Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Departamento Acadêmico de Informática – DAINF
Engenharia de Computação
Oficina de Integração 3 (EEX23) – S71 – 2019.1

Relatório Técnico

BitBox - Uma Cripto-Vending Machine operada via celular

Gustavo Akira Gondo – gustavogondo@alunos.utfpr.edu.br
André Luiz Rodrigues dos Santos – andresantos@alunos.utfpr.edu.br
Giovanni Luiz Zanetti – giovanni-zanetti@hotmail.com
Guilherme Hideki Schneider – guilhermekaik@hotmail.com
Vitor de Oliveira Silva – vitor.de.o.silva@gmail.com

Junho de 2019

Resumo

Neste trabalho é apresentado o desenvolvimento de uma máquina de vendas operada via aplicativo de celular cujas transações são operadas com Criptomoedas. As tecnologias utilizadas são apresentadas individualmente e os resultados do desenvolvimento e dos testes são apresentados na conclusão.

1 Introdução

O objetivo desse projeto é o desenvolvimento de uma *Vending machine* operada via aplicativo de celular, cujas transações são processadas em criptomoedas. O projeto foi inspirado no vídeo *DIY Vending machine*[1] do *Youtube*, mas modificado para receber transações via um aplicativo *mobile* e transacionar o dinheiro via criptomoedas. Uma das maiores motivações do projeto foi o desacoplamento dos sistemas de pagamento físicos como cédulas, cartões, etc. de um sistema completamente automático, portanto não haveriam custos associados ao uso desses tipos de meios de pagamento terceirizados. Outra motivação central foi o desenvolvimento e a construção de um produto que pudesse integrar conhecimentos de interfaceamento de componentes mecânicos, eletrônica e desenvolvimento de *apps* e servidores. A *vending machine* consiste em algumas partes principais, especialmente o aplicativo móvel, o servidor e a *vending machine* propriamente dita, que serão discutidas mais a fundo posteriormente.

O sistema, apresentado na figura 1, funciona da seguinte maneira: o usuário instala o aplicativo no seu *smartphone*, obtém um endereço de cripto-moeda no aplicativo e entra na sua conta. Após isso, o usuário pode escanear o código

[1] https://www.youtube.com/watch?v=BHQBsswUeT0

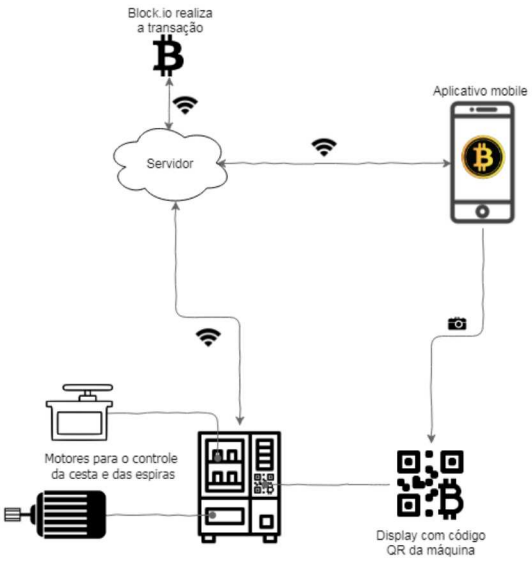


Figura 1: Esquemático geral do projeto

QR da máquina da qual ele quer comprar algum produto e recebe uma lista dos produtos disponíveis. Ele então pode escolher um dos produtos, e então realizar o pagamento para que a máquina dispense o produto e confirme a transação. Caso ocorra algum erro, o usuário poderá notificar e reportar o problema no aplicativo. A *vending machine*, durante esse funcionamento, terá um *display* que irá comunicar ao usuário sobre o estado do processamento de uma transação.

Toda a comunicação dentro do sistema é intermediada por um servidor hospedado em nuvem, esse mesmo servidor também será responsável pela comunicação com a API de cripto-moeda.

Na seção 2 a seguir, são apresentadas as tecnologias utilizadas no desenvolvimento do projeto, e na seção 3, o desenvolvimento de toda a Vending Machine. Ao final do artigo, são apresentados alguns experimentos realizados, e as conclusões do projeto como um todo.

2 Tecnologias Utilizadas

2.1 *Vending Machine*

2.1.1 Servo Motor DS04

O servo motor de rotação contínua DS04 - NFC, apresentado na Figura 2,possui um torque de $60N/cm$ e recebe uma tensão de entrada de $6V - 1A$, e pode ser controlado com o sinal *PWM* do *Raspberry pi* de $3.3V$ [2].

[2] `http://www.ekt2.com/pdf/412_CH_SERVO_MOTOR_SET.pdf`

2.1.2 Motor de passo NEMA 17

Os motores de passo NEMA 17 utilizados são dois $17HS4401$, apresentado na Figura 3, com $4.2kgf$ que operam em $12V/2A$ e um $AK17/1.10F6LN1.8$ com força de $1.2kgf$ que operam em $12V/0.1A$ [1].

2.1.3 Driver Motor de Passo A4988

O *driver* A4988, apresentado na Figura 4, aceita duas entradas do microcontrolador, o *STEP* e *DIR*. A cada borda positiva do sinal de entrada do *STEP*, o *driver* muda o estado para o próximo passo do motor na direção dada no pino *DIR* [2].

2.1.4 *Display TFT* 5LC09

O *Display TFT*, exposto na Figura 5,pode ser conectado ao *Raspberry Pi* diretamente pelos pinos da *GPIO* presentes no microcontrolador, ocupando um total de 26 pinos. Ela pode ser utilizada como um monitor comum a partir do uso do *driver Elecrow-LCD* [3].

Figura 2: Servomotor DS04

Figura 3: Motor de passo NEMA 17

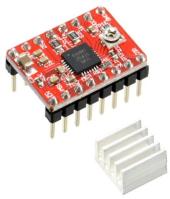


Figura 4: Driver A4988

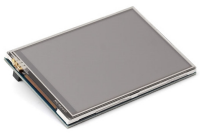


Figura 5: *Display TFT 3.5"*

2.2 Aplicativo móvel

A estação base desse projeto é um aplicativo móvel para *smartphones* com o sistema operacional *Android*. Para o desenvolvimento do aplicativo foi escolhida a plataforma nativa do *Android*, com a linguagem de programação*Kotlin* sendo o ambiente de desenvolvimento o *Android Studio 3.2*

2.3 Servidor

2.3.1 REST

Representational State Transfer (REST), em português Transferência de Estado Representacional é uma arquitetura para a comunicação entre cliente e servidor [3]. Possui requisições do tipo GET e POST, por exemplo, que permitem a transferência de dados e mensagens entre cliente e servidor, sempre através de

[3] `https://github.com/Elecrow-keen/Elecrow-LCD35`

uma requisição do cliente e uma resposta do servidor. É uma arquitetura sem estado, cada requisição é independente das outras.

2.3.2 Spring

Spring [4] é um *framework* desenvolvido para Java, com o propósito de facilitar o projeto e desenvolvimento de aplicações, e possui diferentes módulos.

Um destes módulos é o *Spring Boot*, capaz de criar automaticamente os arquivos de configuração necessários para iniciar um servidor REST, além de tornar fácil a importação de outras dependências do projeto.

Outro módulo é o *Spring Security*, que será utilizado para criptografar as senhas dos usuários.

2.3.3 Heroku

Heroku [5] é uma plataforma para hospedar aplicações na nuvem. Possui suporte para Java, Python, Node.js, entre outras linguagens. Oferece serviço de DNS e fornece diferentes plugins, muitos facilmente integráveis com *Spring*.

2.4 Bitcoin e criptomoedas

O Bitcoin é uma rede descentralizada de pagamentos, em que usuários transacionam entre si usando carteiras digitais, enviando valores de uma carteira para outra. As transações são mantidas em estruturas de dados conhecidas como *blockchain*, que armazenam blocos de transações de acordo com a carteira de origem, a de destino e a quantidade [6].

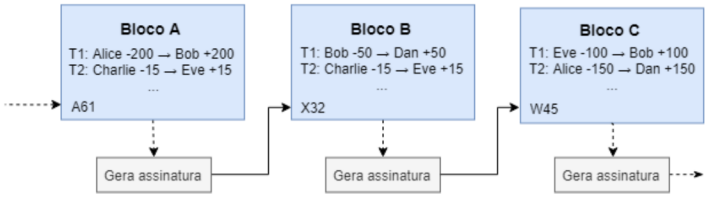


Figura 6: Exemplificação da blockchain

2.4.1 Block.io

A Block.io é um serviço que provê uma interface com a blockchain do Bitcoin e de outras criptomoedas. Através de uma API (*Application Programming Interface*, em português Interface de Programação de Aplicação), ela facilita a implementação de transações ao abstrair o gerenciamento de carteiras e chaves privadas, além de possuir mecanismos para confirmar pagamentos antes mesmo das transações estarem em um dos blocos da *blockchain* [7].

Outra funcionalidade disponível é o uso das testnets de diversas criptomoedas, o que permite usar blockchains de teste durante a fase de desenvolvimento.

3 Desenvolvimento

Explicando brevemente o sistema como um todo, utiliza-se como um material de apoio a Figura 7. Em que o sistema se subdivide em 3 módulos disjuntos, o *Back-end*, a *Vending Machine* e a Estação base, que no caso do nosso trabalho, será um *smartphone Android*.

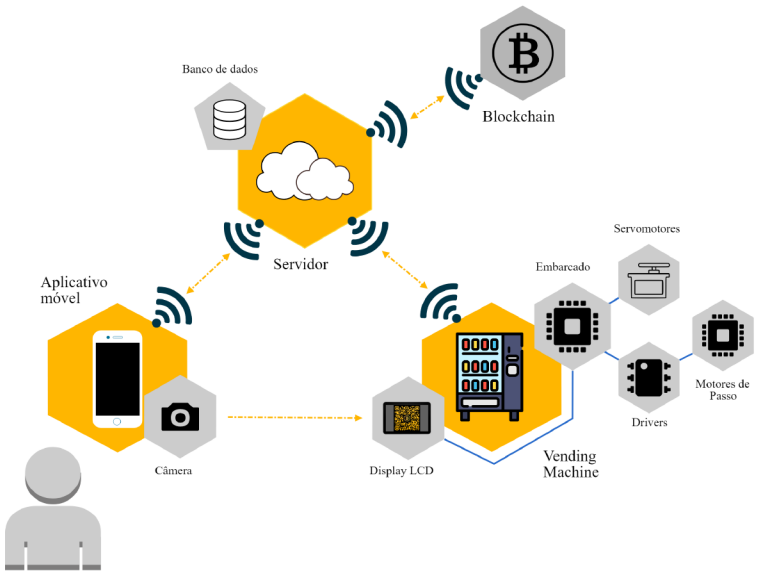


Figura 7: Diagrama em blocos dos módulos e componentes internos do projeto

A *Vending Machine* é responsável pela liberação do produto utilizando um conjunto de motores de passo e servomotores que recebem um sinal do embarcado. Um *display* também é utilizado para atualizar o usuário sobre o andamento da transação e da liberação do produto. Por via dele, a partir de um código *QR*[4] mostrado no *display*, pode-se conectar o aplicativo da estação base a partir de uma câmera de um *smartphone Android* que faz a leitura do código e começa o processo da compra.

A estação base tem como objetivo principal ser a interface central do usuário, ela se dispõe de uma câmera que faz a leitura do código *QR* da máquina e então começa a se comunicar com o servidor,que por sua vez recebe as requisições da estação base, retorna informações pertinentes ao sistema, faz o controle de um banco de dados e processa as transações via a *API* de cripto-moeda.

[4] `https://www.qrcode.com/en/codes/`

A partir disso, pode-se ter uma noção geral do sistema e como é feita a integração entre cada uma das partes do projeto, que serão explicadas mais a fundo nas seções seguintes.

3.1 Embarcado

3.1.1 Controle dos motores de passo

O controle dos motores de passo NEMA 17 foi feito utilizando os *drivers A4988* descritos na seção 2. O sinal de *STEP* é gerado pelo *Raspberry pi* [5] com biblioteca *pigpio*[6] que produz uma curva de aceleração utilizada para aumentar a velocidade do motor de passo ao longo do movimento da cesta. A contagem de passos é feita utilizando os mesmos métodos dessa biblioteca. As conexões simplificadas que são feitas entre o microcontrolador e os motores de passo estão expostas na Figura 8

3.1.2 Controle dos servomotores

Os servomotores dos *dispensers* possuem uma alimentação de 5.5 V gerada pelo conversor *step-down LM2598* usado para o controle da tensão de entrada dos servos. Eles são controlados pelo sinal PWM do *Raspberry pi* gerado via *hardware* pela biblioteca *pigpio*. Como a velocidade dos motores não era perfeitamente igual, foi utilizada uma temporização em *software* para igualar a quantidade que cada motor é rotacionada. As conexões simplificadas entre o microcontrolador e os servomotores está exposta na Figura 9

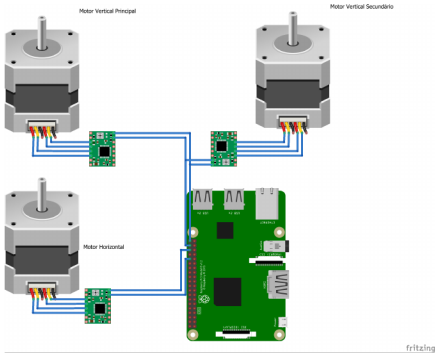


Figura 8: Diagrama simplificado das conexões com os motores de passo

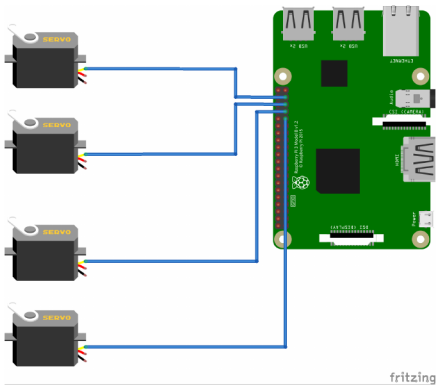


Figura 9: Diagrama simplificado das conexões com os servos

[5] https://www.raspberrypi.org/
[6] http://abyz.me.uk/rpi/pigpio/python.html

3.1.3 Integração do *display* com o projeto

Para utilização do *display* de 3.5" foi usado o *software Elecrow-LCD35*, que permite o *display* exibir imagens no *display*. Com isso, utilizando a biblioteca *TKinter* [7] do *Python*, pode-se ter a exibição das imagens feitas para a interação do usuário e finalizar a interface da máquina. A Figura 10 apresenta a imagem do código QR que fica exposto na máquina em momentos ociosos do sistema e a Figura 11 apresenta a imagem que será exposta quando a máquina completar todo o processo de liberação do produto, completando a transação. Outras imagens foram desenvolvidas para apresentar o processo de dispensa do produto pedido, mas não foram colocadas por brevidade.

Figura 10: Imagem inicial do QRCode

Figura 11: Imagem de transação completa

3.1.4 Eixo horizontal

O eixo horizontal foi construído com uma peça de madeira MDF que segura dois canos de alumínio, sobre os quais é preso um *slider*, construído com 4 rolamentos $22x8mm$ e uma peça de $50x9x9mm$ de MDF. Os rolamentos são colocados em ambos os lados da peça da MDF prendendo-a no eixo. Isso forma o *slider* que é movimentado por um conjunto de polias nas extremidades do eixo com uma correia ao longo do eixo amarrada no *slider* .

3.1.5 Eixo vertical

A estrutura do eixo vertical é dividida em dois eixos diferentes, o eixo principal e o eixo secundário. O eixo principal foi construído de maneira similar ao eixo horizontal, com dois canos e um *slider* com rolamentos em ambos os lados, enquanto o eixo secundário foi feito utilizando apenas um cano de alumínio que apoia um *slider* menor que possui apenas 2 rolamentos em um lado para simplesmente apoiar o eixo horizontal, que então foi então acoplado nessa estrutura e o resultado esta mostrado na Figura 12.

[7] `https://docs.python.org/3/library/tk.html`

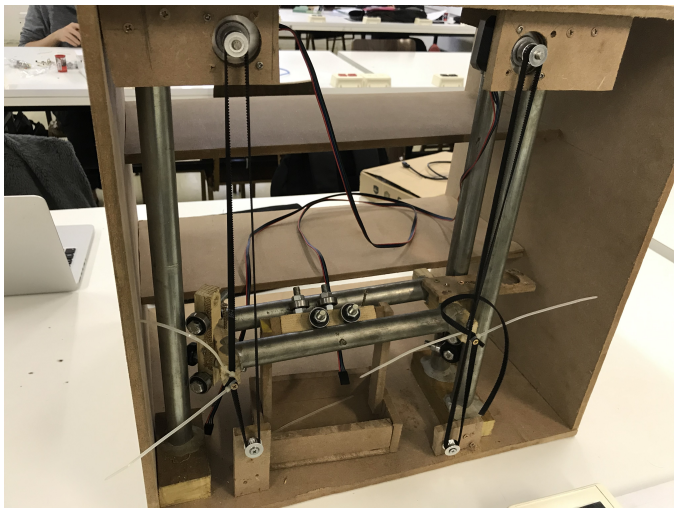


Figura 12: Eixos colocados na estrutura externa da máquina

Figura 13: Estrutura do eixo vertical

A movimentação pelo eixo vertical é feita utilizando dois motores sincronizados que puxam os *sliders* ao longo do eixo, assegurando que a cesta fique sempre alinhada com a horizontal. As polias são acopladas nas extremidades dos eixos e a correia e presa nelas e amarradas nos *sliders* apresentado na Figura 13.

3.1.6 Visor

A partir da construção da mecânica do sistema, foram encaixados o visor e a placa traseira de MDF da *vending machine*. Elas foram feitas com entradas para a mão do usuário, permitindo retirar um produto, uma abertura para o usuário poder ver o funcionamento da máquina e uma abertura para visualizar o *display* usado. Nessa mesma etapa de construção, foram também fixados *LEDs* de $12V$ alto brilho no topo da máquina para iluminar o interior.

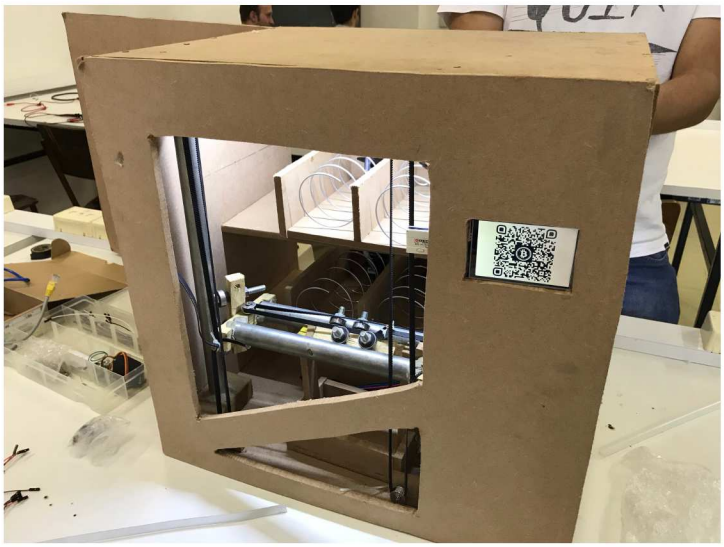


Figura 14: *Vending machine* com todas as placas de MDF fixadas

3.2 Aplicativo móvel

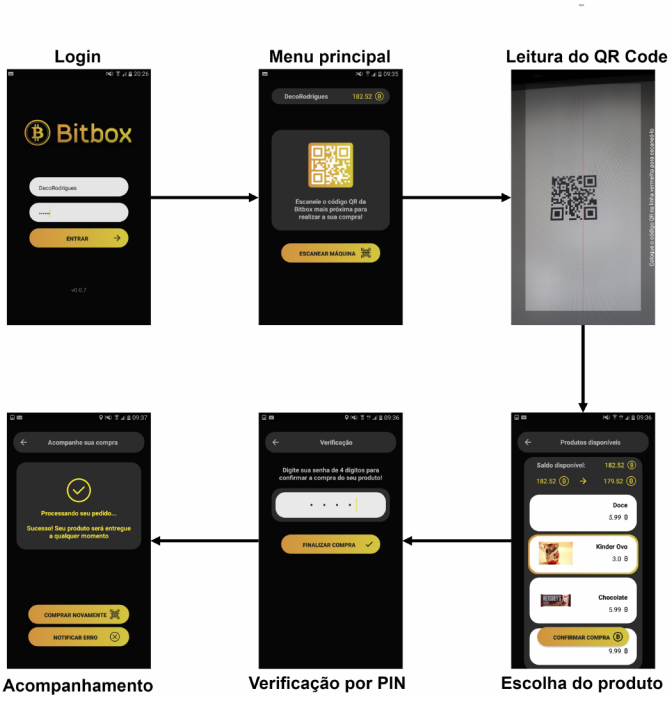
3.2.1 Interface do aplicativo e usabilidade

Projetado para ser a principal fonte de interação com os usuários, o aplicativo móvel possui as telas que processam uma compra.

Figura 15: Fragmento do protótipo

O resultado final do fluxo de navegação, após a implementação das telas, pode ser observado na figura 16.

Figura 16: Fluxo de navegação do aplicativo



3.2.2 Autenticação de usuário no aplicativo

Sendo uma das primeiras integrações feitas no aplicativo, a requisição de autenticação é disparada quando o usuário insere suas credenciais no aplicativo. Depois disso, o aplicativo envia uma requisição para o servidor. A partir dela, o aplicativo tem acesso ao ID do usuário e as informações necessária para se criar uma transação.

3.2.3 Exibição de informações do usuário

Com a integração do servidor com a API do Block.io e ligando um usuário a uma carteira de criptomoeda, o servidor se tornou capaz de retornar também o saldo do usuário nesta carteira. Assim, o aplicativo se tornou capaz de exibir dinamicamente o saldo do usuário, assim como a disposição de elementos visuais e das regras de negócio envolvendo a compra de um produto.

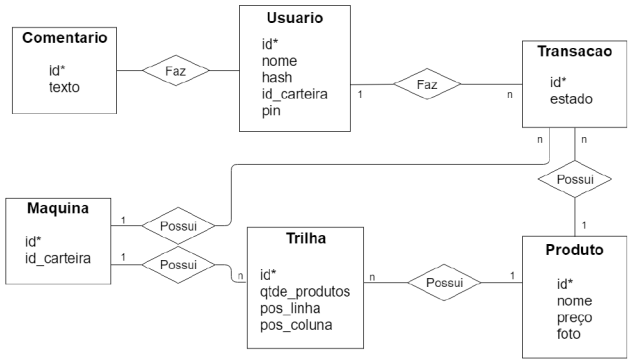
3.3 Servidor

Foi escolhido o *framework Spring* para construir a base do servidor REST, sendo *Java* a linguagem de programação. Para iniciar o projeto, foi utilizado o *Spring Boot*. Foi utilizada a ferramenta *Maven* [8] para compilar o projeto.

3.3.1 Banco de dados

Para manter a integridade dos dados, foi utilizado um banco de dados *PostgreSQL*. A figura 17 apresenta o diagrama de entidade-relacionamento do banco gerado. Esse banco foi criado utilizando *Hibernate* [9], que permite que as entidades e seus relacionamentos sejam criados diretamente no código Java.

Figura 17: Diagrama de entidade-relacionamento



A tabela "Máquinas"representa as máquinas reais, que possuem várias trilhas (*dispensers* ou espiras), sendo que cada trilha possui apenas um produto com uma quantidade específica. Uma transação relaciona um usuário, um produto e uma máquina.

Com o banco de dados configurado, foi necessário criar funções para leitura e escrita. Dessa forma, para cada tabela do banco de dados foram criados *endpoints* - caminhos para acessar alguma funcionalidade do servidor, que permitem adicionar, editar, ler e remover uma linha. Esses *endpoints* servem para facilitar o desenvolvimento das outras partes do projeto, tornando possível atualizar o banco de dados através do próprio servidor.

3.3.2 Login e transações

Estas são as duas funcionalidades mais importantes do servidor: garantir que um usuário possa acessar sua conta e ver seu saldo, e em seguida realizar uma transação com cripto-moeda. Para isso, foi utilizada a *API* da *Block.io*.

Para a transação, é necessário realizar uma assinatura nos dados vindos do *Block.io*. Para realizar essa assinatura foi utilizada a biblioteca provida pelo próprio *Block.io* para Python.

Dessa forma, o servidor (escrito em Java) executa um pequeno *script* em Python, para realizar a comunicação e a assinatura, confirmando assim a transação.

Durante o login, o servidor requisita o saldo do usuário pela *Block.io*. Para uma transação, a requisição é processada pelo servidor para garantir que o produto está disponível e que a máquina não está ocupada com outra transação.

3.3.3 Hospedagem do servidor na nuvem

Para que o servidor seja facilmente acessível pelo cliente e pela máquina, ele deve ser hospedado na nuvem. Foi escolhido o serviço *Heroku*, que fornece tanto a hospedagem do servidor como um banco de dados *PostgreSQL*, além de possuir suporte para *Spring* e para *Maven*, fazendo com que a mudança de um servidor local para nuvem seja simples.

3.3.4 Segurança

Como requisito, foi estipulado que todos os *endpoints* devem possuir segurança nas mensagens. O serviço de hospedagem utilizado oferece segurança TLS[8] em todos os *endpoints*, ou seja, todas as requisições e respostas são encriptadas.

Além disso, as senhas são criptografadas antes de serem salvas no banco de dados, e a autenticação é feita com a senha criptografada, garantindo que em nenhum momento o servidor mantém a senha sem criptografia.

3.3.5 Controle de estoque

O servidor precisa manter a quantidade de cada produto presente nas máquinas, para permitir que o *endpoint* de transação garanta que o produto não está esgotado. Para isso, foi criado um *endpoint* para que o embarcado possa comunicar ao servidor que o produto foi dispensado. Assim, a quantidade de um produto só diminui quando o embarcado confirma a liberação, garantindo coerência em casos de falha na comunicação.

3.3.6 Integração do servidor

As funcionalidades desenvolvidas para o embarcado foram:

- Obter trilha com produto para ser dispensada.
- Obter confirmação de que o produto foi dispensado.

As funcionalidades desenvolvidas para o aplicativo móvel foram:

- Requisitar uma nova transação com uma máquina.
- Receber os dados de um usuário, incluindo saldo.
- Testar o PIN antes de realizar uma transação.
- Obter os produtos presentes em uma máquina.

Cada funcionalidade possui um *endpoint* associado, por onde o embarcado e o aplicativo realizam a comunicação.

[8] Transport Layer Security (TLS), assim como o seu antecessor Secure Sockets Layer (SSL), é um protocolo de segurança projetado para fornecer segurança nas comunicações sobre uma rede de computadores. Maiores informações podem ser encontradas em: https://tools.ietf.org/html/rfc5246.

4 Resultados e Conclusões

4.1 Testes e Resultados

Para que fosse comprovado o funcionamento básico do sistema, foram feitos os seguintes testes:

- Um produto foi colocado em cada *dispenser* e a máquina liberou consecutivamente cada um dos produtos
- Um *dispenser* foi deixado vazio e colocado como esgotado no servidor e o sistema impediu o usuário de pedir e pagar por esse produto
- Pelo menos dois produtos foram colocados em cada *dispenser* e a máquina liberou pelo menos esses dois produtos consecutivamente

Com esses testes os resultados foram os seguintes:

- Todas as liberações foram feitas com sucesso
- O servidor impediu com êxito o usuário de comprar um produto esgotado
- O sistema conseguiu liberar todos os produtos com êxito, com uma taxa de sucesso de 90% em 20 liberações, esse erro é por conta da queda irregular dos produtos e o enrolamento irregular das espiras.

4.2 Problemas

4.2.1 Peso do eixo horizontal

Devido ao peso da estrutura do eixo horizontal, o motor do eixo vertical não tinha força para erguê-lo. A solução mais simples foi alterar a estrutura para suportar mais um motor, colocando assim o motor do eixo horizontal como segundo motor do eixo vertical, tendo dois motores para puxar o eixo horizontal. No eixo horizontal foi colocado um terceiro motor, que não precisa de muita força para movimentar o eixo. Isso demandou tempo para a adaptação da estrutura, além de demandar um motor extra e um novo par de polias, o que atrasou o cronograma em 2 dias úteis.

Além disso, houve certa dificuldade para prender a correia de forma que ela não deslizasse sobre a polia do motor e isso afetou nas tarefas correspondentes no cronograma.

4.2.2 Problema com um dos drivers A4988

Um dos *drivers* de controle do motor de passo começou a ter um funcionamento inesperado durante testes do protótipo. Foi necessário substituir o *driver* em diversas ocasiões, o que acabou afetando ligeiramente no orçamento e no cronograma, como foi previsto no planejamento de riscos.

4.2.3 Heroku e Banco de dados H2

O banco de dados H2 é um banco de dados que trabalha com um arquivo local, e foi utilizado inicialmente para os testes com servidor local. O *Heroku* algumas vezes reinicia o servidor e recompila todo o projeto, apagando assim os dados anteriormente inseridos no banco. Devido à isso, o banco de dados utilizado foi alterado para o *PostgreSQL*, que mantém os dados quando o servidor é reiniciado. Essa mudança de banco de dados não era prevista, mas o *Heroku* possui suporte para o banco de dados, e a mudança foi fácil e demandou pouco tempo, o que não atrasou o cronograma.

4.2.4 Block.IO

Devido a uma mudança na API do *Block.IO* durante a execução do projeto, se tornou necessário assinar o conteúdo da transação antes de enviar. Para isso, ao invés de assinar o conteúdo por conta própria, o que demandaria muito tempo e poderia atrapalhar o cronograma, foi usado um *script* em *Python* para fazer a assinatura, utilizando uma biblioteca provida pela própria *Block.IO*. Assim, foi necessário refazer o código de transações para acomodar as mudanças da API, o que demandou mais tempo para as tarefas correspondentes e afetou ligeiramente no tempo gasto no cronograma.

4.2.5 Placa WIFI do Raspberry

A placa *Raspberry Pi* 3b, utilizada no projeto, não possui placa de rede *wireless* interna. Foi preciso utilizar um adaptador externo USB para conseguir usar a rede sem fio. Apesar disso, o reconhecimento do driver do adaptador pelo sistema operacional foi automático e não causou problemas que podiam demandar muito tempo.

4.3 Conclusão

Para o desenvolvimento do projeto houve a necessidade da integração de várias áreas de conhecimento adquiridas durante o curso, tanto na parte de software como de hardware, na parte de desenvolvimento de aplicação para o embarcado, o desenvolvimento do servidor, o aplicativo móvel, além da integração entre essas partes e do embarcado com os motores. Além disso o desenvolvimento proporcionou o aprendizado de novas ferramentas e tecnologias, como o *Springboot* e *Hibernate* e o uso do *Raspberry Pi*. Houveram alguns problemas, como explicado na seção anterior, tanto na parte de software e eletrônica, como na parte mecânica principalmente, devido à falta de experiência da equipe nessa parte. Isso já era esperado e previsto no cronograma. Com isso viu-se a real necessidade de um cronograma com horas de folga e uma análise de riscos com alternativas para contornar os problemas sem comprometer o tempo de entrega e o orçamento do projeto.

O tempo total previsto no início do projeto pelo cronograma incluindo a folga de 30% era de 421 horas (84.2 horas por integrante), e o tempo total realmente gasto foi de 403 horas (80.6 horas por integrante). A Tabela 1 abaixo mostra o preço estimado de cada produto no início do projeto, o que foi realmente gasto e os totais.

Tabela 1: Orçamento e gasto do projeto.

Produto	**Preço Estimado**	**Gasto Real**
Placa de acrílico	R$35,00	R$ 50,00
Placas de MDF 9mm	R$57,00	R$ 96,00
Motores Nema 17mm	R$150,00	R$225,00
Servomotor DS04	R$200,00	R$220,00
Raspberry Pi 3 b	R$220,00	R$220,00
Driver A4988	R$40,00	R$60,00
Display LCD	R$100,00	R$ 100,00
Fita LED	R$20,00	R$ 12,00
Polias 6mm	R$26,00	R$52,00
Correias 6mm	R$20,00	R$40,00
Kit 10 Rolamentos	R$22,00	R$ 20,00
Conversor Buck Step Down	R$25,00	R$ 25,00
Total	R$915,00	R$ 1110,00

Pode-se ver que houveram gastos que excederam aquilo que era esperado do projeto, isso foi causado pela mudança do mecanismo da cesta ao longo do desenvolvimento, o que necessitou de mais um motor de passo, mais um par de polia e correia e mais um driver 4988 o que aumentou significativamente os custos do projeto. Os custos dos materiais como o MDF e o acrílico foram subestimados na fase de projeto também, mas os custos reais do projeto também foram afetados pela quantidade maior que foi comprada.

A divisão do projeto e distribuição de tarefas em partes independentes facilitou bastante o desenvolvimento, já que cada pessoa pôde focar mais em seu trabalho, principalmente no início. Apesar disso todos os integrantes tinham ciência do funcionamento de todas as partes do projeto. A interação da equipe por meio de reuniões, a integração das partes do projeto e montagem final tendo a presença dos integrantes, também foram importantes, já que um problema qualquer ao testar era mais facilmente identificado pelo responsável por aquela parte.

Pode-se dizer que o projeto contribuiu para o crescimento dos membros da equipe, não só pela parte técnica aprendida e colocada em prática, mas também por fazer perceber a importância de um cronograma e um plano de resposta aos riscos para conseguir contornar os problemas, além da necessidade de se comunicar e cooperar com os outros membros do grupo.

Referências

[1] AKIYAMA MOTORS. *Catálogo Motores de Passo Akiyama*. [S.l.], 2017. Disponível em: <https://www.neomotion.com.br/wp-content/uploads/2017/07/Cat%C3%A1logo-Datasheet-dos-motores-de-passo-R01.pdf>. Acesso em: 23 jun. 2019.

[2] ALLEGRO MICROSYSTEMS. *Datasheet Pololu A4988 Microstepping driver*. [S.l.], 2014. Disponível em: <https://www.pololu.com/file/0J450/a4988.pdf>. Acesso em: 23 jun. 2019.

[3] WHAT is REST. 2019. Disponível em: <https://restfulapi.net/>. Acesso em: 23 jun. 2019.

[4] SPRING Framework. 2019. Disponível em: <https://spring.io/>. Acesso em: 23 jun. 2019.

[5] HEROKU. 2019. Disponível em: <https://www.heroku.com/>. Acesso em: 23 jun. 2019.

[6] ANTONOPOULOSS, A. *Mastering Bitcoin*. O'Reilly, 2017. Disponível em: <https://github.com/bitcoinbook/bitcoinbook>. Acesso em: 23 jun. 2019.

[7] BLOCK.IO. Documentation, *Block.io API Documentation*. 2019. Disponível em: <https://block.io/api/>. Acesso em: 23 jun. 2019.

[8] APACHE. *Apache Maven*. 2019. Disponível em: <https://maven.apache.org/>. Acesso em: 23 jun. 2019.

[9] HIBERNATE. *Hibernate*. 2019. Disponível em: <https://hibernate.org/>. Acesso em: 23 jun. 2019.