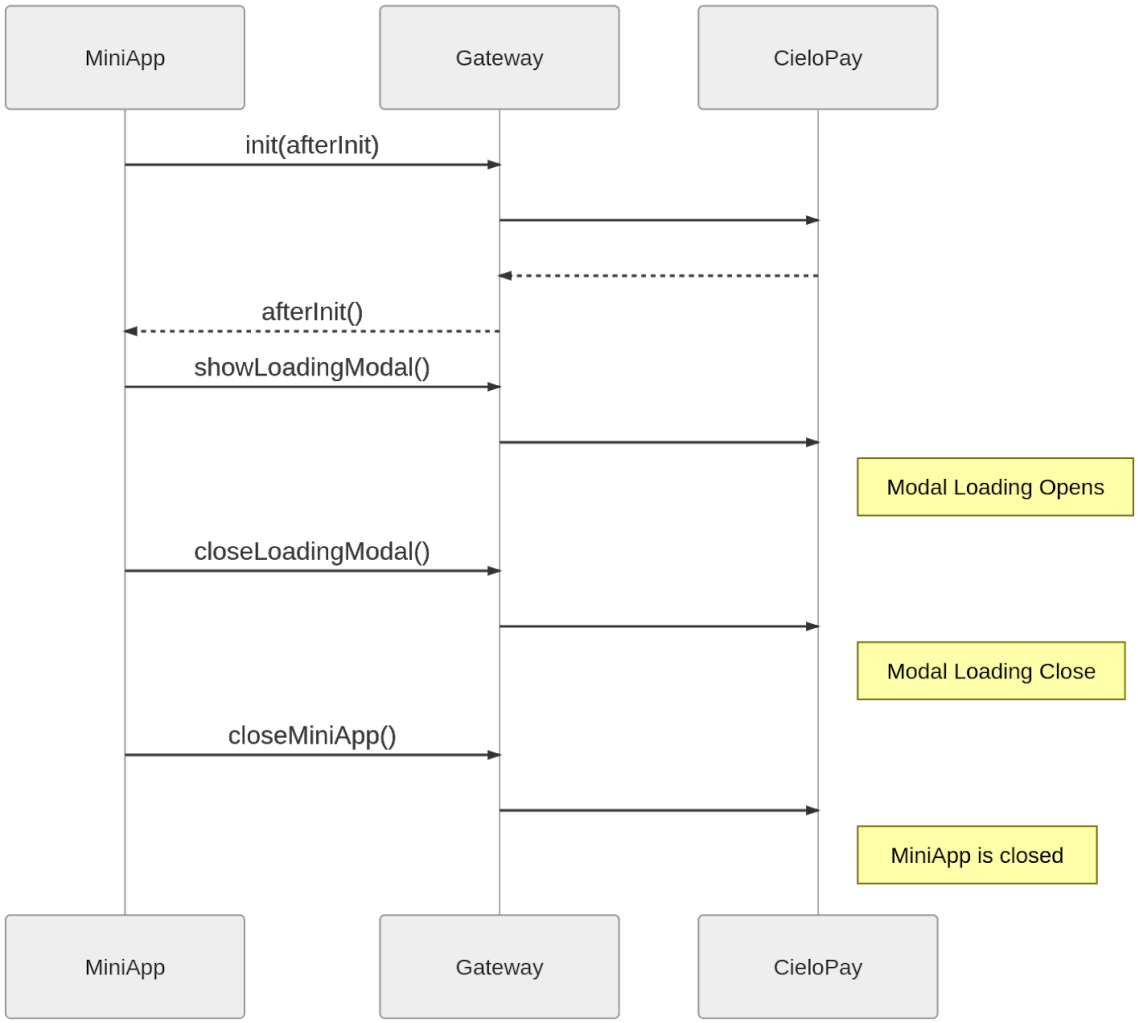
Fluxo para obter localização geográfica do usuário, será necessário fornecer 2 métodos para callback, cobrindo cenário de sucesso e erro.




*Fluxo localização*

### Modal Loading e Fechar Mini App

Fluxo para solicitação de abertura/fechamento de modal de loading na tela e a solicitação de fechar o Mini App:




*Fluxo abrir e fechar modal de loading e fechamento Mini App*

# 7. Funcionalidades

Com o gateway instalado, pode-se chamar as demais funcionalidades.

## Inicialização - init()

Método responsável por fazer a configuração do gateway, informando em seu retorno de qual sistema operacional o usuário está acessando o Mini App. Após essa configuração, o gateway irá saber se comunicar com o aplicativo Cielo Pay e poderá repassar os eventos acionados pela página.

**Método**: `init`

**Parâmetros**: `não aplicável`

**Callback**: `afterInitMiniApp`

| Callback | Descrição |
|---|---|
| *afterInitMiniApp* | Retorno após processo de Inicialização |

**Retorno**: Recebe no parâmetro `data`, pela callback `afterInitMiniApp`

```
{
  miniAppId: "12345"
  miniAppName: "Nome MiniApp"
  platform: "ANDROID"
  userName: "user@email.com"
  version: "1.4.35-xyz"
}
```

| Atributo | Descrição |
|---|---|
| *miniAppId* | MerchantId do Mini App |
| *miniAppName* | Nome do Mini App |
| *platform* | Sistema operacional do aplicativo que está acessando o Mini App, podendo assumir os valores “ANDROID” e “IOS” |
| *userName* | E-mail do usuário logado no Cielo Pay |
| *version* | Versão do Cielo Pay |

**Exemplo**:

```
function afterInitMiniApp(data){
  console.log("afterSetupMiniApp: ", data);
}

init(afterInitMiniApp);
```

Caso sucesso, console irá logar:

```
Módulo instanciado, iniciando configuração...
Gateway Interceptor: Adicionando token de autorização.
Configuração efetuada com sucesso!
```

Caso o callback não esteja implementado, console irá logar:

```
Callback setup não implementado.
```

❗ O campo version é um campo chave para o fluxo de pagamento. Se ele vier vazio deve-se exibir uma mensagem ao usuário para realizar a atualização do aplicativo.

### Autenticação - askMeStartAuth()

Solicita as informações do usuário logado no Cielo Pay.

**Método**: `askMeStartAuth`

**Parâmetros**: `não aplicável`

**Callback**: `afterAuthMiniApp`

| Callback | Descrição |
|---|---|
| *afterAuthMiniApp* | Retorno após processo de autorização |

**Retorno:** recebe no parâmetro `data`, pela callback `afterAuthMiniApp`

```
{
 "name": "user name",
 "document": "12345678901",
 "email": "user@user.com",
 "phone": "11988888888"
}
```

| Atributo | Descrição |
|---|---|
| *name* | Nome do usuário logado no Cielo Pay |
| *document* | Número do documento do usuário, podendo ser o valor do CPF ou CNPJ |
| *email* | E-mail do usuário cadastrado no Cielo Pay |
| *phone* | Número do telefone celular do usuário |

**Exemplo:**

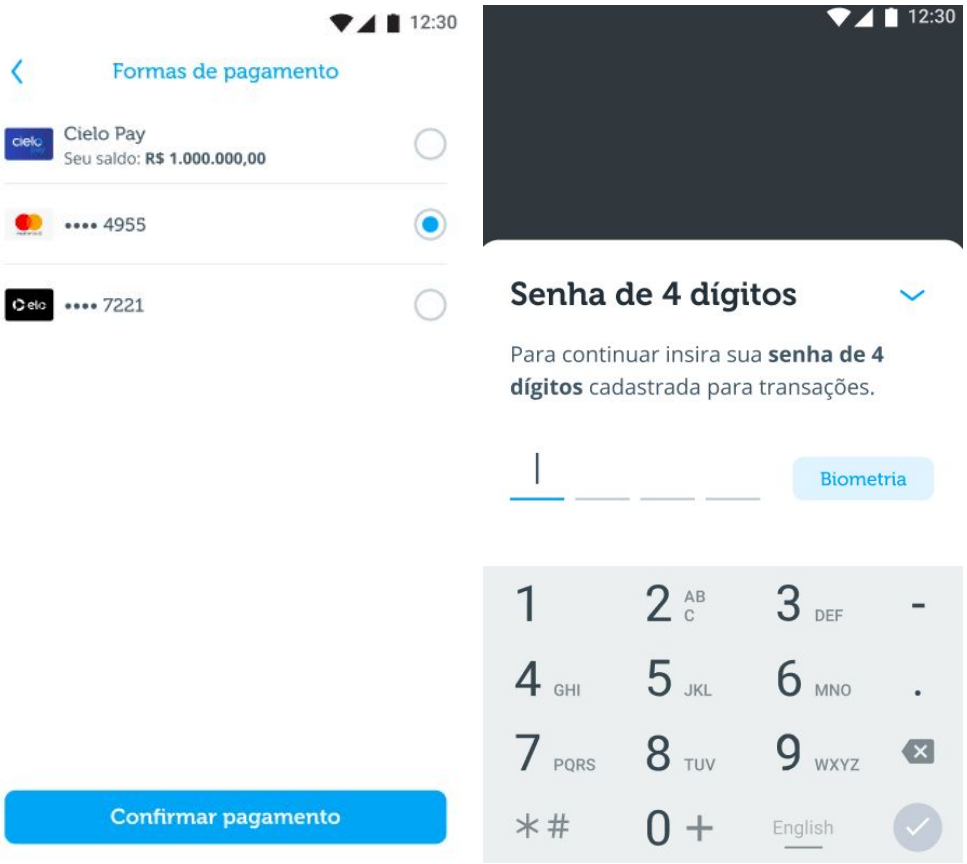
```
function afterAuthMiniApp(data){
  console.log("afterAuthMiniApp: ", data);
}
function afterInitMiniApp(data){
  console.log("afterSetupMiniApp: ", data);
  askMeStartAuth(afterAuthMiniApp);
}
init(afterInitMiniApp);
```

Caso callback não esteja implementado, console irá logar:

```
Callback autenticação não implementado.
```

### Pagamento - startPaymentsFlow()

No momento em que o Mini App chamar o `startPaymentsFlow()` o fluxo de pagamento será iniciado. O aplicativo do Cielo Pay responderá mostrando uma tela com a lista de cartões disponíveis na carteira do usuário e confirmação de pagamento, em seguida a de digitação da senha:




*Tela Formas de pagamento e Digitação de senha*

Após a escolha do cartão, confirmação do pagamento e leitura da senha, o pagamento será processado. A resposta do processamento do pagamento acionará os callbacks respectivos: `afterPaymentsSuccess()` em caso de sucesso e `afterPaymentsError()` em caso de erro. Se o usuário cancelar o pagamento em algum momento da transação, o callback `afterPaymentsCanceled()` será acionado.

❗ Fica a cargo do parceiro definir qual será o comportamento do Mini App de acordo com cada resposta. Exemplo: em um cenário de sucesso, a tela a ser apresentada é de responsabilidade do parceiro.

❗ No retorno de init verificar o atributo **version**. Se ele vier vazio deve-se exibir uma mensagem ao usuário para realizar a atualização do aplicativo e **não prosseguir** com o pagamento.

**Método**: `startPaymentsFlow`

**Parâmetro**: `data`

```
{
  "value": 10.00
}
```

Por padrão, o pagamento será realizado ao estabelecimento comercial do Mini App. Porém será possível realizar o pagamento diretamente a outros estabelecimentos comerciais. Para isso, além do valor a ser pago, deverá ser fornecido o merchantId do estabelecimento no parâmetro `data`. Exemplo:

```
{
  "value": 10.00 ,
  "merchantId": "12345"
}
```

| Atributo | Descrição |
|---|---|
| *value* | Valor a ser pago |
| *merchantId* | Identificador do estabelecimento comercial a quem deverá ser efetuado o pagamento |

**Callback**: `afterPaymentsSuccess`, `afterPaymentsError` e `afterPaymentsCanceled`

| Callback | Descrição |
|---|---|
| *afterPaymentsSuccess* | Retorno de sucesso da transação com as informações necessárias do sucesso |
| *afterPaymentsError* | Quando ocorre um erro durante a transação |
| *afterPaymentsCanceled* | Quando usuário interrompe a transação |

**Retorno:** recebe no objeto `data`, pela callback `afterPaymentsSuccess`

```
{
  "value": 10.00 ,
  "id": "6728jd82-28dj-3jud-ke8n-28dhe0kd7826"
}
```

| Atributo | Descrição |
|---|---|
| *value* | Valor pago |
| *id* | Identificador do pagamento que foi efetuado. Esse id é a chave utilizada na solicitação de reembolso. |

❗ O id recebido no sucesso de um pagamento é necessário para iniciar o fluxo de reembolso.

**Exemplo**:

```
function afterInitMiniApp(data){
  console.log("afterSetupMiniApp: ", data);
}

function afterPaymentsSuccess(data) {
   console.log("Payment success from webview");
}

function afterPaymentsError(data) {
   console.log("Payment error from webview");
}

function afterPaymentsCanceled(data) {
   console.log("Payment canceled from webview");
}

function onClickPayButton(amount) {
   startPaymentsFlow({ value: amount},
                     afterPaymentsSuccess,
                     afterPaymentsError,
                     afterPaymentsCanceled);
}
init(afterInitMiniApp);
```

Caso o callback de **sucesso** não esteja implementado, console irá logar:

```
Callback sucesso do pagamento não implementado.
```

Caso o callback de **erro** não esteja implementado, console irá logar:

```
Callback erro do pagamento não implementado.
```

Caso o callback de **cancelamento** não esteja implementado, console irá logar:

```
Callback cancelamento do pagamento não implementado.
```

### Reembolso - startRefundFlow()

O Mini App poderá utilizar o fluxo de reembolso e solicitar o estorno de um valor pago pelo Mini App através da função `startRefundFlow()`. O fluxo de reembolso só é possível com o id de um pagamento efetuado com sucesso.

**Método**: `startRefundFlow()`

**Parâmetros**: `data`

```
{
  "id": "6728jd82-28dj-3jud-ke8n-28dhe0kd7826"
}
```

| Atributo | Descrição |
|---|---|
| *id* | Identificador do pagamento que foi efetuado e que está sendo utilizado para a solicitação de reembolso. |

**Callback**: `afterRefundSuccess` e `afterRefundError`

| Callback | Descrição |
|---|---|
| *afterRefundSuccess* | Retorno de sucesso da tentativa de reembolso |
| *afterRefundError* | Quando ocorre um erro durante a transação |

**Retorno**: recebe no parâmetro `data`, pela callback `afterRefundError`

```
{
  "id": "6728jd82-28dj-3jud-ke8n-28dhe0kd7826"
}
```

| Atributo | Descrição |
|---|---|
| *id* | Identificador do pagamento que foi efetuado e que está sendo utilizado para a solicitação de reembolso. |

Na callback `afterRefundSuccess` não há dados em `data`

**Exemplo**:

```
/* Fluxo pagamento e reembolso */
function afterInitMiniApp(data){
  console.log("afterSetupMiniApp: ", data);
```

```
}

function afterPaymentsSuccess(data) {
   console.log("Payment success from webview");
   console.log("Código do pagamento:" + data.id);
}

function afterPaymentsError(data) {
   console.log("Payment error from webview");
}

function afterPaymentsCanceled(data) {
   console.log("Payment canceled from webview");
}

function afterRefundSuccess() {
   console.log("Refund success from webview");
}

function afterRefundError(data) {
   console.log("Refund error from webview");
}

function onClickPayButton(amount) {
   startPaymentsFlow({ value: amount},
                     afterPaymentsSuccess,
                     afterPaymentsError,
                     afterPaymentsCanceled);
}

function onClickRefundButton() {
   startRefundFlow({ id: data.id },
                   afterRefundSuccess,
                   afterRefundError);
}

init(afterInitMiniApp);
```

Caso o callback de **sucesso** não esteja implementado, console irá logar:

```
Callback sucesso do reembolso não implementado.
```

Caso o callback de **erro** não esteja implementado, console irá logar:

```
Callback erro do erro não implementado.
```

### Consultar pagamento - askPaymentStatus()

Pode-se consultar o status de um pagamento através de `askPaymentStatus()`.

❗ Este método requer autenticação

**Método**: `askPaymentStatus()`

**Parâmetros**: `data`

```
{
  "id": "6728jd82-28dj-3jud-ke8n-28dhe0kd7826"
}
```

| Atributo | Descrição |
|---|---|
| *id* | Identificador do pagamento que foi efetuado e deseja-se o status |

**Callback**: `afterPaymentStatusSuccess` e `afterPaymentStatusError`

| Callback | Descrição |
|---|---|
| *afterPaymentStatusSuccess* | Retorno de sucesso da requisição de status de pagamento |
| *afterPaymentStatusError* | Quando ocorre um erro durante a solicitação de status de pagamento |

**Retorno**: recebe no parâmetro `data`, pela callback `afterPaymentStatusSuccess`

```
{
  "paymentStatus": "Status"
}
```

Os status do pagamento podem ser:

| Status | Descrição |
|---|---|
| *NOT_FINISHED* | Não finalizado |
| *AUTHORIZED* | Autorizado |
| *PAYMENT_CONFIRMED* | Confirmado |
| *DENIED* | Negado |

| | |
|---|---|
| *VOIDED* | Cancelado |
| *REFUNDED* | Reembolsado |
| *PENDING* | Pendente |
| *ABORTED* | Abortado |
| *SCHEDULED* | Agendado |

**Exemplo**:

```
/* Pagamento e consulta status */
function afterInitMiniApp(data){
  console.log("afterSetupMiniApp: ", data);
}

function afterPaymentsSuccess(data) {
   console.log("Payment success from webview");
}

function afterPaymentsError(data) {
   console.log("Payment error from webview");
}

function afterPaymentsCanceled(data) {
   console.log("Payment canceled from webview");
}

function onClickPayButton(amount) {
   startPaymentsFlow({ value: amount},
                     afterPaymentsSuccess,
                     afterPaymentsError,
                     afterPaymentsCanceled);
}

function afterPaymentStatusSuccess(data) {
   console.log("Status Payment", data.paymentStatus);
}

function afterPaymentStatusError(data) {
   console.log("Status Payment error from webview");
}

function onClickGetStatusButton() {
   askPaymentStatus({ id: paymentCode },
                    afterPaymentStatusSuccess,
```

```
                    afterPaymentStatusError);
}
init(afterInitMiniApp);
```

Caso o callback de **sucesso** não esteja implementado, console irá logar:

```
Callback sucesso do pagamento não implementado.
```

Caso o callback de **erro** não esteja implementado, console irá logar:

```
Callback erro do pagamento não implementado.
```

### Leitura de códigos - showScannerCodeReader()

A partir do Mini App é possível abrir o módulo nativo de leitura de código de barras ou QRCode. O módulo deverá ser chamado através do método `showScannerCodeReader()`. Então o nativo irá utilizar a câmera do aparelho para realizar a leitura do código e após ler chamará o callback `afterFinishScannerCodeReader(data);`

**Metodo**: `showScannerCodeReader()`

**Parâmetro**: `não aplicável`

**Callback**: `afterFinishScannerCodeReader` e `afterRedirectFromScannerCodeReader`

| Callback | Descrição |
|---|---|
| *afterFinishScannerCodeReader* | Retorno de sucesso da leitura |
| *afterRedirectFromScannerCodeReader* | Quando usuário seleciona opção de digitar o código manualmente |

❗ A tela de leitura manual do código de barras, deve ser implementada no Mini App.

**Retorno**: recebe pelo parâmetro `data` na callback `afterFinishScannerCodeReader`

```
{
  "textCode": "Código de barras/QR Code gerado pelo nativo"
}
```

**Exemplo**:

```
function afterInitMiniApp(data){
  console.log("afterSetupMiniApp: ", data);
}

function afterFinishScannerCodeReaderMiniApp(data) {
   console.log("afterRedirectFromScannerCodeReaderMiniApp.");
}

function afterRedirectFromScannerCodeReaderMiniApp() {
   console.log("afterFinishScannerCodeReaderMiniApp: ", data);
}

function onClickGetStatusButton() {
   showScannerCodeReader(afterFinishScannerCodeReaderMiniApp,
                         afterRedirectFromScannerCodeReaderMiniApp);
```

```
}
init(afterInitMiniApp);
```

Caso o callback de **leitor** não esteja implementado, console irá logar:

```
Callback leitor de código de barras não implementado.
```

Caso o callback de **redirecionamento** não esteja implementado, console irá logar:

```
Callback redirecionamento do leitor de código de barras não implementado.
```

### Localização do dispositivo - askLocation()

O Mini App poderá solicitar os dados de localização do usuário, latitude e longitude, através da função `askLocation()`.

❗ Caso o usuário não tenha concedido previamente acesso à localização, o aplicativo solicitará permissão. Dependendo do acesso concedido, o Cielo Pay poderá retornar um callback de sucesso ou erro.

**Método**: `askLocation()`

**Parâmetros**: `não aplicável`

**Callbacks**: `afterLocationSuccess` e `afterLocationError`

| Callback | Descrição |
|---|---|
| *afterLocationSuccess* | Retorno de sucesso da solicitação de localização |
| *afterLocationError* | Retorno de erro da solicitação de localização |

**Retorno**: recebe pelo parâmetro `data` na callback `afterLocationSuccess`

```
{
   "lat": "38.8951",
   "long": "-77.0364"
}
```

| Atributo | Descrição |
|---|---|
| *lat* | Valor da latitude da localização do dispositivo |
| *long* | Valor da longitude da localização do dispositivo |

Recebe pelo parâmetro `data` na callback `afterLocationError`

```
{
  "message": "Não foi possível acessar a localização do usuário"
}
```

**Exemplo**:

```
function afterInit(data) {
   console.log("afterSetupMiniApp: ", data);
}

function afterLocationSuccess(data) {
   console.log("Latitude: " + data.lat);
   console.log("Longitude: " + data.long);
}

function afterLocationError(data) {
   console.log(data.message);
}
function onClickAskLocationButton() {
   askLocation(afterLocationSuccess, afterLocationError)
}

init(afterInitMiniApp);
```

Caso o callback de **sucesso** não esteja implementado, console irá logar:

```
Callback sucesso do pedido de localização não implementado.
```

Caso o callback de **erro** não esteja implementado, console irá logar:

```
Callback erro do pedido de localização não implementado.
```

### Funçoẽs extras

#### Tela de loading

O Mini App poderá usar a tela de loading nativa do Cielo Pay caso necessário. A tela bloqueia o uso do Mini App enquanto estiver aberta.
Para abrir o loading:

```
showLoadingModal();
```

Para fechar:

```
hideLoadingModal();
```

#### Fechar Mini App

O Mini App poderá ser fechado a qualquer momento, retornando para a tela inicial do Cielo Pay. Para isso basta chamar:

```
closeMiniApp();
```

#### Popups Javascript

É oferecido suporte para as funções Javascript alert, confirm e prompt.

#### Upload de Arquivos

É oferecido suporte a upload de arquivos.

## 8. Entendendo o Emulador

O emulador é um aplicativo desenvolvido para que seja possível simular a integração do Mini App do parceiro com o Cielo Pay. Nele é possível validar:

1. A renderização do Mini App do parceiro através da web view.
2. A integração com o módulo JS: Através do acesso aos dados do usuário, fluxo de pagamentos e outras funcionalidades.

No emulador é possível executar os seguintes cenários:

- Fluxo de pagamento: simulação de sucesso
- Fluxo de pagamento: simulação de erro
- Fluxo de pagamento: simulação de cancelamento

- Reembolso de pagamento: simulação de sucesso
- Reembolso de pagamento: simulação de erro
- Scanner de códigos: leitura de código através do uso da câmera do celular
- Scanner de códigos: redirecionamento para input manual do código
- Localização: simulação de sucesso com latitude e longitude fictícias
- Localização: simulação de erro
- Modal loading: abrir e fechar modal loading
- Fechamento da Mini App
- Abertura de popups javascript
- Upload de arquivos

## 7. Comportamento em caso de erro

O aplicativo do Cielo Pay notificará o usuário em caso de falta de comunicação com a internet durante toda a experiência dele dentro do Mini App.

## 8. Ícone do Mini App

A Cielo disponibiliza, junto com o módulo JS, o guia Cielo Pay - Manual de Identidade contendo quais são os padrões que o ícone do Mini App deve atender. Consulte esse guia para desenvolver o ícone que será submetido junto com o Mini App.

## 9. Diretrizes de integração

Segue alguns itens explicando o que pode e não pode ser feito em um Mini App:

### Pode

1. Manter a identidade visual de sua marca.
2. Não há limite no número de páginas para representar sua experiência.
3. Seu Mini App poderá ter a descrição de até 12 caracteres.
4. Caso necessite ofuscar seu código todas as chamadas a APIs e links terão que estar disponíveis.
5. Poderá adicionar ícone da sua marca no Mini App.
6. Seu Mini App poderá usar o fluxo de pagamento Cielo Pay.
7. Seu Mini App terá um espaço na lista do Cielo Pay.
8. Poderá usar o Emulador para se conectar com a Cielo Pay.

### Não pode

1. Não poderá apresentar fluxos de pagamento que não utilizem a Cielo como adquirente.
2. Não será permitido a divulgação visual ou textual de outra adquirente que não a Cielo.
3. Não poderá criar hiperlinks para páginas externas, sendo necessário hospedar todo código no ambiente Cielo.

4. Seu conteúdo não poderá depender do acelerômetro do equipamento, uma vez que a WebView será apresentada em modo retrato.

5. Não alterar a cor da barra superior do App Cielo Pay, que estará sempre presente.

**Recomendado**

1. Recomendamos que o seu Mini App não utilize mecanismo de loading, pois o aplicativo do Cielo Pay já possui um próprio.

## 10. Como publicar um Mini App

O fluxo de publicação de um Mini App no Cielo Pay conta com:

1. Uma fase de desenvolvimento e testes do parceiro. Para o teste, o parceiro poderá utilizar o emulador, validando a renderização do Mini App através da web view e integração com o Cielo Pay através do módulo.

2. Fase de integração. O parceiro fornecerá para a Cielo as informações necessárias para integração do Mini App no Cielo Pay:

    - URL da Web View.

    - Imagem do Ícone e cor de fundo do Mini App, de acordo com o guia de Manual de Identidade.

    - Informações de Estabelecimento Comercial EC, para o fluxo de pagamento.

3. Validação do Mini App do parceiro pela Cielo e publicação.