Manual de integração ELGINPAY

Versão 1.3

Sumário

Histórico de revisão

Data	Autor	Descrição	Versão
30-03-2020	Bruno Cruz	Criação do documento	1.0
01-04-2020	Bruno Cruz	Revisão processo de venda	1.1
16-09-2020	Bruno Cruz	Alteração do link do exemplo	1.2
03-02-2021	Bruno Cruz	Atualização da versão dos componentes descritos	1.3

A quem se destina

Este documento tem como objetivo orientar desenvolvedores no processo de integração com ElginPAY. Serão abordados os processos de venda, de cancelamento e operações administrativas utilizando linguagem de desenvolvimento java.
Para facilitar o entendimento, um exemplo será apresentado e descrito detalhadamente e seu código fonte estará disponível no link

https://bitbucket.org/teamshelgin/utility_elginpay/src/master/app/src/main/java/com/elgin/utilityelginpay/Controller/ElginPAY.java

Requisitos

Versão mínima do SDK Android: 19

Dependências para biuld.grandle

- implementation files('libs/InterfaceAutomacao-v2.0.0.6.aar')
  interface de comunicação com serviço de pagamento elgin pay
- *implementation group: 'org.apache.commons', name: 'commons-lang3', version: '3.4'*
  (Apache Commons Lang, um pacote de classes de utilitários Java para as classes que estão na hierarquia do java.lang ou são consideradas tão padrão que justificam a existência no java.lang)

Configuração do ambiente

Ambiente SMARTPOS

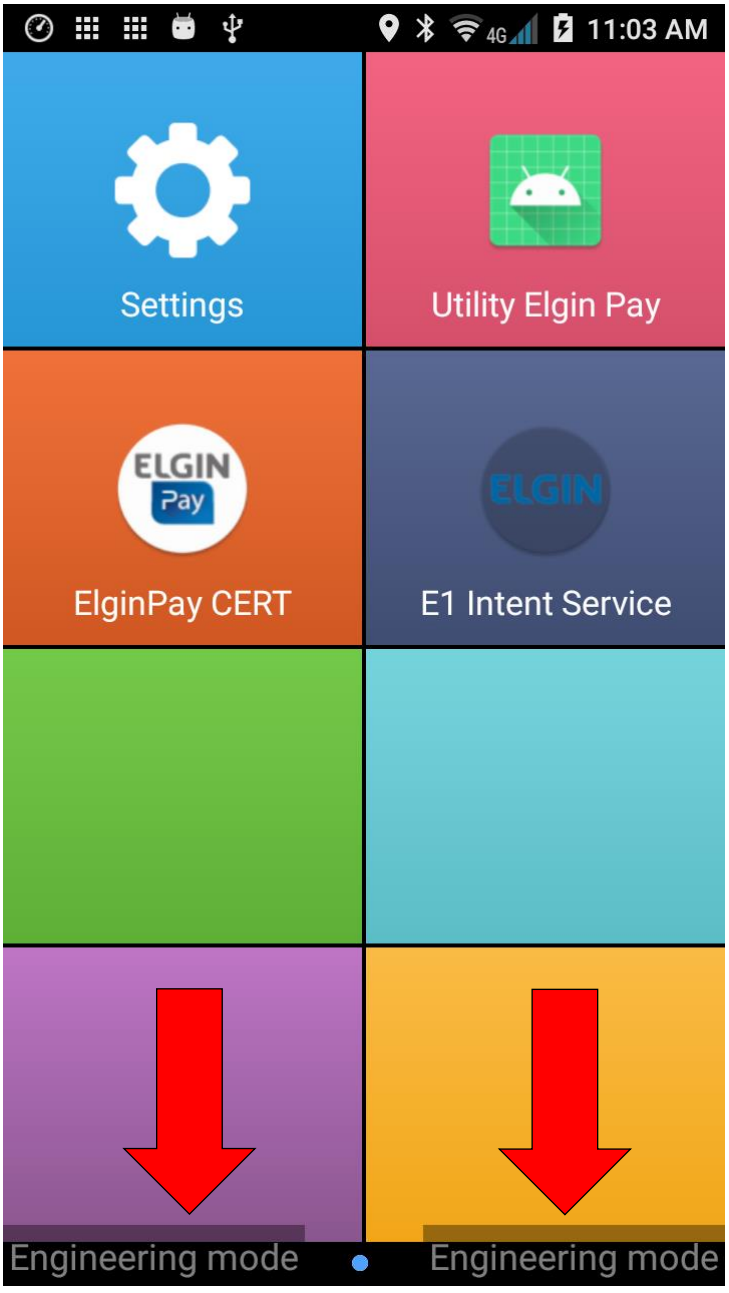
O SMARTPOS utilizado para homologação deve ter os seguintes pontos validados antes de iniciar o desenvolvimento:

Aplicações

- ElginPAY CERT
- Utility Elgin Pay – Aplicação de testes

Terminal

O terminal utilizado deve ser um terminal de desenvolvimento. Os terminais de desenvolvimento contêm uma marca d'água "**Engineering mode**" nos cantos inferiores, vide imagem abaixo:



Firmware

O Firmware do SMARTPOS necessário para o correto funcionamento dos testes precisa ser igual ou superior ao descrito abaixo:

Kernel version
3.10.49-wp1.0.0-3979-gd2b9417
Wed Apr 1 12:09:06 CST 2020
eng
boot= wp1.0.0-3979-gd2b9417 pcbb
oem=elgin-1.0.0-2996
splash=elgin

Essa informação pode ser consultada em **Settings -> AboutPOS**.

Caso seu terminal esteja incoerente com esses pontos passados, entre em contato com a ELGIN para maiores orientações.

Configuração do terminal

Para configuração do terminal serão necessários os seguintes dados

- Senha tecnica
- Id do ponto e captura (Fornecido pela ELGIN)
- CNPJ do cliente (Utilizar CNPJ usado no cadastro com a ELGIN)
- IP do servidor
- Porta do servidor

Os dados de senha, ip e porta do servidor serão fixos, para os demais, caso não tenha esses dados entre em contato com a ELGIN para obtê-los.

Inicie uma operação administrativa utilizando uma aplicação da Elgin. Esta pode ser o app Utility Elgin Pay

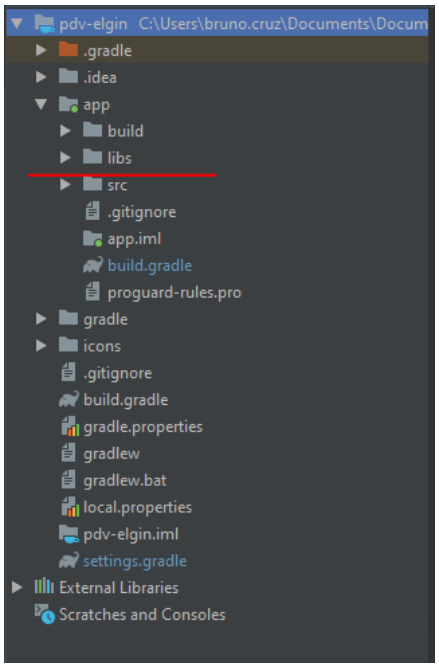
- Clique em CT07 – ADM – Instalação PDC.
- A partir desse ponto serão solicitados os dados conforme lista abaixo:
  - Senha: **314159**
  - Id do ponto e captura:
  - CNPJ do cliente:
  - IP do servidor: **200.219.246.107**
  - Porta do servidor: **7500**
- Após configurar as informações será impresso um comprovante de instalação confirmando a operação.

Ambiente de desenvolvimento

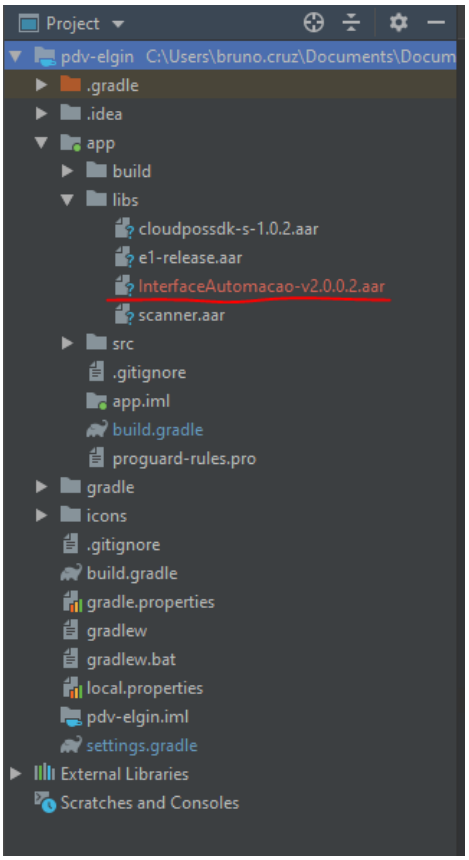
Neste processo o ambiente de desenvolvimento utilizado foi o ANDROID STUDIO versão 3.4.2 e linguagem de desenvolvimento JAVA.

Adicione as dependências conforme os requisitos no arquivo build.gradle da pasta app do projeto.

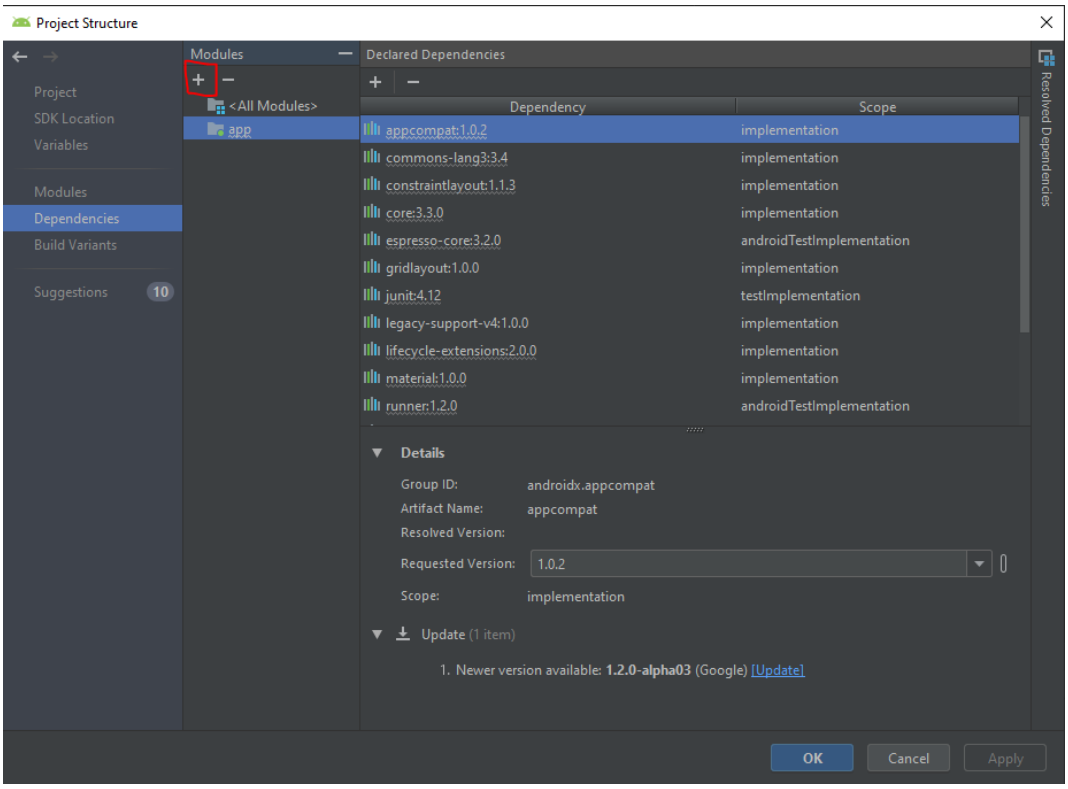
Após a configuração e sincronização do arquivo grandle, crie uma pasta chamada libs no diretório **app** do projeto, conforme imagem abaixo:



Em seguida mova para essa pasta libs o arquivo **InterfaceAutomacao-v2.0.0.6.aar.**

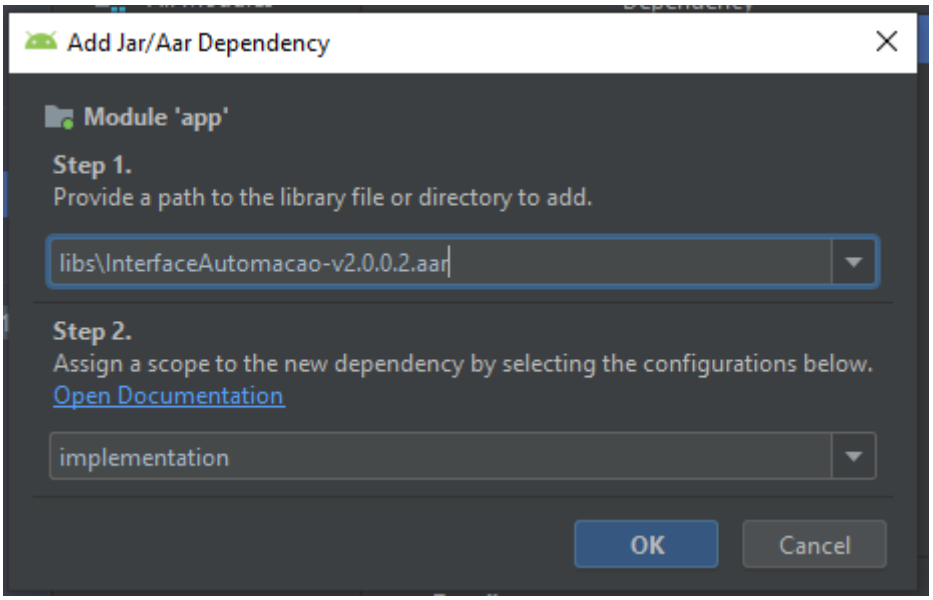


Clique com o botão no projeto e em seguida em Open module settings, uma janela irá se abrir com o título "**Project Structure**", clique em **dependencies** no menu lateral e em seguida **APP**



Clique no botão **+** conforme imagem anterior e em seguida em JAR Dependency.

Uma janela irá se abrir para seleção do arquivo ".aar" (neste caso a **InterfaceAutomacao-v2.0.0.6.aar**) conforme imagem abaixo:



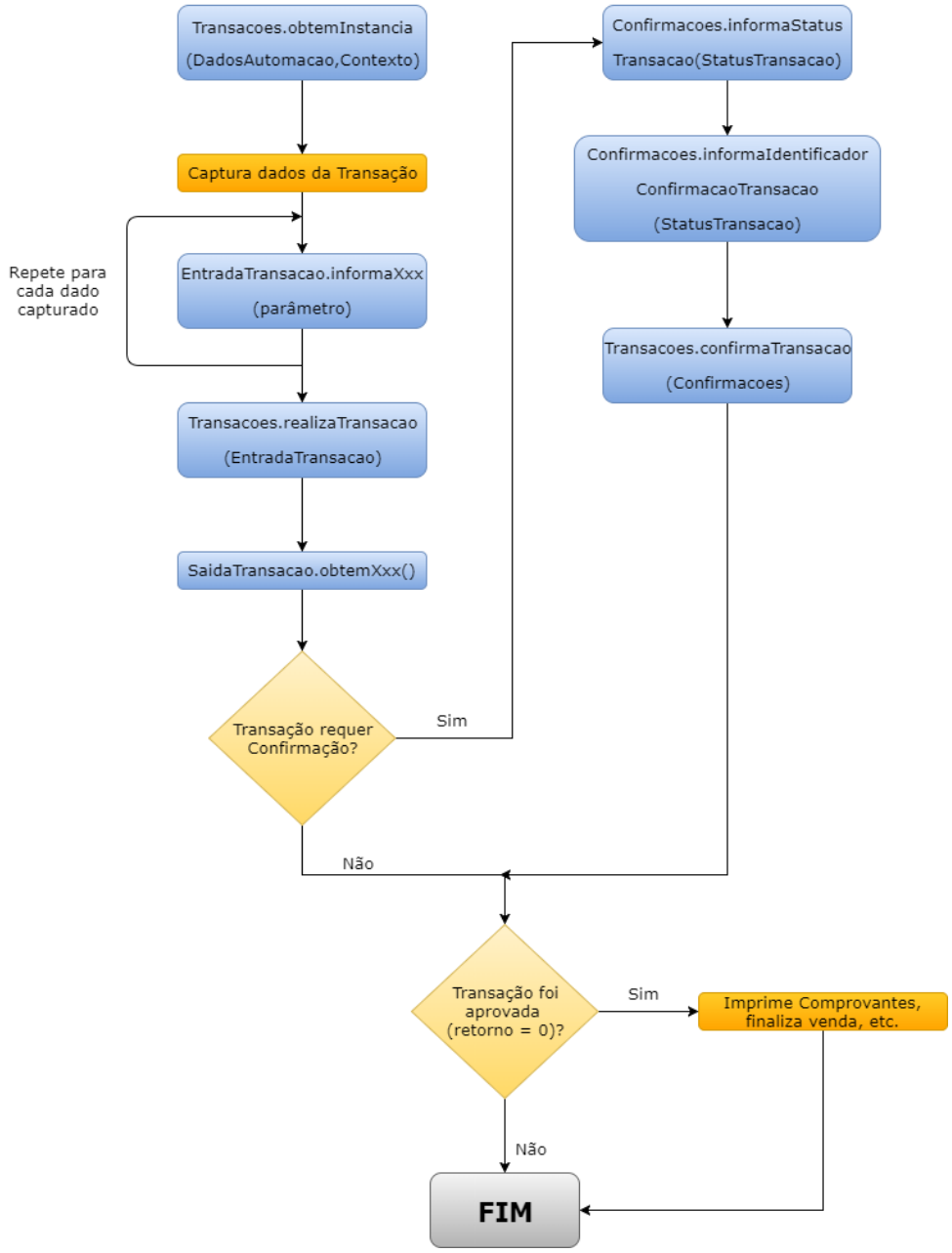
Clique em OK e a interface de comunicação com o ELGINPAY estará pronta para utilização.

Processo de venda

Em linhas gerais o processo de venda deve seguir os seguintes passos

- Instanciar um objeto **DadosAutomacao**, populando-o com seus dados;
- Obter uma instancia de **Transacoes** utilizando o objeto **DadosAutomação** criado anteriormente;
- Capturar os dados de transação, exemplo, valor, tipo de cartão, quantidade de parcelas. Esses dados são informados pelo objeto **EntradaTransacao** e seus respectivos métodos "informaXxx";
- A transação é efetuada através do método **Transacao.realizaTransacao(EntradaTransacao)**. Esse método é bloqueante, retornando apenas ao fim da operação. Sendo assim, é recomendado que o mesmo seja executado em uma Thread;
- Ao fim da transação será retornado um objeto **SaidaTransacao,** contendo todos os dados da operação realizada. A automação pode então obter todos os dados relevantes da operação realizada. O método **SaidaTransacao.obtemInformacaoConfirmacao()** deve ser chamado ao final de cada transação, independente do resultado, pois é ele que informa a obrigatoriedade de confirmação da transação;
- Caso a transação necessite de confirmação:
    - A automação deve instanciar um objeto do tipo **Confirmacoes** e informar o status final da transação (enumerado **StatusTransacao**), indicando se ela deve ser confirmada ou desfeita por algum motivo (método **Confirmacao.informaStatusTransacao(StatusTransacao)**.
    - Através do objeto **Confirmacoes** também deve ser informado qual o identificador da transação a ser confirmada/desfeita, utilizando o método **Confirmacoes.informaIdentificadorConfirmacaoTransacao(String).** Este identificador deve ser obtido através do método **SaidaTransacao.obtemIdentificadorConfirmacaoTransacao()**, ao final de cada operação realizada.
    - A confirmação é efetuada através do método **Transacao.confirmaTransacao(Confirmacao).**
- Caso a transação não necessite de confirmação, ou a confirmação foi efetuada com sucesso, a automação pode encerrar o seu fluxo transacional (impressão de comprovantes, armazenamento de dados, liberação de mercadoria, etc.).

Abaixo, fluxograma exemplificando o processo.



Abaixo será apresentado um exemplo de como esse processo pode ser realizado.

Essa classe apresentada pode processar qualquer operação, venda, cancelamento, operação administrativa bastando apenas ser informado um objeto entradaTransacao com a operação desejada e um handle inicializado.

Para começar crie uma classe que estenda Thread do pacote java.lang. Essa classe será utilizada para realizar o processamento das transações e as operações padrões.

Classe de exemplo:

```
30  public class ElginPAY extends Thread {
31
32      private static final int FIM_OPERACAO = 0;
33      private static final int FIM_IMPRESSAO = 1;
34      private static final int FIM_CONFIRMACAO = 2;
35      private static final int ERRO_IMPRESSAO = 3;
36      private static final int TRANSACAO_PENDENTE = 4;
37
38      private Handler handler, mHandler;
39      private EntradaTransacao entradaTransacao;
40      private SaidaTransacao saidaTransacao;
41      private Transacoes transacoes;
42      private Message message;
43      private Context context;
44      private Confirmacoes confirmacoes;
45
46      public ElginPAY(@NonNull EntradaTransacao e, @NonNull Handler h, @NonNull Context c){...}
86
87      @Override
88      public void run(){...}
109
110     private DadosAutomacao obtemDadosAutomacao(){...}
120
121     private Personalizacao obtemPersonalizacaoCliente(){...}
137
138     private void imprimeComprovante(){...}
227
228     private void finalizaVenda(){...}
239
240     private void resolveTransacaoPendente(){...}
269
270     private void apresentaMensagemPadrao(){...}
285
286     private void imprimeLista(List<String> a){...}
```

Linhas 32 – 36: Constantes utilizadas para indicar o status do processamento.

Linhas 38 – 44: Objetos Globais usados internamente pela classe.

Linha 46: Construtor da classe.
O construtor irá receber como parâmetro:

- Objeto de entrada para transação que defini a operação e as informações de transação
- Um objeto Handler, usado para enviar mensagens a Thread principal que fez a chamada a classe ElginPAY.
- Um objeto Context, usado para apresentação de mensagens ao usuário.

Linha 88: Reescrita do método run da classe Thread que será executado ao chamar Thread.start();

Adiante iremos detalhar cada método dessa classe

ElginPAY

```
46      public ElginPAY(@NonNull EntradaTransacao e, @NonNull Handler h, @NonNull Context c){
47          this.mHandler = h;
48          this.entradaTransacao = e;
49          this.context = c;
50          transacoes = Transacoes.obtemInstancia(obtemDadosAutomacao(), context);
51          confirmacoes = new Confirmacoes();
52
53          handler = new Handler(Looper.getMainLooper()) {
54              @Override
55              public void handleMessage(@NonNull Message msg) {
56                  switch (msg.what) {
57                      case ElginPAY.FIM_OPERACAO:
58                          imprimeComprovante();
59                          break;
60
61                      case ElginPAY.FIM_IMPRESSAO:
62                          finalizaVenda();
63                          break;
64
65                      case FIM_CONFIRMACAO:
66                          apresentaMensagemPadrao();
67                          break;
68
69                      case ERRO_IMPRESSAO:
70                          Toast.makeText(context,
71                                  text: "PROBLEMA NA IMPRESSÃO\n" +
72                                          "O comprovante poderá ser reimpresso pelo menu administrativo!",
73                                  Toast.LENGTH_LONG).show();
74                          apresentaMensagemPadrao();
75                          break;
76
77                      case TRANSACAO_PENDENTE:
78                          resolveTransacaoPendente();
79                          break;
80
81                      default:
82                          apresentaMensagemPadrao();
83                          break;
84                  }
85              }
86          };
87
88      }
```

O construtor irá inicializar os objetos utilizando os parâmetros passados.

Linha 50: Inicialização do objeto transações. Este irá receber um objeto DadosAutomacao retornado pela função obtemDadosAutomacao e o context passado no construtor.

Linha 53: Inicia um objeto handle que será usado internamente para processar a transação. Ao fim de cada operação o método handleMessage será invocado e irá definir qual operação deve ser realizada.

run

```
89
90      @Override
91      public void run(){
92          message = new Message();
93          try {
94
95              saidaTransacao = transacoes.realizaTransacao(entradaTransacao);
96
97              if(saidaTransacao.existeTransacaoPendente()){
98                  message.what = TRANSACAO_PENDENTE;
99              }else {
100                 message.what = FIM_OPERACAO;
101             }
102             //Envia mensagem para thread principal que fez a chamada
103             handler.sendMessage(message);
104
105         } catch (TerminalNaoConfiguradoExcecao | AplicacaoNaoInstaladaExcecao e) {
106             e.printStackTrace();
107             message.what = -1;
108             message.obj = e.toString();
109             handler.sendMessage(message);
110         }
111     }
112
```

O método run será executado ao chamar Thread.start().

Linha 92: Cria um objeto Message do pacote android.os.
Esse objeto será enviado a classe que fez a chamada no fim do processamento.

Linha 95: Realiza a chamada ao método realizaTransacao passando o objeto entradaTransacao com os dados da operação.
Esse é o ponto mais importante do desenvolvimento, pois é o método que irá chamar a aplicação ELGIN PAY.
Esse método é bloqueante e por esse motivo deve ser chamado de uma thread.
O retorno será um objeto SaidaTransacao com os dados do processamento.
Esse método pode lançar duas exceções, terminal não configurado e aplicação não instalada, por esse motivo deve ser criado em um try catch.

Linha 97-101: Validação do retorno da transação. Em caso de transação pendente o atributo what receberá o código especifico para resolver a transação em questão.

obtemDadosAutomacao

```
113 @   private DadosAutomacao obtemDadosAutomacao(){
114         return new DadosAutomacao( empresaAutomacao: "Elgin S/A",
115                 nomeAutomacao: "ELGIN S/A",
116                 BuildConfig.VERSION_NAME,
117                 suportaTroco: true,
118                 suportaDesconto: true,
119                 suportaViasDiferenciadas: true,
120                 suportaViaReduzida: false,
121                 obtemPersonalizacaoCliente());
122     }
```

O método **obtemDadosAutomacao** retorna um objeto **DadosAutomacao**. Nesse método serão definidos os atributos referentes a automação que está utilizando o ElginPAY.

Nesse ponto deve ser informado o nome da automação, versão app, empresa desenvolvedora da solução, suporte a troco, desconto, vias diferenciadas e via reduzida no construtor do objeto.
O PARAMETRO ***suportaViaReduzida*** DEVE SER DEFINIDO COMO FALSE.
Aqui também pode ser passado um objeto ***Personalizacao*** que será utilizado caso a automação deseje alterar a interface do ELGINPAY.

obtemPersonalizacaoCliente

```
124     private Personalizacao obtemPersonalizacaoCliente(){
125         Personalizacao.Builder pb = new Personalizacao.Builder();
126         pb.informaIconeToolbar(new File( pathname: "/sdcard/ic_launcher_round.png"));
127         pb.informaFonte(new File( pathname: "/system/fonts/Clockopia.ttf"));
128         pb.informaCorFonte("#FC9F00");
129         pb.informaCorFonteTeclado("#FC9F00");
130         pb.informaCorFundoToolbar("#FC9F00");
131         pb.informaCorFundoTela("#0C0807");
132         pb.informaCorTeclaLiberadaTeclado("#464B4E");
133         pb.informaCorFundoTeclado("#1B1A1C");
134         pb.informaCorTextoCaixaEdicao("#464B4E");
135         pb.informaCorSeparadorMenu("#FC9F00");
136
137         Personalizacao personalizacao = pb.build();
138         return personalizacao;
139     }
```

Esse método será usado para alterar a interface do ElginPAY.
Nesse exemplo foram alteradas as cores da fonte utilizada, toolbar, fundo da tela e fonte do teclado. Também foram configurados uma fonte diferente e um ícone.

imprimeComprovante

Esse método irá realizar a impressão dos comprovantes, podendo ser o comprovante da loja, do cliente, de cancelamento ou reimpressão.

```
private void imprimeComprovante(){
    message = new Message();
    ViasImpressao v = saidaTransacao.obtemViasImprimir();

    if(v == ViasImpressao.VIA_NENHUMA){
        message.what = FIM_IMPRESSAO;
        handler.sendMessage(message);
        return;
    }

    AlertDialog.Builder builder = new AlertDialog.Builder(context);
    AlertDialog a;

    Termica.setContext(context);
    Termica.AbreConexaoImpressora( tipo: 5,  modelo: "SMARTPOS",  conexao: "",  parametro: 0);

    int ret = Termica.StatusImpressora( param: 0);
    if (ret != 1) {
        //INDICANDO SEM PAPAL
        builder.setTitle("Erro na impressão");
        builder.setMessage("Impressora sem papel\nTroque a bonina");
        builder.setPositiveButton( text: "Reimprimir", (dialog, which) → {
                Termica.FechaConexaoImpressora();
                imprimeComprovante();
        });

        builder.setNegativeButton( text: "Sair", (dialog, which) → {
                Termica.FechaConexaoImpressora();
                message.what = ERRO_IMPRESSAO;
                handler.sendMessage(message);
        });

        a = builder.create();
        a.setCanceledOnTouchOutside(false);
        a.setCancelable(false);
        a.show();
    }else {

         if(v == ViasImpressao.VIA_CLIENTE_E_ESTABELECIMENTO){
             imprimeLista(saidaTransacao.obtemComprovanteDiferenciadoLoja());
             Termica.AvancaPapel( linhas: 4);
             Termica.FechaConexaoImpressora();

             builder.setTitle("Comprovante Cliente");
             builder.setMessage("Deseja imprimir via do Cliente?");
             builder.setPositiveButton( text: "Sim", (dialog, which) → {
                     saidaTransacao.informaViasImprimir(ViasImpressao.VIA_CLIENTE);
                     imprimeComprovante();
             });

             builder.setNegativeButton( text: "Não", (dialog, which) → {
                     message.what = FIM_IMPRESSAO;
                     handler.sendMessage(message);
             });

             a = builder.create();
             a.setCancelable(false);
             a.setCanceledOnTouchOutside(false);
             a.show();
         }else if(v == ViasImpressao.VIA_CLIENTE){
             imprimeLista(saidaTransacao.obtemComprovanteDiferenciadoPortador());
             Termica.AvancaPapel( linhas: 4);
             Termica.FechaConexaoImpressora();

             message.what = FIM_IMPRESSAO;
             handler.sendMessage(message);

         }else if(v == ViasImpressao.VIA_ESTABELECIMENTO){
             imprimeLista(saidaTransacao.obtemComprovanteDiferenciadoLoja());
             Termica.AvancaPapel( linhas: 4);
             Termica.FechaConexaoImpressora();

             message.what = FIM_IMPRESSAO;
             handler.sendMessage(message);
         }
     }
}
```

Linha 143: Obtém um objeto ViaImpressao para validação das vias que serão impressas;

Linha 145 - 149: Em algumas transações não existem vias a serem impressas, por esse motivo é feita uma validação e neste caso o método retorna sem realizar nenhuma operação;

Linha 154 - 155: Utilizando a classe Termica da biblioteca E1 de comunicação, abre a comunicação com a impressora do SmartPOS;

Linha 157 - 183: Verifica o status para papel da impressora. Caso o retorno seja diferente de 1 indica que a impressora está sem papel, sendo assim um alerta é usado para solicitar ao usuário que troque a bobina.
O usuário terá 2 opções, sendo sair para desconsiderar a impressão do comprovante e reimprimir para tentar novamente a impressão do comprovante. Para realizar a reimpressão será utilizado o conceito de recursividade, chamando novamente função;

Linha 185: Caso estejam disponíveis as vias do cliente e do estabelecimento, será feita a impressão da via do estabelecimento e na sequência o usuário é questionado se deseja fazer a impressão do cliente. Se a impressão da via do cliente for solicitada, é configurado para o objeto saídaTransação a via do cliente e o método de impressão é chamado novamente. Isso irá garantir que seja feita uma nova validação do status da impressora antes da impressão;

Linha 212 - 226: Realiza a impressão da via do cliente ou da via do estabelecimento;

Ao finalizar a impressão o status de fim de impressão é configurado e a porta de comunicação com a impressora é fechada

A impressão é feita através do método ImprimeLista utilizando o método ImpressaoTexto da classe Termica. O parâmetro utilizado é stilo = 1 para realizar a impressão usando mini-font.

```
289 @       private void imprimeLista(List<String> a){
290             for (int b = 0; b < a.size(); b++) {
291                 Termica.ImpressaoTexto(a.get(b), alinhamento: 0, stilo: 1, tamanho: 0);
292             }
293         }
294     }
```

finalizaVenda

```
231     private void finalizaVenda(){
232         if(saidaTransacao.obtemInformacaoConfirmacao()
233                 && saidaTransacao.obtemResultadoTransacao() == 0) {
234             confirmacoes.informaStatusTransacao(StatusTransacao.CONFIRMADO_AUTOMATICO);
235             transacoes.confirmaTransacao(confirmacoes);
236         }
237
238         message = new Message();
239         message.what = FIM_CONFIRMACAO;
240         handler.sendMessage(message);
241     }
242
```

Esse método é responsável por realizar a confirmação da venda. Como nem todas operações necessitam de confirmação, é feita uma validação usando o método **obtemInformacaoConfirmacao.**

Para realizar a confirmação, deve-se criar um objeto ***Confirmacoes*** e chamar o método ***informaStatusTransacao*** passando o parâmetro de confirmação automática (**LINHA 234**). Em seguida o método ***confirmarTransacao*** do objeto ***Transacoes*** deve ser chamado passando como parâmetro o objeto **confirmacoes**.

resolveTransacaoPendente

```
243     private void resolveTransacaoPendente(){
244         message = new Message();
245         AlertDialog.Builder builder = new AlertDialog.Builder(context);
246         builder.setTitle("Erro na Venda!");
247         builder.setMessage(saidaTransacao.obtemMensagemResultado() + "\n\nDeseja confirmar a transação?");
248         builder.setPositiveButton( text: "SIM", new DialogInterface.OnClickListener() {
249             @Override
250             public void onClick(DialogInterface dialog, int which) {
251                 confirmacoes.informaStatusTransacao(StatusTransacao.CONFIRMADO_AUTOMATICO);
252                 transacoes.resolvePendencia(saidaTransacao.obtemDadosTransacaoPendente(), confirmacoes);
253                 message.what =-1;
254                 handler.sendMessage(message);
255             }
256         });
257         builder.setNegativeButton( text: "Não", new DialogInterface.OnClickListener() {
258             @Override
259             public void onClick(DialogInterface dialog, int which) {
260                 confirmacoes.informaStatusTransacao(StatusTransacao.DESFEITO_MANUAL);
261                 transacoes.resolvePendencia(saidaTransacao.obtemDadosTransacaoPendente(), confirmacoes);
262                 message.what =-1;
263                 handler.sendMessage(message);
264             }
265         });
266
267         AlertDialog alertDialog = builder.create();
268         alertDialog.setCanceledOnTouchOutside(false);
269         alertDialog.setCancelable(false);
270         alertDialog.show();
271     }
```

Esse método será utilizando quando existirem transações pendentes. O mesmo solicitar ao usuário a confirmação ou o desfazimento da transação.

Em casos de transação pendente o objeto ***SaidaTransacao*** retornara um ***true*** no método ***saidaTransacao.existeTransacaoPendente().*** Diante disso a transação pendente deve ser confirmada ou desfeita.

1. Confirmando transação.
   Para fazer a confirmação deve ser criado um objeto ***Confirmacoes*** e chamar o método ***informaStatusTransacao*** passando o parâmetro de confirmação automática **(Linha 251).** Em seguida chamar o método ***resolvePendencia*** do objeto ***Transacoes*** e passar como parâmetro o retorno da função ***saidaTransacao.obtemDadosTransacaoPendente()*** e o objeto ***confirmações (Linha 252).***

2. Desfazendo a transação.
   Para desfazer a transação deve ser criado um objeto ***Confirmacoes*** e chamar o método ***informaStatusTransacao*** passando o parâmetro de defeito manual **(Linha 260).** Em seguida chamar o método ***resolvePendencia*** do objeto ***Transacoes*** e passar como parâmetro o retorno da função ***saidaTransacao.obtemDadosTransacaoPendente()*** e o objeto ***confirmações (Linha 261)***.

apresentaMensagemPadrao

```
273     private void apresentaMensagemPadrao(){
274         AlertDialog.Builder builder = new AlertDialog.Builder(context);
275         AlertDialog a;
276         builder.setTitle("Retorno");
277         builder.setMessage(saidaTransacao.obtemMensagemResultado());
278         builder.setPositiveButton( text: "OK",  listener: null);
279         a = builder.create();
280         a.setCanceledOnTouchOutside(false);
281         a.setCancelable(false);
282         a.show();
283
284         message = new Message();
285         message.obj = saidaTransacao;
286         mHandler.sendMessage(message);
287     }
```

Esse método é responsável por fazer a apresentação da mensagem de retorno ao fim da operação e informar a classe que fez a chamada para a transação que o processamento foi finalizado utilizando o handle enviado no construtor da classe. Um objeto da classe SaidaTransacao será enviado pelo atributo obj do objeto Message.

handleMessage

Para iniciar devemos criar um objeto handler usado para receber mensagens da classe ElginPAY.

Handle será enviado para classe ELGINPAY e ficara responsável por processar o retorno. O retorno será processado no método handlemessage quando a classe de processamento invocar o metodo sendmessage do objeto handle enviado pelo construtor.

```
handler = new Handler(Looper.getMainLooper()){
    @Override
    public void handleMessage(@NonNull final Message msg) {
        Toast.makeText( context: PagamentoActivity.this,  text: "Fim da operação de venda", Toast.LENGTH_LONG).show();
    }
};
```

Ref: https://developer.android.com/training/multiple-threads/communicate-ui

O último passo é iniciar a operação de venda. A tela abaixo é um exemplo para a seleção da forma de pagamento, nesse exemplo iremos considerar apenas o pagamento por cartão de crédito ou débito.

Usaremos o cartão de débito para exemplificar.
Ao clicar no botão o método abaixo será invocado.

```
228     btnPagamentoDebito.setOnClickListener(new View.OnClickListener() {
229         @Override
230         public void onClick(View v) {
231
232             Random r  = new Random();
233
234             entradaTransacao = new EntradaTransacao(Operacoes.VENDA,
235                     String.valueOf(r.nextInt( bound: 100)));
236
237             entradaTransacao.informaValorTotal("1000");
238             entradaTransacao.informaTipoCartao(Cartoes.CARTAO_DEBITO);
239
240             pay = new ElginPAY(entradaTransacao, handler,  c: PagamentoActivity.this);
241             pay.start();
242
243         }
244     });
245
```

Linha 234: Inicialização do objeto entradaTransacao passando como parâmetro o tipo da operação. Os tipos disponíveis estão na classe Operacoes. Nesse exemplo vamos usar Operacoes.VENDA.

Linha 237: Definimos o valor de entrada para a transação sendo o valor a ser cobrado do cliente. O valor é informado em centavos, nesse caso usamos 1000(mil), assim sendo, estaremos cobrando R$10,00.

Linha 240: Inicialização do objeto ElginPAY, passando como parâmetro o objeto entradaTransacao, handler e o contexto da view para que mensagens possam ser apresentadas.

Linha 241: Invoca o método start que fará a execução do método run da classe ElginPAY.

Assim finalizamos um exemplo para integração com ElginPAY, desde o start da classe ElginPAY que irá chamar a aplicação de pagamento (ElginPAY CERT) até o processamento de cada estado até chegar a confirmação.

Essa classe apresentada pode processar qualquer operação, venda, cancelamento, operação administrativa bastando apenas ser informado um objeto entradaTransacao com a operação desejada e um handle inicializado.

Processo de cancelamento de venda

Para realizar o cancelamento de uma venda algumas informações são necessárias, são elas: Data e hora, código de autorização, NSU e valor da transação. Esses dados são obtidos no retorno da transação no objeto SaidaTransacao através dos métodos, **obtemDataHoraTransacao(), obtemCodigoAutorizacaoOriginal(), obtemNsuHostOriginal() e obtemValorTotal()** respectivamente.

Utilizando a Classe ElginPAY criada no tópico anterior, veremos como o cancelamento funciona.

handleMessage

Primeiro temos a implementação para o processamento da mensagem que será enviada ao finalizar a operação. Aqui será definido o que deve ser feito ao finalizar, por exemplo armazenar em banco de dados as informações de transação.

```
handler = new Handler(Looper.getMainLooper()){
    @Override
    public void handleMessage(@NonNull final Message msg) {
        Toast.makeText(getContext(), text: "Fim da operação", Toast.LENGTH_LONG).show();
    }
};
```

Para iniciar o processo de cancelamento vamos implementar um evento a um botão e em seguida capturar as informações necessárias para a operação.

```
210
211     btnCancelarVendas.setOnClickListener((v) → {
214
215
216         final View dialogView = View.inflate(getContext(), R.layout.cancelamento,  root: null);
217         final AlertDialog alertDialog = new AlertDialog.Builder(getContext()).create();
218
219
220         Button data_time_set = dialogView.findViewById(R.id.btnOK);
221         data_time_set.setOnClickListener((view) → {
224             Date date;
225             EditText txtData = dialogView.findViewById(R.id.txtData);
226             EditText txtAut = dialogView.findViewById(R.id.txtAut);
227             EditText txtNSU = dialogView.findViewById(R.id.txtNSU);
228             EditText txtValor = dialogView.findViewById(R.id.txtValor);
229
230             String data = txtData.getText().toString();
231             data = data.replace( target: "/",  replacement: "");
232             data = data.replace( target: ".",  replacement: "");
233             data = data.replace( target: ":",  replacement: "");
234             data = data.replace( target: "-",  replacement: "");
235             try {
236                 SimpleDateFormat sdf = new SimpleDateFormat( pattern: "ddMMyyyy hhmmss");
237                 date = sdf.parse(data);
238
239             }catch (ParseException e){
240                 Toast.makeText(getContext(), text: "Erro ao converter data", Toast.LENGTH_LONG).show();
241                 return;
242             }
243
244             Random r = new Random();
245
246             entradaTransacao = new EntradaTransacao(Operacoes.CANCELAMENTO, String.valueOf(r.nextInt( bounc
247             entradaTransacao.informaValorTotal(txtValor.getText().toString());
248             entradaTransacao.informaNsuTransacaoOriginal(txtNSU.getText().toString());
249             entradaTransacao.informaDataHoraTransacaoOriginal(date);
250             entradaTransacao.informaCodigoAutorizacaoOriginal(txtAut.getText().toString());
251
252             pay = new ElginPAY(entradaTransacao, handler, getContext());
253             pay.start();
254
255         });
256         alertDialog.setView(dialogView);
257         alertDialog.show();
258
259     });
```

A implementação para esse botão de cancelamento cria um AlertDialog que é preenchido com uma View especifica. Essa View possui quatro Edittext para o operador preencher com os dados de cancelamento e um botão Ok para iniciar o processo.

Linha 246: Inicialização do objeto entrada transação com parâmetro de Operacoes.CANCELAMENTO.

Linha 247: Seta informação de valor total **(saidaTransacao. obtemValorTotal())**

Linha 248: Seta NSU da transação **(saidaTransacao. obtemNsuHostOriginal())**

Linha 249: Seta data e hora da transação **(saidaTransacao.obtemDataHoraTransacao())**

Linha 250: Seta código de autorização **(saidaTransacao. obtemCodigoAutorizacaoOriginal())**

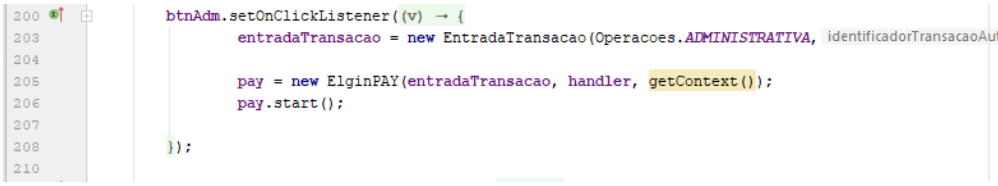
Linha 252-253: Objeto da classe ElginPAY é criado e inicializado com o objetos entradaTransacao, handler e getContext().
Após a criação, é dado start da operação.

No exemplo apresentado os dados são capturados em tela pelo operador utilizando o comprovante da transação, entretanto isso pode ser feito de forma simples se esses dados

fossem gravados em banco de dados, podendo dessa maneira fazer o cancelamento passando apenas um identificador único e fazendo a busca dos dados no banco.

Processo para operações administrativas

Como as demais operações, a operação administrativa é invocada utilizando um objeto entrada transação com o parâmetro de operações administrativa, vide exemplo abaixo:



```
200    btnAdm.setOnClickListener((v) → {
203            entradaTransacao = new EntradaTransacao(Operacoes.ADMINISTRATIVA, identificadorTransacaoAutomacao: "0");
204
205            pay = new ElginPAY(entradaTransacao, handler, getContext());
206            pay.start();
207
208    });
210
```

Processo de personalização do ELGINPAY

O App de pagamento ElginPAY é customizavel. A automação que faz integração com o app pode realizar a alteração de cores de fundo, teclado e toolbar, além de poder alterar as fontes e ícones utilizados pelo ElginPAY.

Por padrão o ElginPAY tem a identidade visual da Elgin, azul com branco, mas isso pode ser alterado de acordo com a necessidade da automação.

Abaixo um exemplo de alteração das características do ElginPAY

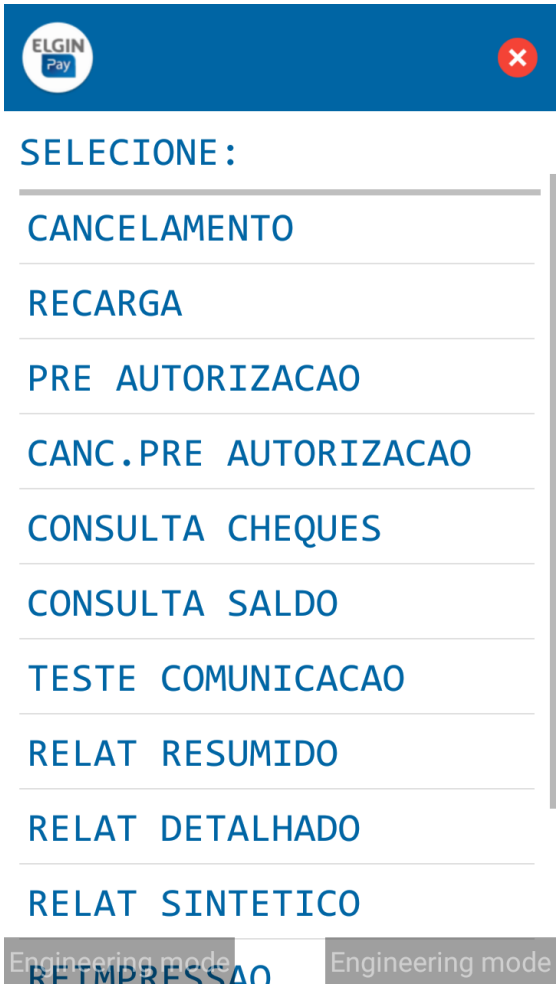


Figura 3 - Layout padrão ElginPAY

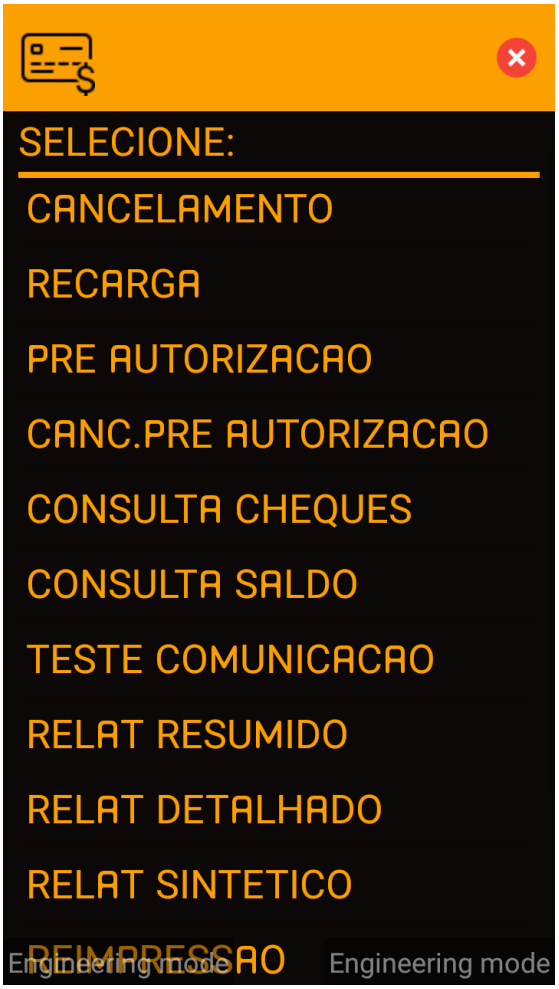


Figura 4 - Layout customizado ElginPAY

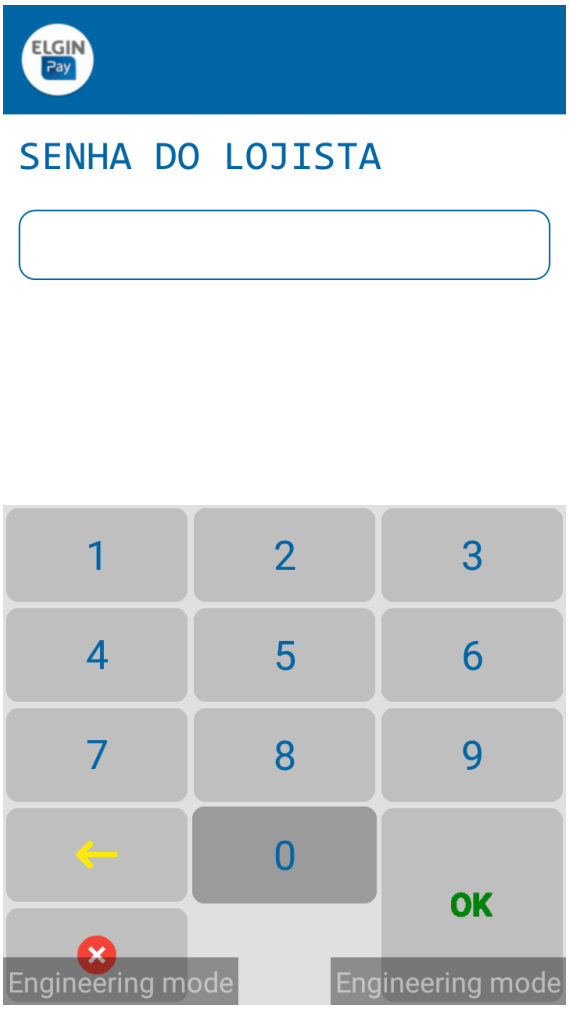


Figura 3 - Layout padrão com teclado

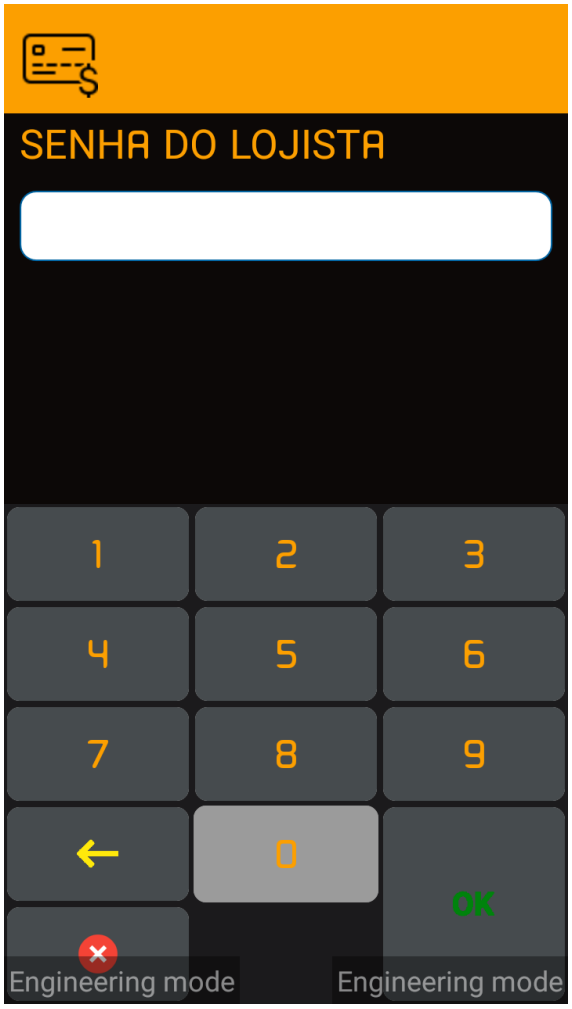


Figura 2 - Layout customizado com teclado

Para realizar essas alterações apresentadas os seguintes valores foram alterados

```
98      private Personalizacao obtemPersonalizacaoCliente(){
99          Personalizacao.Builder pb = new Personalizacao.Builder();
100         pb.informaIconeToolbar(new File( pathname: "/sdcard/ic_launcher_round.png"));
101         pb.informaFonte(new File( pathname: "/system/fonts/Clockopia.ttf"));
102         pb.informaCorFonte("#FC9F00");
103         pb.informaCorFonteTeclado("#FC9F00");
104         pb.informaCorFundoToolbar("#FC9F00");
105         pb.informaCorFundoTela("#0C0807");
106         pb.informaCorTeclaLiberadaTeclado("#464B4E");
107         pb.informaCorFundoTeclado("#1B1A1C");
108         pb.informaCorTextoCaixaEdicao("#464B4E");
109         pb.informaCorSeparadorMenu("#FC9F00");
110
111         Personalizacao personalizacao = pb.build();
112         return personalizacao;
113     }
```

Linhas 100 – 101: Alteram o ícone e a fonte utilizada. O ícone foi enviado para o diretório sdcard do dispositivo e a fonte utilizada é uma fonte do sistema android.

Linhas 103 – 109: Alteram as cores de fonte, fonte do teclado, fundo do toolbar (Barra superior), fundo da tela, cor da tecla liberada, fundo do teclado, texto da caixa de edição e cor dos separadores.
As cores são informadas em *#rgb.*