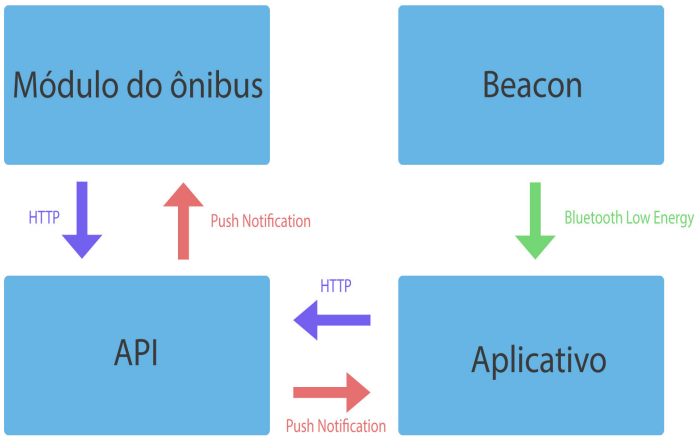
Etapas e dificuldades

Victor Vieira Paulino Arthur Cicuto Pires

29 de agosto de 2017

1 Protocolos de Comunicação



1.1 HTTP

Hypertext Transfer Protocol é um protocolo baseado em requisições. Quando um cliente necessita de uma informação, ele solicita para o servidor que retorna uma resposta. Sua especificação permite requisições do tipo *GET, POST, DELETE*, dentre outros. Sua principal vantagem é não haver uma conexão aberta a todo momento para trafegar mensagens, permitindo que conexões e informações trafeguem apenas quando necessário. O formato para trafego das informações neste trabalho, por meio deste protocolo é a notação *JSON*.

Para o cenário deste projeto, tanto o aplicativo quanto o módulo do ônibus estão em cenários não favoráveis para o trafego de informação em grande escala, tendo em vista a baixa qualidade das redes 3G/4G dos smartphones e das redes WiFi que possuem nos ônibus.

1.2 Push Notification

Push Notification é um serviço de entrega de mensagens, parecido com *SMS* (*Shot Message Service*), mas que usa exclusivamente a internet para entregar. Cada plataforma possui seu próprio serviço *Push*. Um bom uso deste serviço, é quando o emissor precisa enviar algo para o destinatário, sem a necessidade do destinatário ter solicitado antes, como ocorre no *HTTP*.

Neste projeto temos duas situações que a tecnologia é conveniente: primeiro, existe a necessidade de avisar o motorista que é necessário, em um dado momento, parar no próximo ponto para um deficiente visual. Segundo, precisamos avisar ao deficiente visual que seu ônibus já chegou e ele pode se dirigir a ele.

Nestes dois cenários precisamos avisar os dispositivos sobre algum evento e não temos uma conexão aberta constantemente como eles. Fazendo o serviço de *Push Notification* ser a melhor escolha.

Uma alternativa ao uso deste serviço são plataforma de *Realtime Database*. Eles funcionam de forma parecida com o protocolo *MQTT*, quando há alguma alteração em algum nó, os *subscribers* são notificados sobre o novo dado.

1.3 Bluetooth Low Energy

O *Bluetooth* é uma tecnologia de transmissão dados. Na sua versão 4.0+ ele se tornou *BLE* (ou *Bluetooth Smart*), trazendo a transmissão de dados com baixo consumo de energia.

Os pontos de ônibus não costumam possuir energia elétrica, com isso, surge a necessidade de uma tecnologia que tenha um moderado consumo de eletricidade. Com a necessidade do baixo consumo de energia e o envio constante, o *BLE* em seu modo *Beacons* ativado, se

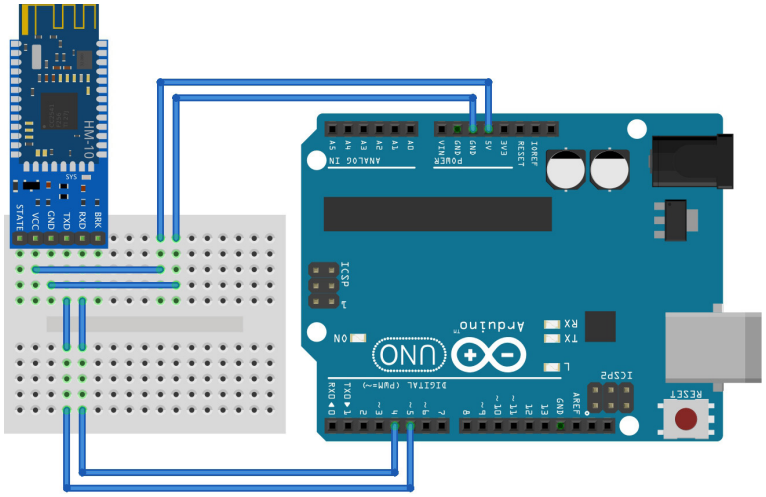
demonstrou ser a melhor alternativa suprindo todas as necessidades do projeto.

2 Módulo do Ponto de Ônibus

2.1 Hardware

1. HM-10 - Bluetooth 4.0 BLE module
2. Arduino Uno

Arduino Uno é utilizado apenas como ponte para configurar o módulo HM-10. A ligação deve seguir o diagrama abaixo.



2.2 Software

- Arduino IDE 1.8.3 ou superior.

2.3 Configuração

Conecte o arduino Uno ao computador e compile o código abaixo utilizando a IDE do arduino.

```
#include <SoftwareSerial.h>
```

```
// SoftwareSerial (TX / RX)
SoftwareSerial mySerial(7, 8);

// Serial value
String val;

// Receive info
boolean receiveInfo = false;

// Bauds for detectBleBaudRate function
long bauds[] = {
    // major
    9600, 57600, 115200,

    // others
    19200, 38400, 4800, 2400, 1200, 230400
};

// Detect BLE Baud Rate
bool detectBleBaudRate() {
  Serial.println("-----------------------------");
  Serial.println("Detecting BLE baud rate:");

  for (int i=0; i < (sizeof(bauds) / sizeof(long)); i++) {
    Serial.write("Checking baud rate: ");
    // Checking baud rate
    long cur_baud = bauds[i];
    Serial.println(cur_baud, DEC);

    mySerial.begin(cur_baud);
    mySerial.write("AT");
    mySerial.flush();

    delay(50);

    String response = mySerial.readString();

    if (response == "OK") {
```

```
      Serial.println("-----------------------------");
      Serial.print("BLE Baud Detected: ");
      Serial.println(cur_baud);
      Serial.println("-----------------------------");

      return true;
    } else {
      mySerial.end();
    }
  }

  detectBleBaudRate();
}

// Uncomment function to wake up bluetooth or discover BLE Baud
    Rate
void setup() {
  mySerial.begin(9600);
  Serial.begin(9600);
}

void wakeUp() {
  Serial.println("-----------------------------");
  for (int i = 0; i < 30; i++) {
     mySerial.write("
        12345678901234567890123456789012345678901234567890123456789012345678
        ");
  }

  Serial.println("Wake up command has been sent to HM-10");
  Serial.println("-----------------------------");
}

// Reads strings that are sent to mySerial
void loop() {
  while (Serial.available() > 0) {
     delay(10); // Delay needed to recognize key and value
        properly
```

```
      String val = Serial.readString();
      Serial.print("Serial sent: ");
      Serial.println(val);

      // Receive info from HM-10
      receiveInfo = true;

      // If you need to wake up the HM-10 sensor, send wake command
      if (val == "wake") {
         wakeUp();
         receiveInfo = false;
      }

      // If you want to detect BLE Baud Rate, send detect command
      if (val == "detect") {
         detectBleBaudRate();
         receiveInfo = false;
      }

      mySerial.print(val);
      mySerial.flush();

      delay(50);

      if (receiveInfo) {
          // Read string from HM-10 (Software Serial)
          String response = mySerial.readString();

          Serial.print("mySerial received: ");
          Serial.println(response);

          Serial.println("------------------------------");
      }
   }
}
```

Após compilador, utilizando o Serial Monitor da IDE, execute os comandos AT na seguinte ordem:

Obs: Quanto menor o tempo de envio, maior a economia de energia.

1. AT+RENEW //Coloca nos padrões de fábrica
2. AT+RESET //Reinicia para aplicar os padrões de fábrica
3. AT+MARJ0xNNNN //Define o valor Marjor
4. AT+MINO0xNNNN //Define o valor Minor
5. AT+NAMEMeuBeacon //Define o nome do Beacon
6. AT+ADVI5 //Define tempo de envio. 5 = 546.25 millisegundos
7. AT+ADTY3 //Define como não pareável
8. AT+IBEA1 //Habilita como Beacon
9. AT+DELO2 //Configura para apenas emitir sinal
10. AT+PWRM0 //Habilita auto-sleep para economizar energia
11. AT+RESET

Após configurado, pode ser ligado em uma bateria 3v para utilização.

2.4 Referências

HM-10 Bluetooth 4.0 BLE module Datasheet

Arduino IDE

Repositório da Metractive - Como construir Beacons

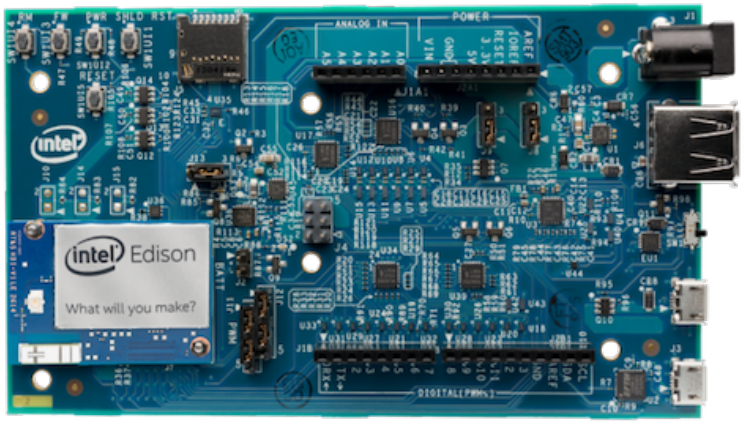
3 Módulo do Ônibus

3.1 Hardware

- Intel Edison

- Raspberry Pi 3
- Tela LCD 7”(em breve)
- NEO u-blox 6 GPS Modules

3.1.1 Intel Edison



Inicialmente foi adotado o Intel Edison com placa de expansão arduino. Foi escolhido devido a fácil acesso a um exemplar e ótimo hardware. Ele conta com WiFi, Bluetooth, portas I/O, processador Intel Atom de 500 MHz, 1GB de memória RAM DDR3 e 4GB eMMC. Sua utilização foi fácil e não obtivemos nenhuma dificuldade em instalar o sistema que escolhemos.

Problemas encontrados em adotar como solução:

Preço

Embora tenha um ótimo hardware e uma empresa séria por trás da sua construção, o preço, em 07/2017, que gira em torno de R$ 600,00, não justifica sua adoção como a melhor solução para o projeto já que existem alternativas com preços melhores e bom desempenho.

Ausência de controlador gráfico

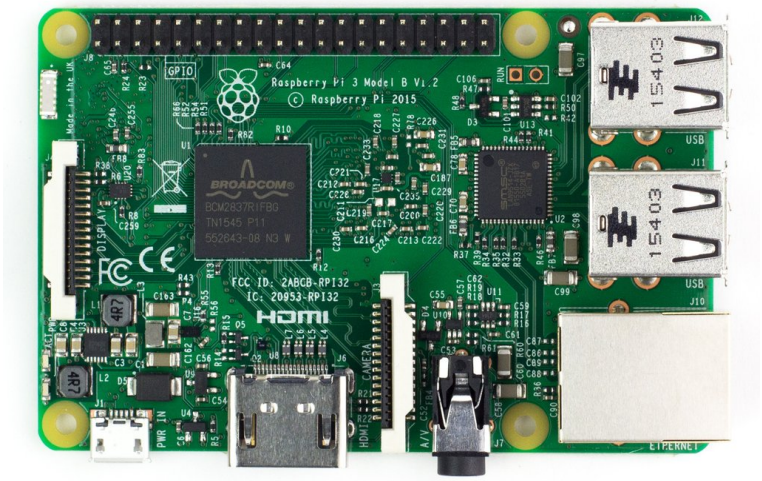
Uma das features do projeto é emitir alertas visuais para o motorista por meio de

telas LCDs. A placa Intel Edison nos permite fazer alertas visuais utilizando LEDs e afins.

Descontinuidade da placa pela Intel

em 07/2017, a Intel anunciou a descontinuidade do desenvolvimento de algumas placas que fabrica. O Intel Edison foi uma delas.

3.1.2 Raspberry Pi 3



Testes realizados no Raspberry Pi 3 demonstraram ser uma boa alternativa ao Intel Edison. Foi fácil a instalação do sistema e a placa vem com saída HDMI permitindo utilizar telas LCD para fazer os alertas visuais. Seu preço, em 08/2017, gira em torno de R$ 150,00, 1/4 do preço do Intel Edison. Seu hardware contém boas especificações:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board

- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A

Embora tenha um hardware com especificações superiores ao Intel Edison, não houve ganho de desempenho ao rodar o sistema, devido a ausência de algoritmos complexos no sistema. Assim, a grande vantagem de se utilizar o Raspberry Pi 3 ao invés do Intel Edison, é seu baixo custo e recurso de chip gráfico.

3.1.3 Módulo NEO u-blox 6 GPS

Para realizar o rastreamento do ônibus foi adotado o módulo NEO u-blox 6 GPS Modules, devido a compatibilidade com as placas que contém o sistema embarcado e preço acessível.

Localização

Uma característica desse módulo é trabalhar com GPS, fazendo comunicação direta com no mínimo 3 satélites para triangular sua posição com mais precisão. Alguns módulos disponíveis no mercado trabalham com A-GPS, que usam torres de telefonia móvel para conhecer sua posição. O uso do GPS trás maior precisão, porém demora mais para estabelecer conexão com satélites. O A-GPS fornece a localização com menor tempo, porém com menor precisão e a um custo mais alto.

Comunicação

O módulo realiza comunicação UART (Universal Asynchronous Receiver/Transmitter), o que permite fácil comunicação com as placas utilizadas para testes.

Preço

Seu preço, em 08/2017, gira em torno de R$ 60,00 e pode ser encontrado com facilidade na internet para venda.

3.2 Software

3.2.1 Sistema Operacional

Android Things Em 2016 o Google anunciou o Android Things, uma versão do Android voltada para IoT (Internet of Things). Ele é, atualmente, uma versão do Android Marshmallow reduzida. Sua escolha foi devido a facilidade de embarcar em placas como o Raspberry Pi e Intel Edison, e a variedade de recursos que já estão disponíveis no SO que facilitam o desenvolvimento do módulo, como o recurso LocationManager. **[Detalhar mais essa parte]**

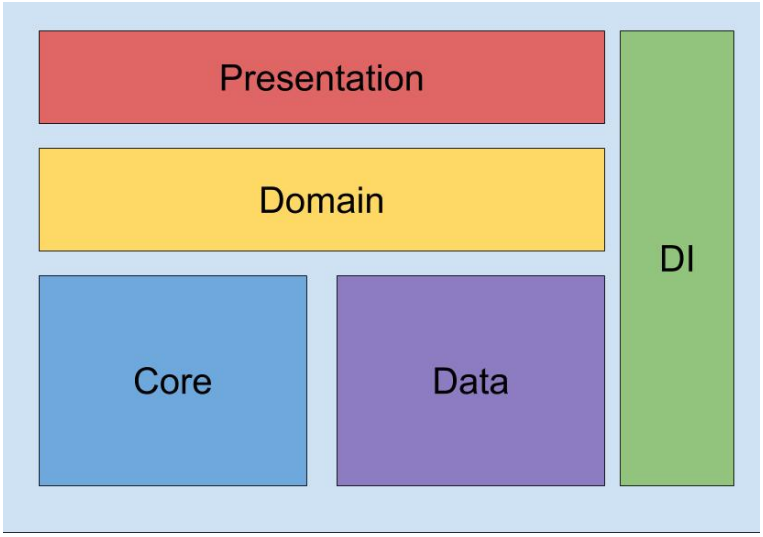
3.2.2 IDE

Foi escolhido o Android Studio como IDE do projeto. Ela é desenvolvida pela IntelliJ e tida pelo Google como ferramenta oficial de desenvolvimento para aplicativos Android.

3.2.3 Linguagem

O Google tem duas linguagens de primeiro nível para desenvolvimento Android: Java e Kotlin. Para esse projeto adotamos a linguagem Kotlin, que possui sintaxe muito simplificado em comparação ao Java. Embora Java tenha sido a primeira linguagem oficial para desenvolvimento, Kotlin oferece acesso aos mesmo recursos do sistema. Algumas bibliotecas disponíveis, desenvolvidas por terceiros, ainda não migraram para o Kotlin, obrigando a implementar algumas classes em Java. Como Kotlin tem interoperabilidade com Java, não existe nenhum impeditivo de utilizar Kotlin e eventualmente alguma classa Java.

3.2.4 Arquitetura



Para desenvolvimento do software, foi adotado o padrão *Clean Archtecture*. É um padrão que visa um maior desacoplamento das classes e distruibui bem as responsabilidades.

Core Não contém nenhuma lógica de negócio. Esta camada provê informações comuns, como

configurações estáticas da placa a toda a aplicação. Possui também algumas classes e interfaces bases.

Data Responsável por prover dados para toda aplicação. Ela adota o Padrão de Arquitetura *Repository*, tendo uma interface de acesso aos dados. Uma grande vantagem em utilizar essa camada com esse padrão de arquitetura, é o respeito a responsabilidade única, um dos princípios do *SOLID*. Ela encapsula toda lógica de busca de dados, assim, caso uma classe precise de algum dado específico, ela solicita através da interface de comunicação e a classe que implementa a interface, cuida de toda lógica de busca de dado, seja um dado armazenado localmente, em cache ou em um servidor remoto. Tudo fica transparente para a classe que solicitou o dado.

Domain Esta camada encapsula toda regra de negócios da aplicação. Toda vez que é necessário realizar processamentos em dados para satisfazer funcionalidades, é feito por esta camada.

Presentation Responsável por toda interface gráfica. Toda lógica de criação de telas e interceptação de interações do usuário com o aplicativo, é feito aqui. Quando é necessário procurar dados para exibir ao usuário, é feito solicitações deles para a camada Domain ou Data para que seja possa exibir os dados.

3.3 Referências

Site Oficial Intel Edison

Datasheet Intel Edison

Anúncio do fim da produção do Intel Edison

Site Oficial Raspberry Pi

Datasheet Raspberry Pi 3

Datasheet NEO u-blox 6 GPS Modules

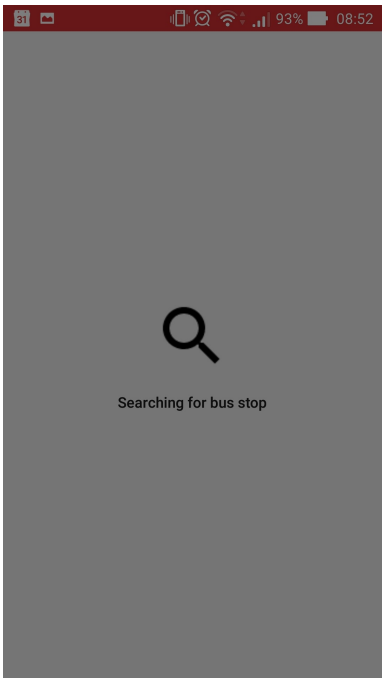
Site Oficial Android Things

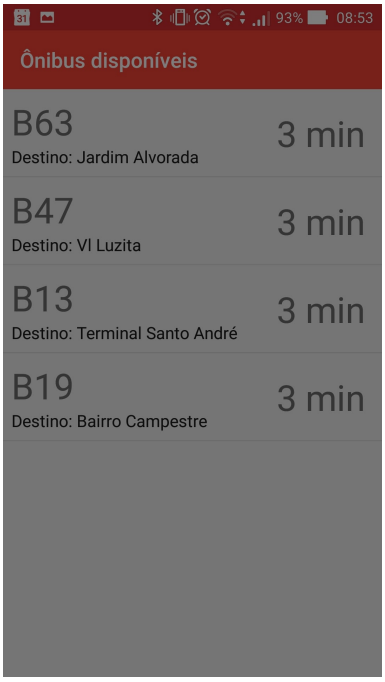
Configuração do Android Things no Intel Edison

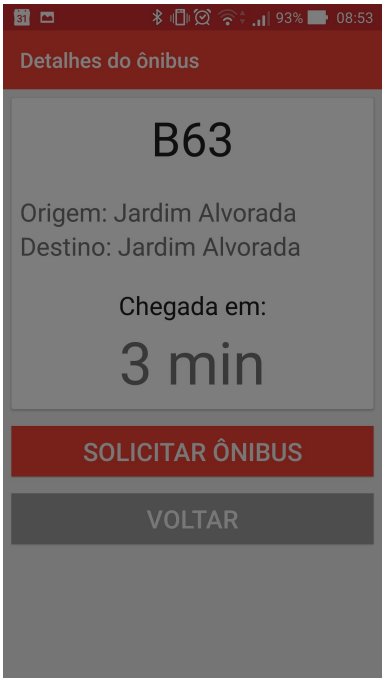
Configuração do Android Things no Raspberry Pi 3

4 Aplicativo

4.0.1 Telas







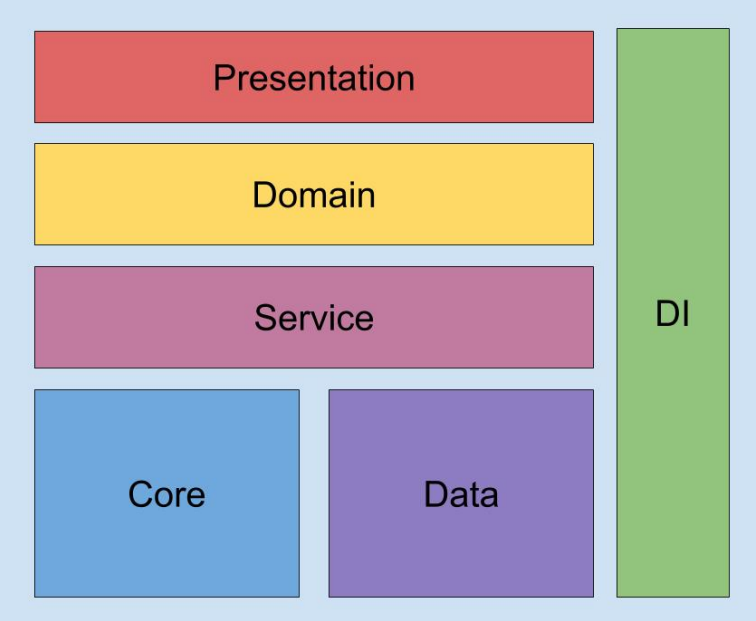
4.0.2 IDE

Foi escolhido o Android Studio como IDE do projeto. Ela é desenvolvida pela IntelliJ e tida pelo Google como ferramenta oficial de desenvolvimento para aplicativos Android.

4.0.3 Linguagem

O Google tem duas linguagens de primeiro nível para desenvolvimento Android: Java e Kotlin. Para esse projeto adotamos a linguagem Kotlin, que possui sintaxe muito simplificado em comparação ao Java. Embora Java tenha sido a primeira linguagem oficial para desenvolvimento, Kotlin oferece acesso aos mesmo recursos do sistema. Algumas bibliotecas disponíveis, desenvolvidas por terceiros, ainda não migraram para o Kotlin, obrigando a implementar algumas classes em Java. Como Kotlin tem interoperabilidade com Java, não existe nenhum impeditivo de utilizar Kotlin e eventualmente alguma classa Java.

4.0.4 Arquitetura



Para desenvolvimento do software, foi adotado o padrão *Clean Architecture*. É um padrão que visa um maior desacoplamento das classes e distruibui bem as responsabilidades.

Core Não contém nenhuma lógica de negócio. Esta camada provê informações comuns, como configurações estáticas da placa a toda a aplicação. Possui também algumas classes e interfaces bases.

Data Responsável por prover dados para toda aplicação. Ela adota o Padrão de Arquitetura *Repository*, tendo uma interface de acesso aos dados. Uma grande vantagem em utilizar essa camada com esse padrão de arquitetura, é o respeito a responsabilidade única, um dos princípios do *SOLID*. Ela encapsula toda lógica de busca de dados, assim, caso uma classe precise de algum dado específico, ela solicita através da interface de comunicação e a classe que implementa a interface, cuida de toda lógica de busca de dado, seja um dado armazenado localmente, em cache ou em um servidor remoto. Tudo fica transparente para a classe que solicitou o dado.

Service Provê serviços para qualquer camada. No caso do aplicativo, a implementação do serviço de voz fica neste pacote e é injeta pelo pacote de Injeção de Dependências.

Domain Esta camada encapsula toda regra de negócios da aplicação. Toda vez que é necessário

realizar processamentos em dados para satisfazer funcionalidades, é feito por esta camada.

Presentation Responsável por toda interface gráfica. Toda lógica de criação de telas e interceptação de interações do usuário com o aplicativo, é feito aqui. Quando é necessário procurar dados para exibir ao usuário, é feito solicitações deles para a camada Domain ou Data para que seja possa exibir os dados.

DI Este projeto utiliza o padrão de arquitetura *Injeção de Dependências*. Esta camada provê todas dependências, fazendo a implementação mais limpas nas outras classes, já que não precisam saber como instânciar uma classe, apenas usam.

4.1 Áudio Descrição

Uma das funcionalidades do aplicativo é descrição da tela que o deficiente está. O *TalkBack* fala para o usuário em qual componente ele está tocando, porém, não descreve em qual tela ele acabou de entrar. A implementação por áudio descrição foi simples com uso da API nativa *TextToSpeech*, onde podemos passar textos personalizados e o serviço se encarrega de sintetizar a voz.

O uso de uma camada de DI (Injeção de Dependências) facilitou o processo de implementação, fazendo ela na camada de serviço e configurando a instanciação no padrão *Singleton* para que todos que vão utilizar (nesse caso são os *presenters*), apenas solicitem a instância sendo passada por construtor.

5 *Web service*

5.1 Linguagem

Para o desenvolvimento do *web service* foi escolhida a utilização da pilha MEAN, que engloba quatro tecnologias para desenvolvimento *web* que possuem como base a linguagem JavaScript.

5.1.1 JavaScript

JavaScript é uma linguagem de programação *client-side*, utilizada para manipular os comportamentos de uma página, controlando o HTML e o CSS. Outra característica dela, é que ela é uma linguagem orientada à eventos. Para explicar melhor o que são eventos, é importante citar que uma página HTML utiliza tags para representar seus elementos, podendo conter menus, botões e formulários em seu corpo. Cada elemento possui alguns atributos, sendo alguns desses atributos de eventos, como por exemplo o *onClick* que realiza alguma função caso o elemento referente seja clicado pelo usuário. Tais funções podem ser desenvolvidas em JavaScript, entrando aqui para dizer qual comportamento a página terá ao disparo do evento.

5.2 *MEAN stack*

A pilha *MEAN* é um conjunto de *frameworks* desenvolvidos em JavaScript, que englobam os lados do cliente, do servidor e do banco de dados. Por possuírem a mesma linguagem como base, os elementos dessa pilha contam com uma maior produtividade no desenvolvimento. MEAN é um acrônimo para MongoDB, Express, Angular e Node.

5.2.1 MongoDB

É um banco de dados não relacional com uma escalabilidade muito boa. Ele utiliza conceitos de *collections* e *documents* em sua construção. As *collections* são equivalentes aos bancos de um ambiente que utiliza o SQL. Já os *documents*, se equivalem aos registros de cada banco. Os dados são guardados em arquivos similares aos de formato JSON (*JavaScript Object Notation*). Outro item importante sobre o MongoDB é o fato de ser schemaless, tornando-o bem flexível em relação a inclusão de dados diferentes em uma mesma *collection*, fazendo com que a validação de dados fique nas mãos dos desenvolvedores.

5.2.2 Express

É um framework que ajuda na organização de sua aplicação, caso use a arquitetura MVC, no lado do servidor. Uma de suas funções é a de facilitar a criação e manutenção de rotas, realizando uma configuração inicial com os caminhos para os *controllers, models* e *views* utilizados pela sua aplicação, além de informar os dados de configuração do servidor.

5.2.3 Angular

Framework utilizado no lado do cliente. Possui um conjunto adicional de atributos para as páginas HTML, passando parte do processamento dos dados da página para o lado do cliente. Isso possibilita a criação de interfaces dinâmicas e assíncronas além de diminuir a carga de processamento do servidor.

5.2.4 Node

Plataforma principal para o funcionamento da pilha MEAN. Ele utiliza o gerenciador de pacotes npm para organizar as bibliotecas utilizadas pela sua aplicação. É ele que realiza a conexão com servidores e diz quais bancos de dados serão utilizados pela aplicação.

5.3 Dificuldades

Apesar de simples de aprender, devido à grande quantidade de métodos para se realizar os mesmos processos, fica um pouco difícil para assimilar quais os arquivos que devem ser modificados para o funcionamento adequado da aplicação.

5.4 Referências

Atributos de eventos
Guia introdutório sobre JavaScript

ALMEIDA, Flávio. MEAN - Full stack JavaScript para aplicações web com MongoDB, Express, Angular e Node. ed. Casa do Código, 2016.

Falar sobre o MEAN stack e sobre as dificuldades que estou encontrando sobre como trabalhar com cada tecnologia da pilha.