

Ferramenta de Manutenção de Terminais

Manuel Soares, F. Jorge F. Duarte

Instituto Superior de Engenharia do Porto
1161148@isep.ipp.pt; fjd@isep.ipp.pt

Resumo. Os terminais de multibanco são dispositivos que facilitam pagamentos de uma forma simples e segura. No entanto, são difíceis de aceder e manipular, dificultando a sua manutenção e o teste das suas funcionalidades. Este artigo aborda este problema, tendo como objetivo, fornecer uma aplicação com uma interface com o utilizador simples que comunica com terminais da Verifone através de comandos VIPA (instruções para o terminal executar uma funcionalidade), e que possibilite uma manutenção rápida e eficaz. Para tal, foi feita uma análise aos requisitos impostos pela empresa e desenhada uma arquitetura que responde aos mesmos, baseando-se em programação paralela. O desenho e a implementação da solução, são descritos neste documento, com o intuito de dar conhecimento do sistema.

Palavras-chave: Terminal, Manutenção, Comandos

1 Introdução

Os terminais multibanco têm um software específico e restrito que torna a interação com o dispositivo, pesada e complicada. Não é possível, por exemplo, criar ou ler ficheiros diretamente do terminal, sendo necessário, o uso de comandos para comunicar com o mesmo. Estes comandos, consistem em conjuntos de bytes enviados ao terminal que representam instruções a ser executadas pelo mesmo. Estes devem seguir um formato específico, contendo diversos parâmetros, tornado, assim, a interação com o terminal difícil.

A empresa pretendia uma aplicação que fosse capaz de comunicar com o terminal, de uma forma simples e eficiente e ajudasse a equipa a fazer manutenção dos terminais e a testar as suas funcionalidades. A aplicação deveria ser não bloqueável (*non blocking API*), ou seja, que fosse capaz de tratar de pedidos e respostas de forma assíncrona. Devia também ser capaz de controlar as respostas vindas do terminal, visto que o mesmo nem sempre as envia de forma linear, já que as pode remeter sem serem solicitadas.

Para tal, foi analisada a documentação fornecida pela empresa e foi feito um estudo sobre as tecnologias a serem utilizadas no desenvolvimento da aplicação, para corresponder a todos os requisitos impostos pela mesma. Foi desenhada uma arquitetura que permitisse baixo acoplamento e uma elevada eficiência, utilizando diversas *threads* [1] para dividir as diferentes tarefas da aplicação.

2 Tecnologias utilizadas

Nas seções seguintes são apresentadas as tecnologias que foram utilizadas. São apresentadas tanto as que dizem respeito à aplicação, como ao terminal.

2.1 VIPA Software

O VIPA Software [2] é um bundle que é instalado no terminal, que ajuda a gerir a configuração do mesmo, lidando com alguns dos seus ficheiros de configuração, e simplificando a interação com as suas funcionalidades. Após o VIPA estar instalado, é possível comunicar com o terminal através de comandos VIPA.

Estes comandos baseiam-se em comandos APDU (Application Protocol Data Unit), que são utilizados na comunicação com cartões inteligentes. Todo o seu conteúdo encontra-se no formato hexadecimal e possui um cabeçalho obrigatório composto por quatro bytes que representam quatro campos diferentes (CLA, INS, P1, P2). Estes informam, respetivamente, o tipo de comando, qual o comando e os seus parâmetros.

O VIPA disponibiliza um conjunto de comandos que permitem explorar diferentes funcionalidades do terminal, nomeadamente, o envio de um ficheiro, a leitura de um ficheiro, a verificação da quantidade de memória disponível, a apresentação de texto ou imagens no ecrã.

2.2 Spring boot e Spring MVC

O Spring boot [3] foi criado, com o intuito, de executar aplicações em Spring, de uma forma rápida e fácil, com o mínimo de configurações possível. É um módulo da framework Spring que auxilia o processo de build da aplicação, tratando de todas as configurações necessárias e permitindo ao programador, apenas se focar no conteúdo da aplicação. O Spring boot acelera a fase inicial de desenvolvimento em Spring, escondendo toda a complexidade desnecessária.

O Spring MVC [4] é um módulo de Spring, dedicado ao desenvolvimento de aplicações web, incluindo aplicações RESTful. Utiliza o protocolo HTTP, os endpoints são facilmente definidos através de uma anotação e, por omissão, utiliza como container o Apache Tomcat.

Ao associar-se o Spring Boot ao MVC, é extremamente fácil criar uma RESTful API, visto que o segundo nos fornece as ferramentas necessárias para o desenvolvimento web e o primeiro facilita o começo deste processo, pois abstrai todas as configurações de uma aplicação Spring, tornando mais rápido todo o processo de setup da aplicação. Para além disto, utiliza-se, ainda, o dependency-injection fornecido pela framework, reduzindo o acoplamento da API e facilitando todo o processo de testes.

2.3 Angular

O Angular [5] é uma framework criada pela Google para construir Single-Page Applications (SPA), utilizando HTML, CSS e, principalmente, JavaScript (ou TypeScript). A vantagem de desenvolver uma SPA, em oposição a páginas web clássicas, consiste na forma como esta processa pedidos e carrega os diferentes componentes que a compõem. Quando uma SPA é carregada pela primeira vez, esta é carregada totalmente e vai sendo atualizada conforme os pedidos do utilizador, isto é, é atualizada dinamicamente, em vez de estar constantemente a ser recarregada na sua totalidade. Assim, existe uma performance muito superior, pois a quantidade de tráfego e processamento são

reduzidos. Apenas os componentes com os quais o utilizador interage são atualizados, fazendo-o através de routing.

Contudo, uma das maiores vantagens do Angular, reside nas boas práticas inerentes à sua arquitetura. Este usufrui de padrões, como o Observer [6], para controlar diferentes funcionalidades da aplicação e interações entre os componentes, ou apenas para controlar pedidos ao back-end. Utiliza também, Dependency-Injection que facilita tanto o processo de testes, como o de implementação. Evita código duplicado e desnecessário, tornando-o mais simples e claro. Estes dois fatores foram cruciais na decisão de optar por Angular, visto que facilitam a implementação de alguma lógica, que é obrigatória estar presente no front-end (como, por exemplo, o Polling).

No caso da aplicação desenvolvida, era necessário um paradigma específico para interagir com o back-end que envolvesse Polling e que consistia no envio constante de pedidos ao back-end para verificar se existem novas respostas. Esta implementação, torna-se simples, usufruindo dos serviços e do padrão Observer disponibilizados pelo Angular, sendo, por isso, a melhor escolha.

3 Funcionalidades e outros requisitos

Inicialmente, foi feita uma análise aos requisitos impostos pela empresa, para ter uma noção de quais as funcionalidades a implementar e quais os limites do sistema a desenvolver.

Para os requisitos funcionais, foram definidos dezanove comandos que a equipa da 3C Techlab considerou necessários para fazer a manutenção dos terminais.

Para além dos requisitos funcionais, foram definidos alguns requisitos não funcionais, como a necessidade de construir uma non-blocking API (uma aplicação que permita utilizar a UI sem esta bloquear, através de pedidos assíncronos) e que as diferentes camadas da aplicação estivessem a correr em threads separadas.

A Tabela 1 apresenta o nome dos 19 comandos desenvolvidos e as respetivas funcionalidades. Estes foram escolhidos pela equipa da 3C Techlab, visto serem considerados os mais relevantes para a manutenção do terminal.

Tabela 1. Comandos para a manutenção dos terminais

Nome do comando	Funcionalidade
Display Text	Mostrar texto ou imagens no ecrã
Select File	Selecionar o ficheiro para executar outros comandos no mesmo.
Delete Binary	Apagar um ficheiro.
Read File	Ler um ficheiro.
Stream Upload	Enviar conteúdo para um ficheiro no terminal.
Reset Device	Fazer um reset ou reboot ao terminal.
Abort	Cancelar a operação em execução
Display HTML	Mostrar no ecrã o conteúdo de um ficheiro HTML.
Display QR Code	Mostrar no ecrã um QR Code.
Battery Status	Verificar o estado da bateria.
Free Space	Verificar a quantidade de espaço disponível.
Get Alphanumeric Data	Receber o conteúdo alfanumérico introduzido no terminal.
Get Numeric Data	Receber um número introduzido no terminal.
Get Printer Status	Verificar o estado da impressora.

Nome do comando	Funcionalidade
Request Choice	Receber a opção escolhida de uma lista e mostrar no ecrã do terminal.
Password Entry	Receber uma password introduzida no terminal.
Print Data	Imprimir o conteúdo pela impressora.
Keyboard Status	Receber informação das teclas pressionadas.
Get Binary Status	Receber informação de um ficheiro.

3.1 Arquitetura

Para responder ao problema, foi desenhada uma arquitetura que possibilitasse uma interação com o terminal, de forma eficiente e com um baixo acoplamento. Para tal, a aplicação foi dividida em três camadas principais: Controller, Communication e Model. A primeira tem a responsabilidade de gerir os pedidos vindos do exterior e as respostas vindas do terminal, bem como gerar os comandos pedidos pelo utilizador. A segunda trata de interagir com o terminal através do tipo de comunicação desejado.

Foi desenvolvida uma comunicação TCP, utilizando Socket, mas poderia também ser implementada uma comunicação através de porta serial. Finalmente, a terceira camada contém o domínio do problema, que neste caso é o conjunto de comandos desenvolvidos.

Para além disto, a aplicação segue uma arquitetura REST, para facilitar a sua interação com diferentes interfaces de utilizador. Assim, cada comando pode ser representado por um endpoint, recebendo os parâmetros no formato JSON, facilitando a utilização da aplicação num ambiente Web.

4 Implementação

Inicialmente, foi desenvolvida a camada Controller, que é executada numa única thread, para aumentar a eficiência da aplicação. Esta é responsável por guardar os pedidos recebidos numa queue e executar um de cada vez, esperando pela resposta vinda do terminal, antes de proceder ao próximo pedido. Para cada pedido, o Controller é responsável por gerar o comando e reportar quaisquer erros durante este processo.

De seguida, foi desenvolvida a camada da comunicação, que consiste na execução de duas threads distintas, uma para enviar conteúdo vindo do Controller para o terminal e outra para ler as respostas, que são guardadas numa queue presente no Controller. Esta camada é abstraída por uma interface que expõe os seus métodos à camada do Controller, de forma a possibilitar diferentes implementações da camada de comunicação. Isto é útil, pois no futuro podem ser desenvolvidas diferentes implementações para a comunicação com o terminal, como por exemplo, a comunicação por porta serial, sem interferir com o Controller.

Finalmente, foram desenvolvidos os comandos capazes de interagir com o terminal. Estes seguem um formato idêntico aos comandos APDU, com cabeçalhos semelhantes e conteúdo distinto uns dos outros e cada um, com um conjunto de parâmetros que oferecem controlo ao utilizador, para os poder manipular da forma desejada. Cada comando contém um conjunto de variáveis constantes que definem o seu cabeçalho e diferentes tags que servem para manipular o seu conteúdo. Cada tag representa um campo do comando que pode ser alterado pelo utilizador. A título de exemplo, no caso de um Display Text, existe a tag “DF8104” (em hexadecimal) que se refere ao texto a

ser apresentado no terminal. Ao gerar o comando, é utilizado o formato TLV (tag-length-value) e, a seguir ao cabeçalho, é colocada a tag referida, o tamanho do texto a ser apresentado e o texto em formato hexadecimal. Desta forma, o terminal interpreta que o conteúdo seguido da tag “DF8104” é o texto a ser apresentado no ecrã.

Foi também desenvolvido um mockup para a UI, com o intuito de demonstrar como seria a interação da aplicação com uma interface Web. Apenas algumas funcionalidades da aplicação se encontram representadas na mesma, sendo, por isso, necessário no futuro, o desenvolvimento de uma UI. No entanto, foi desenvolvido um protótipo que contém algumas das principais funcionalidades. O componente de visualização do comando Display Text, contém um simples formulário com todos os parâmetros necessários para o total funcionamento do comando, como é o caso do texto a ser apresentado, o caminho da imagem (caso seja desejada), a claridade do ecrã do terminal, os alinhamentos do texto e a ordem do texto e da imagem.

Esta UI comunica com o back-end através de pedidos HTTP e recebe as respostas através de Polling. Para tal, a aplicação disponibiliza um endpoint, que fornece acesso à fila de espera das respostas e, sempre que existir uma resposta, o front-end pode recebê-la e apresentá-la ao utilizador.

Assim, a aplicação é capaz de receber pedidos HTTP ou HTTPS, contendo os diferentes endpoints, definidos numa classe que serve de abstração para o resto da aplicação. Cada endpoint representa um comando, sendo que o utilizador pode enviar pedidos para qualquer um, com os parâmetros desejados, podendo, assim, testar diversas funcionalidades do terminal e fazer a manutenção do mesmo.

5 Conclusões

A aplicação desenvolvida traz diversos benefícios à equipa da 3C Techlab, visto ser uma ferramenta capaz de ajudar na manutenção de terminais utilizados pela mesma. Com a aplicação desenvolvida, atingiu-se os objetivos definidos pela empresa. Aplicação é capaz de enviar todos os comandos exigidos, de uma forma eficaz, controlando todas as respostas vindas do aparelho e gerindo eventuais erros.

Futuramente, seria interessante o desenvolvimento de uma UI mais elaborada que complementasse a aplicação, para dar uma melhor experiência ao utilizador.

Referências

1. Ronaldo. (2013). Java Threads: realizando processamentos paralelos, <https://www.devmedia.com.br/trabalhando-com-threads-em-java/28780>, último acesso 2020/10/02.
2. Vipa 6.x Interface Specification. A guide for system integrators (2016).
3. Afonso, A. O que é Spring Boot? <https://blog.algaworks.com/spring-boot/>, último acesso 2020/10/01.
4. Eduardo. Aplicação Web com Spring Boot e Spring MVC. <https://www.devmedia.com.br/desenvolvendo-uma-aplicacao-web-com-spring-boot-e-spring-mvc/34122>, último acesso 2020/09/30.
5. Afonso, A. O que é Angular? <https://blog.algaworks.com/o-que-e-angular/>, último acesso 2020/10/01.
6. Observer Design Pattern. https://sourcemaking.com/design_patterns/observer, último acesso 2020/10/01.