Relatório

acerca da integração odhenPOS-Getnet (Newland 910)
E guia para futuras integrações

Paulo Lopes do Nascimento

Sumário:

Definições Gerais

Objetivos técnicos e considerações

1-Getnet

1.1-Integração

1.2-Impressões

1.3-Considerações Finais

Definições Gerais :

Este relatório tem como objetivo documentar o processo de integração feito entre o odhenPOS e a máquina POS Newland 910 com integração Getnet.Em primeiro plano,falaremos sobre os objetivos e considerações que surgiram durante esse processo, e então entraremos em resultados e detalhamentos técnicos.

Para realizar quaisquer integrações no momento em que esse relatório está sendo feito,são necessários requisitos que giram em torno de utilizar da aplicação em JavaScript e de uma camada de Java/Kotlin para fazer as ações dentro do Android contido na máquina.

Existem algumas diferenças nas integrações que foram feitas antes da N910/Getnet, a principal é que o fluxo de código foi alterado.O que acontecia antes é que existia um arquivo de integração e um arquivo de impressão para cada integração que fora implementada, porém o que foi possível de perceber é que as funções se repetiam na maioria dos arquivos,exceto IntegrationSitef mas é um caso à parte,e dessa forma poderíamos abstrair vários arquivos em um só realizando validações para qual integração deveria ser chamada e afins.

No caso da integração,não foi criado nenhum arquivo e todas as funções da N910 rodam no arquivo IntegrationService.js, porém no caso da impressão foi necessária a criação de um arquivo PrinterGetnet.js assim como existia para as outras integrações, já que a forma como as impressões funcionam requer que exista uma instância com as especificidades de cada máquina.É uma possibilidade de no futuro abstrait também esses arquivos,mas não é o foco no momento.

Em termos gerais,podemos dizer que a conexão entre o JS e o Java/Kotlin ocorre por meio de algumas chamadas,sendo mais específico,fazemos a chamada ao plugin do cordova que permite que a aplicação então faça chamadas nativas no mobile,ou seja, ao Java/Kotlin para executarmos o restante dos processos.

Objetivos técnicos e considerações:

Como foi dito,nosso objetivo era o de concluir uma integração entre nossa aplicação e a máquina em questão,porém não pelos meios que já existiam para esse feito.Anteriormente,tínhamos vários arquivos de integração,cada qual correspondendo à uma integração diferente(Cielo,Rede,Cappta,Gertec,Getnet,etc),e dessa forma podíamos abstrair no JS do odhen usando um array com essas integrações (IntegrationService.js -> INTEGRATION_TYPE) e de acordo com o código proveniente da sessão atual (IDTPTEF) podíamos escolher a integração correta,e assim podíamos usar um IntegrationClass.function(), em que function() pode ser qualquer uma das funções principais e existentes em qualquer um desses arquivos de integração.A parte de impressão também possui um sistema como esse para escolher a integração que será usada,porém entraremos nela depois.

O novo modo de fazer seria executar uma abstração geral,de forma a não mais usarmos vários arquivos com suas chamadas de integração,e sim inserir essas funções que são gerais para todas as integrações em um arquivo só : IntegrationService.js.Para isso,teríamos que colocar todas as funções ou então usar try's e catch's para tentar executar sem validar um código ou algo do gênero.Para especificamente a máquina em questão nesse relatório,o código está compactado corretamente nesse novo modo de abstração.Para as outras integrações teríamos que realizar mudanças,mas que serão mais simples baseadas nas diretrizes deste relatório.

Agora,os objetivos específicos seriam o funcionamento das funções de : pagamento, estorno e impressão.Existem algumas considerações técnicas a se fazer em torno de como os arquivos conversam entre si e com o plugin,então entrarei com mais detalhes em seguida.

Existem algumas considerações que devem ser feitas,como por exemplo o fato de não ser possível realizar retornos DIRETOS no Java/Kotlin para o JS,mas é possível,e a forma que usaremos,retornar por meio de promises.

1-GetNet

1.1-Integração:

Começamos com o novo fluxo de execução das funções das integrações que acontece como se segue para o caso de transações:

PaymentController.js -> PaymentService.js ->assets/www/js/IntegrationService.js

Até aqui apenas o fluxo de chamadas dentro do odhen foi executado.

->assets/www/plugins/com.odhen.www/IntegrationService.js
(originalmente se encontra em
odhenPOS/platforms/android/platform_WWW/plugins/com.odhen.POS.www/IntegrationService.js)
->assets/www/cordova.js

Esse é o momento em que fazemos a chamada ao plugin IntegrationService.js que é a porta de entrada para o cordova,chamando a função de executar que está definida no cordova.js.

->odhenPOS/platforms/android/app/src/main/java/com/odhen/POS/IntegrationService.kt

O arquivo IntegrationService.KT,possui todas as funções propriamente ditas e realiza
as chamadas dentro do Java/Kotlin,de forma a organizar o processo.

Em termos específicos, além da alteração feita no fluxo do código, não houveram grandes mudanças no modus operandi de integrações.As chamadas de transações são feitas no arquivo IntegrationService.js e as de Impressão no arquivo PrinterGetnet.js.

A função que foi chamada no IntegrationService.js (payment,print,refund) é executada no Java/Kotlin,e no caso de pagamento e estorno,são chamadas a função chamaVenda e chamaEstorno, que estão na classe VendaController.kt, respectivamente, de forma a passar a execução adiante,e dentro dessas funções a transação é ou não concluída, e suas mensagens de erro já estão definidas na função sobrescrita onIntegrationResult(),(tanto os códigos de erro internos como a mensagem que aparece na tela via Toast já estão definidos nessa função).

1.2-Impressões

No caso das impressões,existem alguns pormenores, como por exemplo o fato de que a Getnet possui apenas alguns códigos de erro,e na documentação existe uma lista com os códigos e suas mensagens, de forma que implementamos todas.Na função de impressão no IntegrationService.kt verificamos o tipo de impressão (texto,qrCode,código de barras ou imagem) e a realizamos, logo depois fazendo um callback enviando o código de retorno e se ocorreu erro com um true ou se não com um false.O código então retorna para o arquivo PrinterGetnet.js que é responsável pelas chamadas de impressão para essa integração;esse callback então retorna para a função printResult() que se encontra no mesmo arquivo e que definitivamente resolve (literalmente) a chamada de impressão para a promise que foi feita no arquivo PrinterService.js, que então verifica se houve erro e toma as medidas necessárias de acordo com a situação: se ocorreu erro, cria uma notificação com o erro específico e se tudo correu corretamente, checa se há mais comandos para imprimir e os executa recursivamente.

Houve alguns obstáculos nas impressões, um deles foi que a impressão com cartões estava reiniciando as definições de fonte, então foi necessário realizar a chamada da função de inicialização dos parâmetros da impressora (initPrinterParams) dentro da função de imprimirTexto para manter a fonte e o tom de cinza corretos.

Além disso, houve um problema na obtenção do código de status da impressora, aparentemente o que acontece é que na primeira impressão que será feita a máquina demora um certo tempo para registrar o novo status,seja ele qual for, sucesso ou erro específico ,dessa forma foi necessário utilizar um Thread.sleep(1_000), que faz com que nossa aplicação “durma” por 1 segundo, que é tempo suficiente para a máquina terminar o registro para podermos receber o código correto.

1.3-Considerações finais:

*No caso do estorno,ao enviar as informações para a função chamaEstorno,o que acontece é que o cvNumber deve estar em string para funcionar corretamente,o resto está bem documentado na documentação da getnet.

*A função onIntegrationResult só é chamada após a última chamada da integração acontecer,logo a validação sobre se a integração anterior deu certo ou não acontece nas chamadas do JS,especificamente em integrationPaymentResult no IntegrationService.JS e que se encontra nos assets/www proveniente de odhenPOS dentro de platforms/android/app/.

*No caso do estorno,o processo começa no arquivo SaleCancelController.JS na função tefRefound()

*a função handleOpenRegister no RegisterController verifica se a integração é da SITEF

Se sim:carrega as tabelas SITEF e exporta os logs

Se não:apenas chama a função de login

Aqui é onde pode ocorrer o popup dizendo :"Não foi possível chamar integração.Sua instância não existe" devido a um IDTPTEF que cai na SITEF porém a transação não é SITEF.

*Foram utilizadas flags no arquivo IntegrationService.KT na função de impressão para facilitar o direcionamento do tipo de impressão que será feita.

*As recursividades de impressão só são possíveis por causa do callback realizado no IntegrationService.KT,o retorno dependerá dos argumentos passados na chamada da função em questão e.g :

```
window.cordova.plugins.IntegrationService.print("TEXTO",success,error)
```

Em que para ir para success dentro do KT :

integrationInstance.callbackContext.success("retorno")

e para o erro:

integrationInstance.callbackContext.error("retorno")

O retorno pode ser qualquer tipo de dado

*Nessa integração não foi necessário realizar delay entre as impressões, ao contrário do que poderia ser considerado devido à falta de função de corte,porém a função existe no arquivo PrinterGetnet.JS

*Existe possibilidade de editar o nodo dessa máquina no arquivo ImpressaoUtil.php que se encontra em odhenPOS/backend/vendor/odhen/api/src/Lib/ (talvez seja possível criar um registro com formatação melhor do que a que é entregue pela integração,porém aparentemente inclui arquivos de extensão .cs com nomes bastante específicos,é necessária uma busca maior já que DLL's são importadas e etc)