

APOSTILA: MYSQL, PHP, ANGULAR E IONIC

Feita por: Hudson Moreira Guimarães dos Santos;

E-mail: hudymoreira@gmail.com

Objetivo: Cria uma aplicação web e mobile que se comuniquem através do mesmo web service. Um banco de dados MySql será criado, o PHP fará toda a manipulação dos dados. Uma aplicação web feita em Angular e outra aplicação mobile feita em Ionic irão transmitir e receber dados com PHP.

Pré-requisitos: Para o total aproveitamento desta apostila:

Ter acesso à um banco de dados MySql ou ser capaz de configurar um; *(1)

Ter acesso à um servidor Apache com PHP instalado ou ser capaz de configurar um; *(2)

Ter em seu computador instalado o Node.js, Visual Studio Code, Angular e o Ionic; *(3)

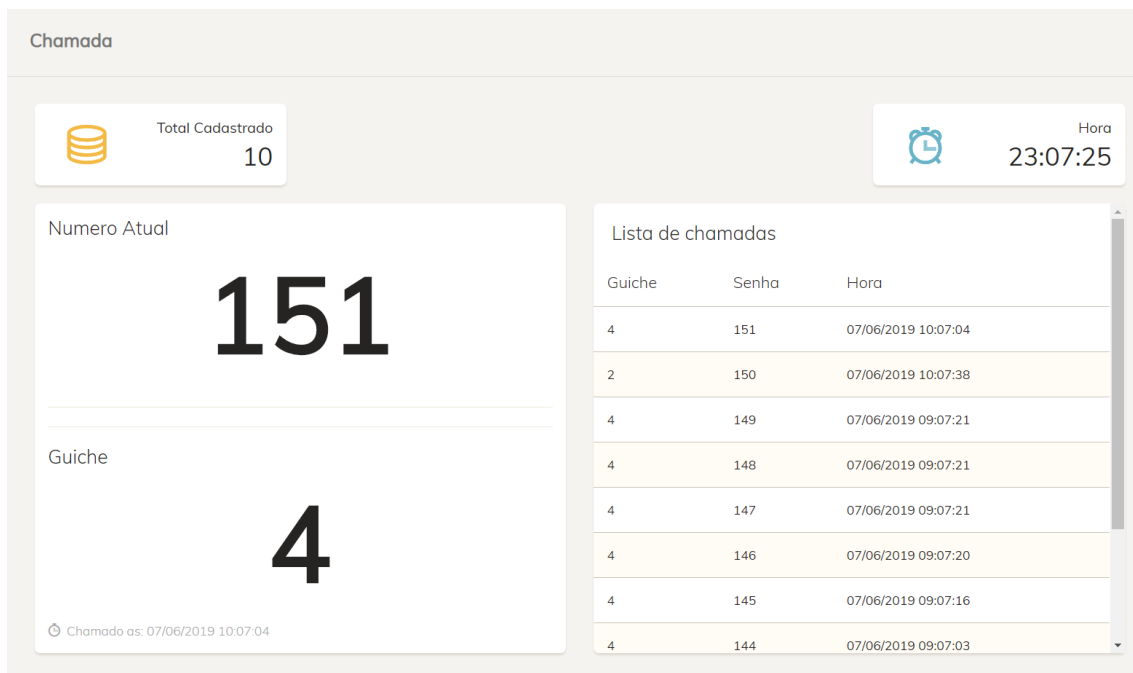
Ter conhecimento básico de HTML e Java Script;

Ter noções muito básicas de Banco de Dados e PHP;

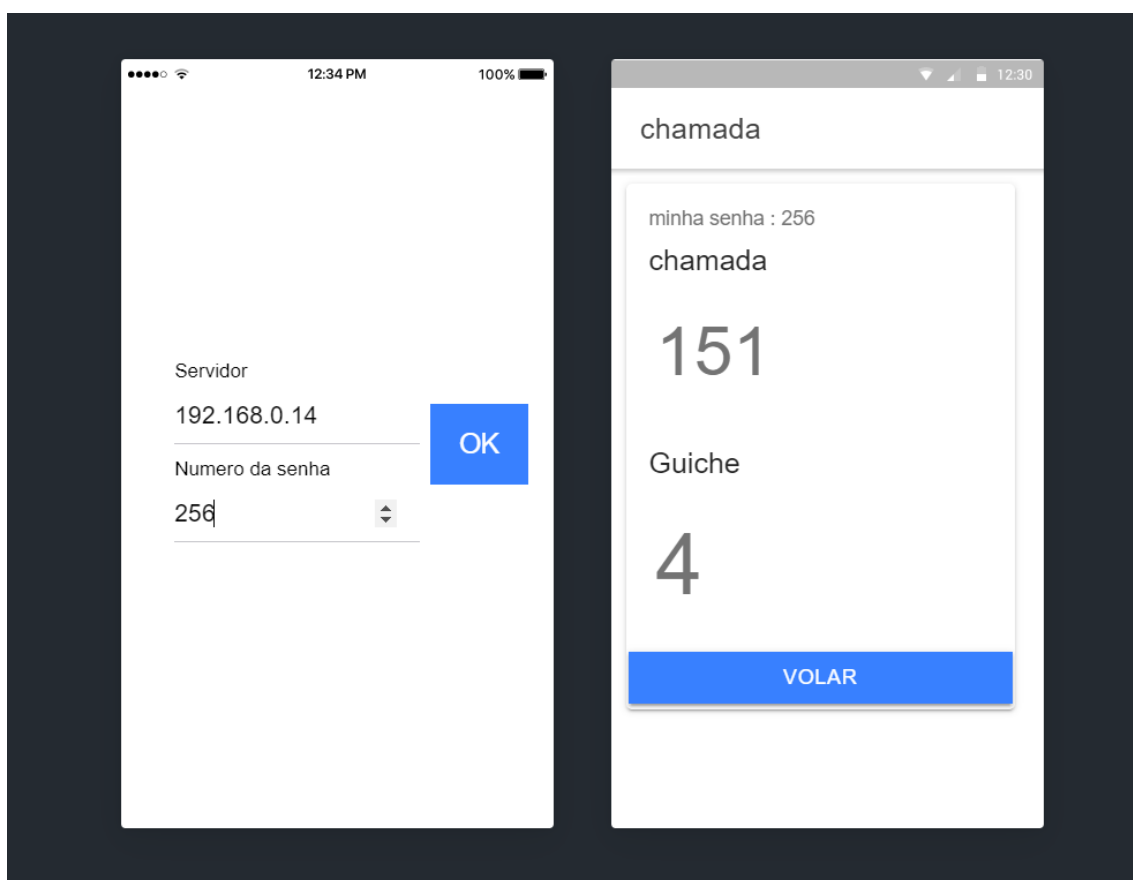
Projeto Base: Para esta apostila foi criado um projeto base onde teremos um cadastro de pacientes (figura 1) para uma clínica odontológica, teremos um painel de chamadas (figura 2) com uma “senha” e um aplicativo mobile (figura 3) híbrido para acompanhar a chamada on-line.

A interface web 'Cadastro de Pacientes' apresenta uma barra superior com o título e uma mensagem de status: 'Carregando dados do paciente : Hudson'. O layout é dividido em três seções principais. À esquerda, há um cartão de perfil com uma imagem de perfil (um ícone de cabeça com um estetoscópio) e estatísticas: 'Meu guiche' (2), 'Senha atual' (151) e 'Total Cad.' (4). Abaixo, uma lista 'Pacientes Cadastrados' mostra Hudson, Elaine, Joana e Stephany, cada um com um ícone de perfil e um botão de lupa. À direita, o formulário de cadastro contém campos para: Empresa (desativado), Email (hudymoreira@gmail.com), Nome (Hudson) e Sobrenome (Moreira Guimarães dos Santos), Endereço (Rua Nilo, 285), Cidade (Embu das Artes), Estado (São Paulo) e CEP (0681040). Há também um campo de Observações com o texto 'Consulta com a Dra Elaine'. No rodapé do formulário, há botões 'Salvar' e 'cancelar'.

(Figura 1: Cadastro)



(Figura 2: Chamada)



(Figura 3: APP- Chamada)

BASE DO PROJETO:

Para a criação deste projeto usaremos um template pronto (Free) feito pela Creative Tim (<https://www.creative-tim.com>).

O download do template "COMPLETO" pode ser feito no link: <https://www.creative-tim.com/product/paper-dashboard-angular>.

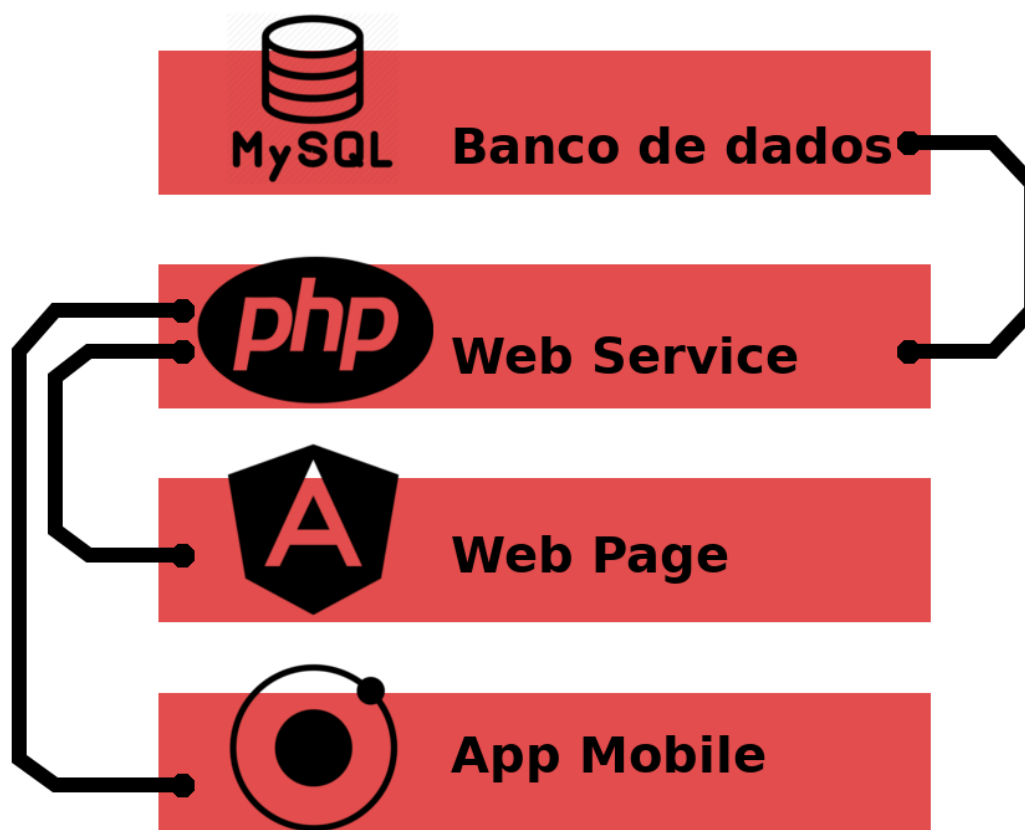
O download do template modificado para acompanhar os passos dessa apostila está disponível no git hub no link: *****

O download da aplicação pronta está disponível no github também nos links:

Projeto Angula:*****

Projeto Ionic:*****

ESQUEMA DO CURSO:



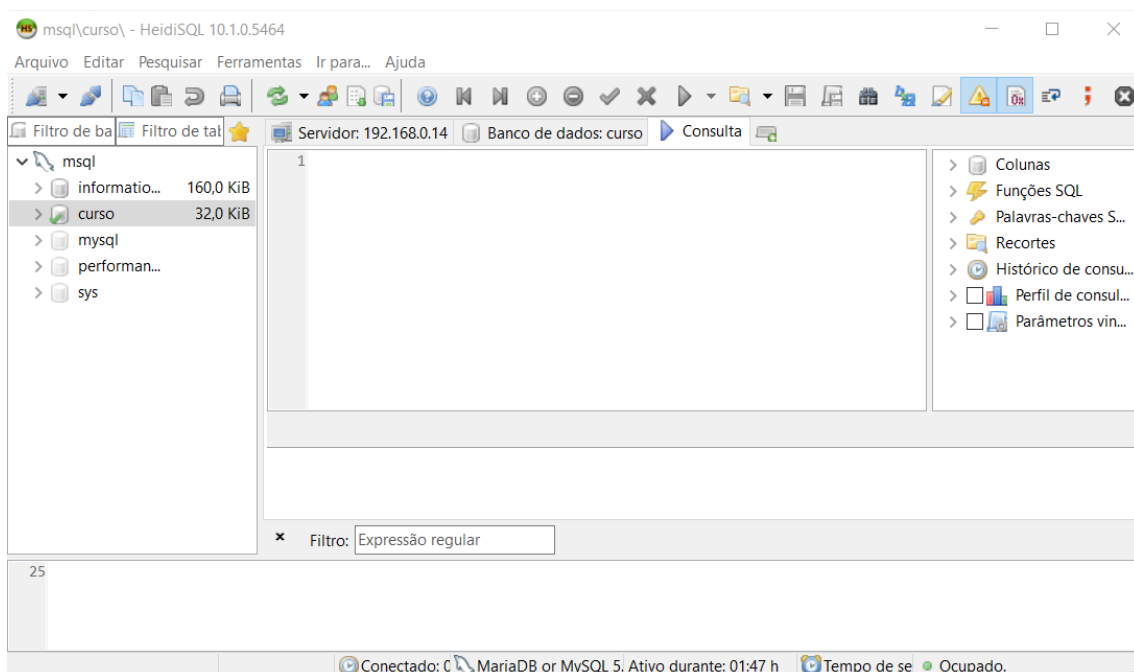
Seguiremos essa ordem:

1. Banco de dados com Mysql;
2. web service com PHP;
3. webPage Angular;
4. Aplicativo com Ionic.

Ressalto que a configuração do servidor web, do banco de dados, do Visual Studio Code e do node.js é por sua conta, existem muitos tutoriais na web que te ajuda a configurar esses recursos no seu computador sem muita dificuldade. Como o Objetivo é montar uma aplicação simples e funcional, essa apostila ficaria muito grade e o curso ficaria muito complexo. Dado essas introduções, podemos começar.



No curso o aluno recebe uma instancia do banco de dados semelhante ao que se é adquirido em um provedor (como HostGator ou LocaWeb) e a ferramenta cliente usada será o Heidi SQL. No exemplo curamos um Banco de dados chamado “curso”, mas provavelmente a instancia do banco terá o nome do aluno.



Para o nosso sistema de cadastro criaremos duas tabelas apenas. Tabela [chamada] e tabela [paciente].

```
CREATE TABLE `chamada` (  
  `idChamada` INT(11) NOT NULL AUTO_INCREMENT,  
  `guiche` INT(11) NOT NULL,  
  `dataRegistro` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  PRIMARY KEY (`idChamada`)  
)  
COLLATE='latin1_swedish_ci'  
ENGINE=InnoDB  
AUTO_INCREMENT=164  
;
```

```

CREATE TABLE `paciente` (
  `idPaciente` INT(11) NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(200) NULL DEFAULT NULL,
  `nome` VARCHAR(200) NULL DEFAULT NULL,
  `snome` VARCHAR(200) NULL DEFAULT NULL,
  `endereco` VARCHAR(200) NULL DEFAULT NULL,
  `cidade` VARCHAR(200) NULL DEFAULT NULL,
  `estado` VARCHAR(200) NULL DEFAULT NULL,
  `cep` VARCHAR(200) NULL DEFAULT NULL,
  `obs` TEXT NULL,
  `dataRegistro` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`idPaciente`)
)
COLLATE='latin1_swedish_ci'
ENGINE=InnoDB
AUTO_INCREMENT=8
;

```

Criaremos também 3 stored procedures chamadas (setChamada), (setPaciente) e getChamada:

```

DELIMITER $$
CREATE PROCEDURE `setChamada`(IN `guiche` INT)
BEGIN
  INSERT INTO chamada (`guiche`) VALUES (guiche);
  SELECT LAST_INSERT_ID() as senha;
END
$

```

```

DELIMITER $$
CREATE PROCEDURE `setPaciente`(
  IN `email` VARCHAR(200), IN `nome` VARCHAR(200),
  IN `snome` VARCHAR(200), IN `endereco` VARCHAR(200),
  IN `cidade` VARCHAR(200), IN `estado` VARCHAR(200),
  IN `cep` VARCHAR(200), IN `obs` VARCHAR(200)
)
BEGIN
  INSERT INTO paciente (
    `email`, `nome`, `snome`, `endereco`, `cidade`, `estado`, `cep`, `obs`)
  VALUES
    (email , nome , snome , endereco , cidade , estado , cep , obs);
END
$

```

```

DELIMITER $$
CREATE PROCEDURE `getChamada`()
BEGIN
  SELECT idChamada, guiche, DATE_FORMAT(dataRegistro, '%m/%d/%Y
%H:%m:%s') AS dataRegistro FROM chamada ORDER BY idChamada DESC
  LIMIT 10;
END
$

```



Assim como a instancia do banco de dados, o aluno receberá uma cessão do apache onde ele depositará o seu "index.php" contendo todas as instruções necessárias para servir tanto a aplicação Angular quanto a aplicação Ionic. O Editor de texto utilizado será o NotPad++, ele tem um recurso de ressaltar a sintaxe do texto que facilita o entendimento do código.

Abra o editor de texto e comece a codificar. Primeiro começaremos com o cabeçalho, as diretrizes aqui inseridas são necessárias para a requisição http do Angular e do Ionic.

```
<?php
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, POST, PATCH, PUT, DELETE, OPTIONS');
header('Access-Control-Allow-Headers: Origin, Content-Type, X-Auth-Token');
```

Para conectar no banco de dados usaremos o PDO (PHP Data Object), mas primeira configuraremos as variáveis de conexão:

```
$srvdb    = "mysql:host=192.168.0.14;dbname=curso";
$usuario  = "root";
$senha    = "master";
```

E agora a criação do objeto de conexão em uma variável global

```
$con = new PDO($srvdb, $usuario, $senha,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\''));
```

Função que inserir o Paciente:

```
function insertPaciente($paciente) {
    $valor = array();
    foreach($paciente as $k => $v){
        $valor[] = $v;
    }
    $sql= $GLOBALS['con']->prepare("CALL setPaciente (?,?,?,?,?,?,?,?) ");
    $sql->execute($valor);
    $r = $sql->errorInfo();
    return $r;
}
```

Função que faz uma chamada:

```
function fazerChamada($guiche) {  
    $sql = $GLOBALS['con']->prepare("CALL setChamada (?)");  
    $sql->execute(array($guiche));  
    $r = $sql->fetchAll(PDO::FETCH_ASSOC);  
    return $r;  
}
```

Essas são as funções principais, vamos testá-las e em seguida criaremos as demais funções para que completem a regra de negócio da aplicação. Primeiro a testaremos a função insertPaciente(). Ela receberá um array associativo, se você reparar bem, ela percorrer todos os itens desse array separando somente os valores e descartando a chave montando um array simples para que seja passado para o PDO, que usa o “prepare” garantindo que seja executado a nossa query corretamente.

```
$paciente = array(  
    "email"    =>"hudymoreira@gmail.com",  
    "nome"     =>"Hudy",  
    "snome"    =>"Moreira",  
    "endereco"=>"Rua Nilo",  
    "cidade"   =>"Embu das Artes",  
    "estado"   =>"São Paulo",  
    "cep"      =>"06810410",  
    "obs"      =>"Zumbisouro");  
  
$r = insertPaciente($paciente);  
print_r($r);
```

Para fazer a chamada é só passar o número do guichê:

```
$r = fazerChamda(2);  
Print_r($r);
```


E agora as duas funções restantes:

```
function getPacientes() {
    $sql = $GLOBALS['con']->prepare("select * from paciente ");
    $sql->execute();
    $r = $sql->fetchAll(PDO::FETCH_ASSOC);
    return $r;
}
function getChamada () {
    $sql = $GLOBALS['con']->prepare("CALL getChamada()");
    $sql->execute();
    $r = $sql->fetchAll(PDO::FETCH_ASSOC);
    return $r;
}
```

Faremos agora a regra de negócio da aplicação, vamos atender tanto POST quanto GET vindos do Angular e do ionic, vamos ver linha por linha:

Primeiro tentamos pegar o conteúdo vindo da requisição POST vindo da Content-Type:

```
$dados = file_get_contents('php://input');
```

Depois declaramos uma variável com valor 'false' e verificamos se \$dados não está vazia, depois decodificamos o json para um array associativo:

```
$decode = false;
if (strlen($dados) > 0) {
    $decode = json_decode($dados, true);
}
```

Agora trataremos de interpretar as opções vinda da aplicação, nesse caso será o que receberemos do Angular:

```
switch ($decode['opt']) {
    case 1: echo json_encode(getPacientes ()); ;break;
    case 2: echo json_encode(insertPaciente($decode['paciente'])); ;break;
    case 3: echo json_encode(fazerChamada ($decode['guiche'])); ;break;
    case 4: echo json_encode(getChamada ()); ;break;
}
```

E agora atenderemos uma requisição GET que vira do Ionic, caso ela seja válida será retornado a chamada:

```
if (isset($_GET['evento'])) {  
    if($_GET['evento'] == "chamada") {  
        echo json_encode(getChamada());  
    }  
}
```

Com isso concluímos nosso web service, ele atenderá todas as requisições da nossa aplicação, algo a mais fica por conta da sua criatividade.



Web Page

Começamos com o download do template angular feito sob medida para esta apostila. Para isso é necessário o GIT (4) instalado no seu computador.

Neste exemplo usarei o PowerShell como prompt de comando e o Visual Studio Code como editor, porém você pode usar qualquer ferramenta que possibilite fazer atividades semelhantes as que virão a seguir.

Download do projeto:

```
PS C:\Users\hudy\Documents> git clone https://github.com/hudymoreira/angular-curso.git
Cloning into 'angular-curso'...
remote: Enumerating objects: 141, done.
remote: Counting objects: 100% (141/141), done.
remote: Compressing objects: 100% (128/128), done.
remote: Total 141 (delta 8), reused 141 (delta 8), pack-reused 0 receiving objects:
86% (122/141), 372.00 KiB | 730.00 KiB/s
Receiving objects: 100% (141/141), 695.06 KiB | 828.00 KiB/s, done.
Resolving deltas: 100% (8/8), done.
PS C:\Users\hudy\Documents>
```

Entre no diretório angular-curso e instale todas as dependências do projeto com o npm

```
PS C:\Users\hudy\Documents> cd .\angular-curso\
PS C:\Users\hudy\Documents\angular-curso> npm install
```

Testando a aplicação:

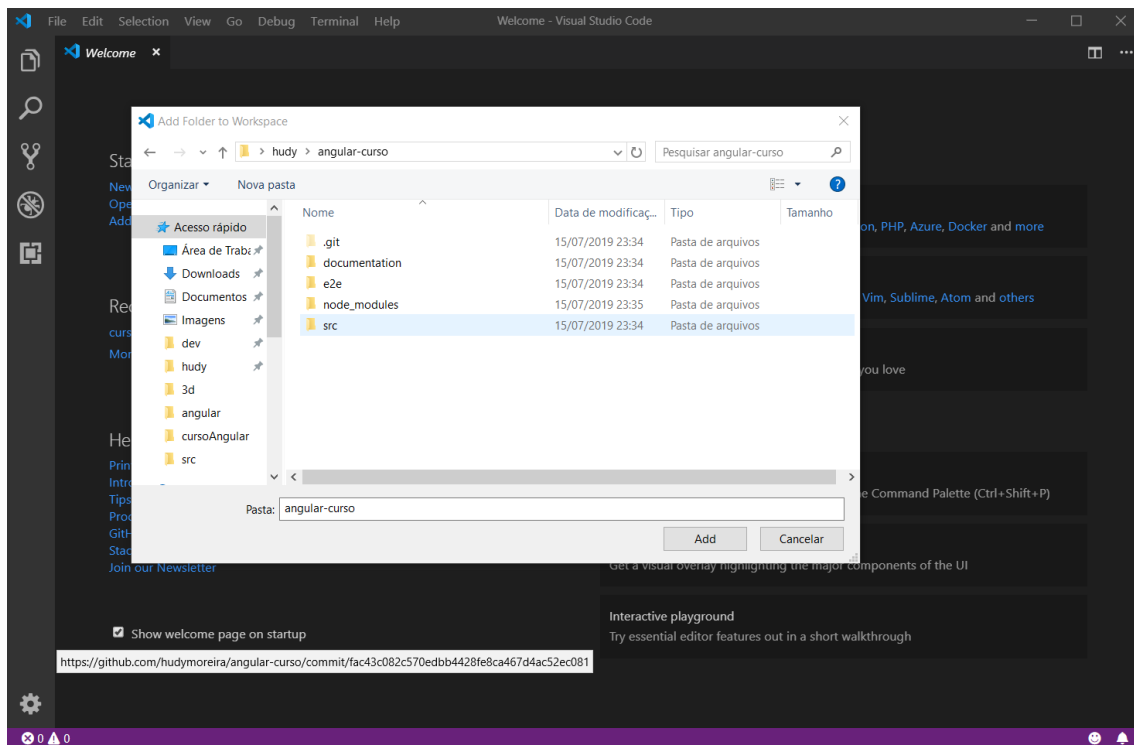
```
PS C:\Users\hudy\Documents\angular-curso> ng serve
Your global Angular CLI version (8.0.3) is greater than your local
version (1.4.2). The local Angular CLI version is used.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".
** NG Live Development Server is listening on localhost:4200, open your browser on
http://localhost:4200/ **
Date: 2019-07-13T04:32:04.229Z
Hash: ebd6642957d30f576e36
Time: 7948ms
chunk {inline} inline.bundle.js, inline.bundle.js.map (inline) 5.83 kB [entry] [rendered]
chunk {main} main.bundle.js, main.bundle.js.map (main) 55.1 kB {vendor} [initial] [rendered]
chunk {polyfills} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 193 kB {i
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 113 kB {inline} [
chunk {vendor} vendor.bundle.js, vendor.bundle.js.map (vendor) 3.08 MB [initial] [rendered]

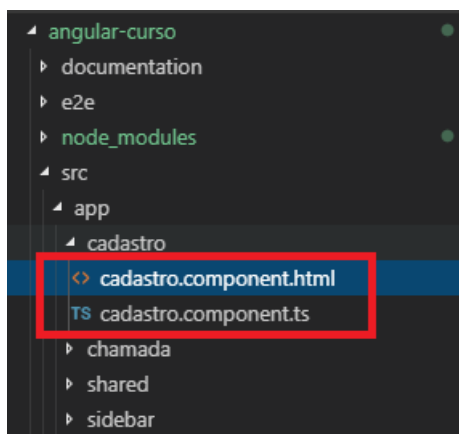
webpack: Compiled successfully.
```

A partir deste ponto podemos continuar a editar o nosso projeto Angular no Visual Studio Code.

Abra o Visual Studio Code, em seguida clique em “File” e depois em “Add Folder to Work” abra a pasta do seu projeto Angular (no meu caso fica em “C:\Users\hudy\angular-curso”) e finalize essa etapa clicando em “Add”.



Depois de carregado sua aplicação no Visual studio code, se preferir, é possível executar a aplicação direto por aqui, basta clicar no menu: [Viwer]>>[Terminal]. No rodapé do editar você tem acesso ao terminal onde você pode executar o comando [ng serve].



Agora começamos a programar nosso formulário de cadastros.

Expandindo as pastas você terá que encontrar esses dois arquivos:

- cadastro.component.html;
- cadastro.component.ts ;

Vamos começar criando o objeto dentro de cadastro.component.ts que terá todos os dados do paciente.

```
export class CadastroComponent implements OnInit{  
  public paciente : any;
```

```
  ngOnInit(){  
    this.paciente = {  
      email : null,  
      nome : null,  
      snome : null,  
      endereco : null,  
      cidade : null,  
      estado : null,  
      cep : null,  
      obs : null  
    }  
  }  
}
```

Vamos iniciar esse objeto com os campos pré-definidos:

No HTML faça a associação de todos os campos como nesse exemplo do Email. Com isso já temos o suficiente para mandar nosso cadastro para o nosso web service.

```
<input  
  type="email"  
  class="form-control border-input"  
  placeholder="Email"  
  [(ngModel)]="paciente.email" >
```

```
<button  
  class = "btn btn-info btn-fill btn-wd"  
  (click) = "cadastrar()">Salvar</button>
```

No botão salvar, crie o evento (click) com a função

cadastrar.

Ok, agora vamos preparar o envio dos dados e criar uma porção de procedimento antes de mandar o cadastro do paciente para o nosso banco de dados.

Primeiro importamos as classes necessárias:

```
import { HttpClient,HttpHeaders } from '@angular/common/http';
```

Depois criamos a variável de cabeçalho:

```
public headers = new HttpHeaders();
```

Agora adicione parâmetro http no método construtor da classe e depois inicie o cabeçalho:

```
constructor(public http : HttpClient){  
  this.headers = this.headers.set("Content-Type","application/json; charset=UTF-8");  
}
```

```

cadastrar(){
  let opt : any = {
    opt : 2,
    paciente : this.paciente
  }

  this.http.post(environment.apiUrl,
    JSON.stringify(opt), {headers: this.headers}
  ).subscribe(data => {
    if (data != null){
      console.log(data);
    }
  });
}

```

E por fim você o método de envio de dados que atende as especificações do nosso web service.

Essa é a base básica para o envio, no próximo passo faremos a requisição da chamada para o próximo paciente no guichê de cadastros.

```

public resumo : any = {
  meuGuiche : 0,
  senhaAtual : 0,
  totalCadastrado : 0
}

```

Configurando a chamada:

Primeiro de tudo crio um objeto chamado “resumo” que terá dados relevante ao status do guichê.

```

<select [(ngModel)]="resumo.meuGuiche" >
  <option value="1">1</option>
  <option value="2">2</option>
  <option value="3">3</option>
  <option value="4">4</option>
  <option value="5">5</option>
</select>

```

No seletor contido no arquivo HTML associamos a variável correspondente ao guichê atual.

```

<div class="author" (click)="proximo()">

```

função próximo()).

Também definimos o evento click para a

```

proximo(){
  let post : any = {
    opt : 3,
    guiche : this.resumo.meuGuiche
  }
  this.http.post(environment.apiUrl,
    JSON.stringify(post),
    {headers: this.headers}).subscribe(data => {
    if (data != null){
      console.log(data)
    }
  });
}

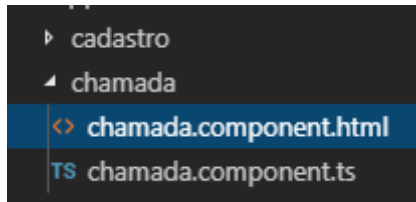
```

Na função próximo mandamos o guichê que está solicitando a chamada e recebemos de volta do servidor a próxima senha.

Com isso a aplicação é capaz de transmitir dados para o web service, você pode usar sua criatividade para melhorar a relação da sua aplicação com o armazenamento de dados na nuvem.

Agora para finalizar essa parte faremos o painel de chamada.

Vamos editar os arquivos .ts e .html da chamada.



```
public chamada : any = {  
  numero : 0,  
  guiche : 0,  
  hora : '0'  
}
```

Em chamada.component crie o objeto que receberá os dados da chamada.

Associe as variáveis no seu arquivo html.

```
<div class="content">  
  <h4 class="title">Numero Atual</h4>  
  <div class="a"> {{chamada.numero}} </div>  
  <hr /><hr />  
  <h4 class="title">Guiche</h4>  
  <div class="a"> {{chamada.guiche}} </div>  
  <div class="footer">  
    <div class="stats">  
      <i class="ti-timer"></i> Chamado as: {{chamada.hora}}  
    </div>  
  </div>  
</div>
```

E agora a função que busca a chamada:

```
getChamada(){
  this.http.post(environment.apiUrl,
    JSON.stringify({opt:4}),{headers: this.headers}).subscribe((data:any) => {
    if (data != null){
      if (data[0] != undefined){
        if ( data[0].idChamada != this.chamada.numero){
          this.alarme();
          this.chamada.numero = data[0].idChamada;
          this.chamada.guiche = data[0].guiche;
          this.chamada.hora = data[0].dataRegistro;
          this.listaChamada.length = 0;
          for (let item of data){
            this.listaChamada.push([item.guiche,item.idChamada,item.dataRegistro]);
          }
        }
      }
    }
  });
}
```

E agora faremos o loop que verifica a 5 segundos:

```
constructor( public http : HttpClient) {
  this.headers = this.headers.set("Content-Type","application/json; charset=UTF-8");
  setInterval(() => this.getChamada(), 5000);
}
```

Com isso, toda vez que um guichê chamar uma nova senha será apresentada no painel e assim concluímos essa parte, o que fizemos até foi o básico para tudo funcionar, no fim da apostila deixo o link para a versão full desse projeto com algumas implementações extras, e agora seguimos com o projeto no aplicativo.



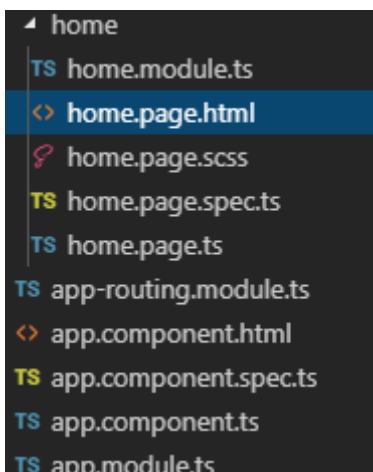
App Mobile

Chegou a hora de desenvolver o aplicativo, faremos isso do zero, primeiro de tudo vamos usar o terminal para criar um projeto ionic.

```
PS C:\Users\hudy\dev\apostila> ionic start curso-ionic blank
```

Depois de criado entre na pasta e teste sua aplicação no simulador

```
PS C:\Users\hudy\dev\apostila> cd .\curso-ionic\  
PS C:\Users\hudy\dev\apostila\curso-ionic> ionic serve --lab
```



Abra o projeto com o Visual Studio Code e prepare o HTML, CSS e o TS como os exemplos a seguir, essa parte é bem básica por isso só tem o exemplo mesmo, nessa parte o formulário somente passa para a outra página o ip (endereço) do web service e a sua senha atual para comparação.

HTML

```
<ion-content padding>
  <div>
    <ion-grid>
      <ion-row>
        <ion-item>
          <ion-label position="floating">Servidor</ion-label>
          <ion-input type="url" [(ngModel)]="dados.servidor"></ion-input>
        </ion-item>
      </ion-row>
      <ion-row>
        <ion-item>
          <ion-label position="floating">Numero da senha</ion-label>
          <ion-input type="number" [(ngModel)]="dados.senha"></ion-input>
        </ion-item>
      </ion-row>
    </ion-grid>
    <ion-row>
      <ion-button size="large" expand="full" (click)="chamada()">OK</ion-button>
    </ion-row>
  </div>
</ion-content>
```

CSS

```
ion-content {
  div {
    height: 100%;
    width: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
  }
}
```

TS

```
import { Component } from '@angular/core';
import { Router, NavigationExtras } from '@angular/router';
@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage {
  public dados = {
    senha: '',
    servidor: ''
  }
  constructor(private router: Router) {}
  chamada(){
    let navigationExtras: NavigationExtras = {
      state: {
        dados: this.dados
      }
    };
    this.router.navigate(['chamada'], navigationExtras)
  }
}
```

Vamos criar a página de chamada que consumirá com nosso web service, no terminal digite o seguinte comando:

```
PS C:\Users\hudy\dev\apostila> ionic generate page chamada
```

Após criada a página, no Visual Studio Code vamos editar os arquivos.

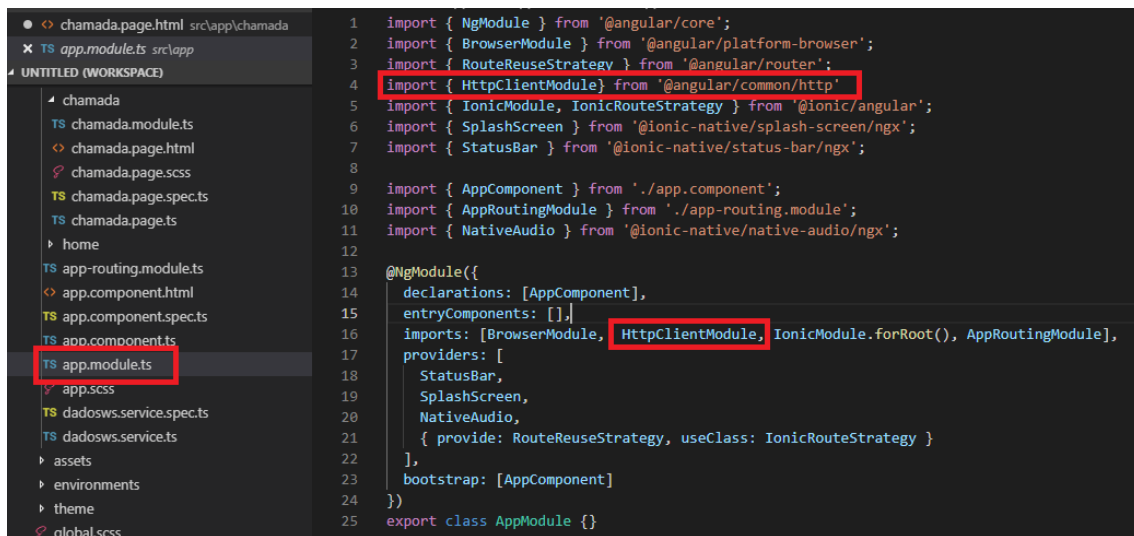
HTML

```
<ion-header>
  <ion-toolbar>
    <ion-title>chamada</ion-title>
  </ion-toolbar>
</ion-header>
<ion-content>
  <ion-card>
    <ion-card-header>
      <ion-card-subtitle>minha senha : {{dados.senha}}</ion-card-subtitle>
      <ion-card-title>chamada</ion-card-title>
    </ion-card-header>
    <ion-card-content>
      <ion-col class="font">{{chamada.senha}}</ion-col>
    </ion-card-content>
    <ion-card-header>
      <ion-card-title>Guiche</ion-card-title>
    </ion-card-header>
    <ion-card-content>
      <ion-col class="font2">{{chamada.guiche}}</ion-col>
    </ion-card-content>
    <ion-button expand="full" (click)="voltar()">voltar</ion-button>
  </ion-card>
</ion-content>
```

CSS

```
ion-content {
  div {
    height: 100%;
    width: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
  }
  ion-card{
    width: 90%;
    justify-content: center;
    align-items: center;
  }
}
.font{
  font-size:48px;
}
.font2{
  font-size:58px;
}
```

Antes de editarmos o TS do componente de chamada teremos que configurar o app.module que permitirá que o seu aplicativo possa fazer requisições via post e get através de uma url:



```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { RouteReuseStrategy } from '@angular/router';
4 import { HttpClientModule } from '@angular/common/http';
5 import { IonicModule, IonicRouteStrategy } from '@ionic/angular';
6 import { SplashScreen } from '@ionic-native/splash-screen/ngx';
7 import { StatusBar } from '@ionic-native/status-bar/ngx';
8
9 import { AppComponent } from './app.component';
10 import { AppRoutingModuleModule } from './app-routing.module';
11 import { NativeAudio } from '@ionic-native/native-audio/ngx';
12
13 @NgModule({
14   declarations: [AppComponent],
15   entryComponents: [],
16   imports: [BrowserModule, HttpClientModule, IonicModule.forRoot(), AppRoutingModuleModule],
17   providers: [
18     StatusBar,
19     SplashScreen,
20     NativeAudio,
21     { provide: RouteReuseStrategy, useClass: IonicRouteStrategy }
22   ],
23   bootstrap: [AppComponent]
24 })
25 export class AppModule {}
```

Depois de dado o acesso, vamos criar um serviço responsável por se conectar no web service chamdo “ws”:

```
PS C:\Users\hudy\dev\apostila\curso-ionic> ionic generate service dadosws
> ng.cmd generate service ws
CREATE src/app/ws.dadoservice.spec.ts (313 bytes)
CREATE src/app/ws.dadoservice.ts (131 bytes)
[OK] Generated service!
PS C:\Users\hudy\dev\apostila\curso-ionic>
```

Vamos editar o arquivo dadosws.service.ts:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class DadoswsService {

  constructor(public http: HttpClient) { }

  getChamada(servidor) {
    return new Promise((resolve, reject) => {
      this.http.get('http://' + servidor + '/?evento=chamada').toPromise()
        .then((dados) => {
          resolve(dados);
        }).catch((erro) => {
          reject(erro);
        })
    });
  }
}
```

```
export class ChamadaPage implements OnInit {
  public dados : any={
    senha :0,
    servidor :''
  }
  public chamada : any = {
    senha : 0,
    guiche : 0
  }
}
```

Com o serviço criado já podemos finalizar o .ts da chamada. Começamos criando dois objetos que receberão dados da página anterior e a atualização da chamada:

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { DadoswsService } from '../dadosws.service';
```

Antes de configurar o método construtor, importe as classes que permitam receber dados da página anterior e a do

web service:

```
getChamada(servidor){
  this.ws.getChamada(servidor).then(chamada=>{
    if (chamada[0] != undefined){
      this.chamada.senha = chamada[0].idChamada;
      this.chamada.guiche = chamada[0].guiche;
    }
  });
}
```

Antes de configurar o construtor para receber todos os dados e fazer o loop de verificação no servidor, criaremos agora o método de pega os dados primeiro.

E agora o construtor:

```
constructor(  
  private route: ActivatedRoute,  
  private router: Router ,  
  public ws : DadoswsService) {  
  this.route.queryParams.subscribe(params => {  
    if (this.router.getCurrentNavigation().extras.state) {  
      this.dados = this.router.getCurrentNavigation().extras.state.dados;  
      setInterval(function(){ this.getChamada(this.dados.servidor);}.bind(this), 3000);  
    }  
  });  
}
```

Se os parâmetros vierem corretamente da página anterior ele faz um loop e verifica a chamada a cada 3 segundos.

E para finalizar, faremos a função do botão “voltar”:

```
voltar(){  
  this.router.navigate(['home']);  
}
```

E assim concluímos nossa aplicação, abaixo links para a aplicação completa angular e ionic:

Angular: <https://github.com/hudymoreira/angular-professor>

Ionic : <https://github.com/hudymoreira/ionic-professor>